TUCS

Simon Holmbacka

Energy Aware Software for Many-Core Systems

Turku Centre *for* Computer Science

TUCS Dissertations
No 208, December 2015

Energy Aware Software for Many-Core Systems

Simon Holmbacka

To be presented, with the permission of the Faculty of Science and Engineering of the University of Åbo Akademi, for public criticism in Auditorium Granö-sali on December 11, 2015, at 12 noon.

Åbo Akademi University
Faculty of Science and Engineering
LT1, Universitetsbacken, 20520 Åbo, Finland

2015

Supervisor

Professor Johan Lilius
Faculty of Science and Engineering
Åbo Akademi University
Joukahainengatan 3-5, 20520 Åbo
Finland

Advisor

Doc. Dr. Sébastien Lafond
Faculty of Science and Engineering
Åbo Akademi University
Joukahainengatan 3-5, 20520 Åbo
Finland

Reviewers

Professor Mats Brorsson
Avdelningen för Programvaruteknik och Datorsystem
Kungliga Tekniska Högskolan
SE-100 44 Stockholm
Sweden

Professor Jean-François Nezan
Institut d'Electronique et de Télécommunications de Rennes
Institut National des Sciences Appliquées
20 Avenue des Buttes de Coësmes, Rennes
France

Opponent

Professor Christoph Kessler
Department for Computer and Information Science
Linköping University
SE - 581 83 Linköping
Sweden

ISBN 978-952-12-3305-0
ISSN 1239-1883

“*Wenn es Ihnen beim Studium der Quantenmechanik nicht schwindelig wird, dann haben Sie sie nicht wirklich verstanden.*”

— Niels Bohr

Abstract

Many-core systems provide a great potential in application performance with the massively parallel structure. Such systems are currently being integrated into most parts of daily life from high-end server farms to desktop systems, laptops and mobile devices. Yet, these systems are facing increasing challenges such as high temperature causing physical damage, high electrical bills both for servers and individual users, unpleasant noise levels due to active cooling and unrealistic battery drainage in mobile devices; factors caused directly by poor energy efficiency.

Power management has traditionally been an area of research providing hardware solutions or runtime power management in the operating system in form of frequency governors. Energy awareness in application software is currently non-existent. This means that applications are not involved in the power management decisions, nor does any interface between the applications and the runtime system to provide such facilities exist. Power management in the operating system is therefore performed purely based on indirect implications of software execution, usually referred to as the workload. It often results in over-allocation of resources, hence power waste.

This thesis discusses power management strategies in many-core systems in the form of increasing application software awareness of energy efficiency. The presented approach allows meta-data descriptions in the applications and is manifested in two design recommendations:

1) *Energy-aware mapping*

2) *Energy-aware execution*

which allow the applications to directly influence the power management decisions. The recommendations eliminate over-allocation of resources and increase the energy efficiency of the computing system. Both recommendations are fully supported in a provided interface in combination with a novel power management runtime system called *Bricktop*. The work presented in this thesis allows both new- and legacy software to execute with the most energy efficient mapping on a many-core CPU and with the most energy efficient performance level. A set of case study examples demonstrate real-world energy savings in a wide range of applications without performance degradation.

Sammandrag

Mångkärniga datorsystem har en hög prestandapotential tack vare dess massivt parallella hårdvarustruktur. Dessa datorsystem integreras för tillfället i de flesta delar av vårt vardagliga liv. Allt från storskaliga serverfarmer till persondatorer, bärbara datorer och mobila enheter. Utmaningarna för sådana system har dock ökat i form av höga processortemperaturer som ökar energikostnader, orsakar obekväma ljudnivåer på grund av aktiv nerkylning och orealistisk batterianvändning i mobila enheter. Dessa faktorer är en direkt konsekvens av otillräcklig energieffektivitet.

Strömhantering har traditionellt sett varit en forskningsfråga för hårdvaruområden, eller som en inbyggd funktionalitet i operativsystemet. Energimedvetenhet i applikationer som sådan existerar inte. Detta betyder att applikationerna inte deltar i beslut som fattas inom strömhanteringen. Det finns för tillfället inte heller något gränssnitt mellan applikationerna och operativsystemet som kunde erbjuda en sådan tjänst. Strömhantering i operativsystemet genomförs därför endast baserad på indirekta implikationer orsakad av mjukvaruexekveringen, som hänvisas till som systemets arbetsbörda (workload). Detta sätt att genomföra strömhantering resulterar ofta i överallokering av resurser och därmed i slöseri med ström.

Denna avhandling framlägger strömhanteringsstrategier i mångkärniga datorsystem genom energimedveten mjukvara. Strategierna tillsammans med direkta implementationer låter applikationer inkludera meta-data som används för att göra beslut för strömhanteringen. Avhandlingen konkretiserar detta med hjälp av två designrekommendationer:
1) *Energimedveten applikationsfördelning*
2) *Energimedveten applikationsexekvering*
som låter applikationerna medverka i beslut som fattas för strömhantering. Rekommendationerna minimerar överallokering av resurser och ökar energieffektiviteten i datorsystemet. Båda rekommendationerna stöds till fullo genom ett implementerat gränssnitt ihopkopplat med en ny typ av strömhanterare kallad *Bricktop*. Arbetet som presenteras i denna avhandling låter både nya och redan implementerade applikationer exekvera på ett optimalt antal processeringselement, och med optimal prestanda för att minimera energikonsumtionen. Exempel från fallstudier visar energiinbesparingar vid användning av olika typer av applikationer utan att prestandadegradering.

Acknowledgements

The work in this thesis was carried out in the Faculty of Science and Engineering at Åbo Akademi University between April 2011 and April 2015.

Firstly, I would give my most expression of gratitude to my supervisor and friend Johan Lilius for believing in my efforts and who has given me motivation and excellent support for completing this thesis. I would like to thank my opponent Christoph Kessler and reviewers Mats Brorsson and Jean-François Nezan for giving me constructive and helpful pointers to my work, and for great discussions outside of the work of my thesis.

I am very grateful to my co-workers and co-authors. First and foremost my advisor Sébastien Lafond who has, with great patience, consulted my research to continuously improve the material. The field of computer engineering always requires programming and people who know how systems actually work. For helping me in this effort, I would like to give thanks to Wictor Lund, Stefan Grönroos and Dag Ågren who have saved me many Google hours by sharing their expertise.

Cooperation is the key to success, and I would like to thank all my co-authors who contributed to my publications making this thesis possible: My French friends in the signal processing team: Erwan Nouges, Maxime Pelcat and Daniel Menard. My German power management colleagues in Hagen: Jörg Keller and Patrick Eitschberger. My Spanish safety-critical experts: Jose-Luis Gutierrez and Javier Diaz. My former in-house Linux hackers Fredric Hällis and Robert Slotte. My System-on-Chip colleagues from the Finnish side of the glass door in the ICT building Amir-Mohammad Rahmani-Sane and Mohammad Fattah. To all other people in the ESlab, thanks for a great time, let's keep up this kind of atmosphere!

I would like to thank the very competent and friendly administration in the ICT building handling everything between fixing traveling arrangements and providing SD cards. The thanks goes to Nina Hultholm, Christel Englom, Tove Österroos, Joakim Storrank, Karl Rönnholm, Marat Vagapov, Tomi Mäntylä, Susanne Ramstedt and Solveig Vaherkylä. No task is too small or too big for these guys.

Many thanks goes to the former IT-department at Åbo Akademi and the TUCS platform for graduate students. I would like to acknowledge Tekniikan edistämissäätiö, Svenska tekniska vetenskapsakademien i Finland and Otto A. Malms donationsfond for external funding leading to the completion of this thesis.

Finally, I must thank my parents Inga and Dennis and their everlasting support. You have always been my motivating factor no matter the task. To my grandmother, I must announce that my studies are finally completed: *böv int sita nameir i skolbentschin!* Most importantly, my dedicated support comes from my wife Selina; not only by her motivation but also with her insight into the importance of energy efficiency and ecologically sustainable development. I hope that my thesis will contribute to these areas not only for my own interest but also for future generations.

Simon Holmbacka
Åbo
November 2, 2015

List of Original Publications

1. Simon Holmbacka, Sébastien Lafond, Johan Lilius. Power Proportional Characteristics of an Energy Manager for Web Clusters. In *Proceedings on Embedded Computer Systems: Architecture, Modeling and Simulation, 2011 IEEE International Conference*, pages 51–58, Samos, Greece.

2. Simon Holmbacka, Sébastien Lafond, Johan Lilius. A PID-Controlled Power Manager for Energy Efficient Web Clusters. In *Proceedings of the International Conference on Cloud and Green Computing, 2011 IEEE International Conference*, pages 712–728, Sydney, Australia.

3. Simon Holmbacka, Mohammad Fattah, Wictor Lund, Amir-Mohammad Rahmani, Sébastien Lafond, Johan Lilius. A Task Migration Mechanism for Distributed Many-Core Operating Systems. In *The Journal of Supercomputing, 2014 Springer*, pages 1141–1162.

4. Fredric Hällis, Simon Holmbacka, Wictor Lund, Robert Slotte, Sébastien Lafond, Johan Lilius. Thermal Influence on the Energy Efficiency of Workload Consolidation in Many-Core Architectures. In *Proceedings of the 24th Tyrrhenian International Workshop on Digital Communications, 2013 IEEE International Conference*, Genoa, Italy.

5. Simon Holmbacka, Dag Ågren, Sébastien Lafond, Johan Lilius. QoS Manager for Energy Efficient Many-Core Operating Systems. In *Proceedings of the 21st International Euromicro Conference on Parallel, Distributed and Network-based Processing, 2013, IEEE International Conference*, pages 318–322 Belfast, UK.

6. Simon Holmbacka, Erwan Nogues, Maxime Pelcat, Sébastien Lafond, Johan Lilius. Energy Efficiency and Performance Management of Parallel Dataflow Applications. In *The 2014 Conference on Design & Architectures for Signal & Image Processing [BEST PAPER], ECSI International Conference*, pages 133–141. Madrid, Spain.

7. Simon Holmbacka, Jörg Keller, Patrick Eitschberger, Johan Lilius. Accurate Energy Modelling for Many-Core Static Schedules. In *Proceedings of the 23rd International Euromicro Conference on Parallel, Distributed and Network-based Processing, 2015, IEEE International Conference*, pages 525–532 Turku, Finland.

8. Simon Holmbacka, Sébastien Lafond, Johan Lilius. Performance Monitor Based Power Management for big.LITTLE Platforms. In *Proceedings of the HIPEAC Workshop on Energy Efficiency with Heterogeneous Computing, 2015*, pages 1–6 Amsterdam, Netherlands.

Contents

Part I

Research Summary

Chapter 1

Overview of Original Publications

"*I can make it painless and perfect.*"
— Dr. John W. Thackery, The Knick - 2014

1.1 Paper I: Power Proportional Characteristics of an Energy Manager for Web Clusters

The paper primarily focused on the problem of increased energy consumption in modern server infrastructures. Computer systems in data centers alone stands for 1.5% of the European energy consumption and is steadily increasing. Measurements indicate a double in energy consumption between the years 2006 and 2011, and by 2020 the energy consumption increase in data centers in Western Europe is projected to over 100 TWh per year.

The idea in the paper, is to regulate resource usage in data centers according to the incoming request rate in order to make power dissipation proportional to the workload. The introduced term *power proportionality* is a measurement on the ability to regulate resources according to resource needs. Focus was put on sleep state based power management rather than DVFS because of the large scale system in mind and the ability to enable/disable any number of computing nodes at runtime. In order to increase the workload granularity and to enable better power proportionality, mobile CPUs were evaluated rather than high-end CPUs. By using smaller-but-many mobile CPUs, the same performance can be obtained as when using high-end CPUs, but the power dissipation of the running nodes can be more precisely tuned.

In order to manage the power dissipation and to evaluate the power proportionality, a simulation framework containing a PID controller is presented in the paper. The simulation framework is capable of generating web request patters either by using a synthetic pattern generator or by using real-world input. The framework uses the PID controller to determine how many nodes (since single-core systems are used; how many cores) should be used in order to handle all web requests within an acceptable time limit. The output of the PID controller determines the number of cores to wake up or shut down while taking the transition delay into account.

The results indicate that the tuning of the PID parameters are crucial to the efficiency of the controller with respect to the load patterns. The load patterns should therefore be analyzed in order to efficiently tune the controller at runtime with dynamic PID parameters. An example of runtime updatable parameters was presented in the work of Holmbacka [31], in which a PID controller in a video player is used for regulating the framerate. The PID parameters in the controller were updated during runtime to obtain a more stable playback. This paper is however not part of this thesis. In future work, more work was planned on the automatic tuning of PID parameters and the whether a combination of DVFS and sleep states can increase the power proportionality further.

Author's contribution: The co-authors suggested a single metric, power proportionality, to determine the relation between power dissipation and work executed. Based on the ideas and problem statements by the co-authors, the PID controller and the simulation framework was implemented by the author in order to simulate the level of power proportionality. The author and the co-author Sébastien Lafond discussed different use cases and scenarios for simulation after which the author performed the simulations. All authors interpreted the results of the simulations as well as drawing the concluding remarks on dynamic PID parameters. The author contributed with the paper write-up and presentation.

1.2 Paper II: A PID-Controlled Power Manager for Energy Efficient Web Clusters

From the directions in Paper I, the work continues in the field of energy efficient web clusters. Based on the fact that web servers operate on a 10 to 50 percent utilization rate, system level power management is crucial for future large scale web server installations.

The focus is, similarly to Paper I, set on low power nodes which offer a finer granularity of system level power control. The PID controlled manager is further developed, and more focus is put on the tuning of the PID parameters. Two different tuning methods are presented and evaluated in a multi-node simulation framework. Evaluations present both the energy reduction compared to a system using DVFS, and the impact on application QoS (Quality of Service) with respect to the tuning methods used.

The paper introduces the concept of *application performance* and how the metric is used to ensure sufficient QoS in applications. A web server was executed on a real platform to determine the possible request rate as a function of request size. The metric used measure performance was *requests per second*, according to which the power manager allocates the resources. From this concept we initiated the investigation on a generic performance parameter to which any application can relate its QoS, and what information is needed for a generic power manager controlling any kind of application.

Author's contribution: Based on received feedback in Paper I, the author investigated the parameter tuning and suggested two common tuning methods for evaluation. The author and co-authors concluded in frequent discussions that a parameter describing performance is required in order to drive the controller efficiently and in a more generic setting. The author contributed with the simulations and the paper write-up while the co-author Sébastien Lafond presented the paper.

1.3 Paper III: A Task Migration Mechanism for Distributed Many-Core Operating Systems

From the earlier work focusing on system level control, Paper III shifts focus to OS level control. While still targeting power management in a multi-node system, the OS level view offers a more detailed research in task execution on chip level, and its impact power dissipation. An executing task on a multi-core CPU will impose a certain workload level (usually expressed in the percentage of the core's capacity) on the core, which in turn dissipates power. The amount of power dissipated by a core depends primarily on the clock frequency, load level and temperature of the core.

The research focus on the practical aspects on moving tasks (called task migration) in a many-core system, and how to find a task distribution which dissipates the minimum amount of power with sufficient performance. Paper III is a journal extension to an earlier publication [32], in which the notion of network-on-chip processors were considered in addition to simple bus-based systems. In order to gain more knowledge of the mechanisms behind task migration, we implemented a task migrator on a multi-core real-time OS called FreeRTOS. In contrast to normal desktop OSes such as Linux or Windows, FreeRTOS is completely asynchronous and utilizes separate schedulers on each core. With a non-symmetric OS view, the state transfer of tasks is handled explicitly such as the transfer of task stack, heap, function references etc.

The evaluations were performed on two separate use-cases; the first of which used task migration to distribute high-resolution video player tasks to all cores in order to gain performance due to parallelization. The second use-case collected low-resolution video player tasks onto a single core in order to save power and still keep a steady framerate of 25 frames per second.

The main part of the publication is, however, the evaluation of the task migration overhead, i.e. the latency of migrating a task to another core. In our extended study we used both bus-based systems with a relatively predictable overhead and network-on-chip architectures with a more unpredictable overhead. Task migration in a network-on-chip simulator was executed to determine the confidence interval of the overhead with respect to migration distance and cache utilization. In the following papers, task migration will be used as one of the primary mechanisms for achieving energy efficiency on many-core systems.

Author's contribution: The task migration issue was discussed frequently by the author and co-authors in terms of OS scalability and power management. The decision to investigate the functionality behind task migration was made by the co-authors Johan Lilius and Sébastien Lafond, and the practical implementation on FreeRTOS was made by co-author Wictor Lund. The author mainly contributed to the theoretical foundation and the investigation of scalability in future many-core operating systems heavily reliant on task migration. Co-authors Mohammad Fattah and Amir-Mohammad Rahmani from University of Turku contributed to the journal extension and the network-on-chip simulations and measurements. The author contributed with the journal write-up.

1.4 Paper IV: Thermal Influence on the Energy Efficiency of Workload Consolidation in Many-Core Architectures

With under utilized computer systems, much CPU time is spend in idle mode with no workload to execute. While the cores are active but not executing workload, the system is simply wasting power keeping the core running while waiting for work to arrive. On the other hand, shutting down CPU cores can degrade the performance of the system during bursts of high workload. Optimally, the system should activate as many cores as needed while keeping the rest of the cores shut down.

The approach in Paper IV was to consolidate the workload onto as few cores as possible while keeping the rest of the cores shut down in order to save power. A new Linux scheduler (the Overlord Guided Scheduler) was implemented, which consolidated tasks to the highest loaded, non-overloaded core. With such an approach it is simple to disable the unused cores since the cores contain either high or no workload. Consolidating tasks onto few cores instead of distributing them improves on the locality in terms of task communication which involves less memory accesses. Less task migrations are also performed since the system is not forced to keep an evenly distributed schedule.

The scheduler was evaluated against the default CFS (Completely Fair Scheduler) in terms of energy consumption and performance. The evaluations consisted of applying a selected load level in several tasks on the system after which the energy was measured and the total execution time of the tasks. Results indicate that neither full load consolidation or completely fair scheduling is always the best approach depending on the workload. The optimal number of cores and the clock frequency is use is thus dependent on the type of tasks and their type of workload applied on the system.

Author's contribution: Load consolidation as an idea was suggested by the co-authors as shutting down the unused cores would improve on energy efficiency. The OGS scheduler was implemented and tested by the co-authors Fredric Hällis and Robert Slotte, who also contributed to the main writing part of the publication. The benchmark Spurg-Bench used in the evaluation of the scheduler was implemented by co-author Wictor Lund, and the evaluations were executed by co-author Fredric Hällis. The author created the main structure of the publication and wrote the discussions regarding power and energy in Section III and the workload mapping and task migration in Section IV. The paper was presented by co-author Fredric Hällis.

1.5 Paper V: QoS Manager for Energy Efficient Many-Core Operating Systems

Resource allocation in modern operating systems is usually based purely on the workload level. As the this level exceed a certain value, more resources are allocated in terms of clock frequency, number of cores etc. While the metric is generic to any type of application, it does not describe what kind of resources to allocate. For example an application could either use a higher clock frequency or an increased number of cores to increase performance. In order for the runtime system to perform the most energy efficient allocation, applications should describe what resources they are able to use most efficiently.

With Paper I and II primarily focusing on the controller, the extension of a generic interface between applications and resource allocation was discussed in Paper V. The presented QoS manager is a runtime resource allocator designed to intercept meta-data from applications in order to allocate resources in the most energy efficient manner. Applications are able to follow a declaration language to design a performance parameter specifically related to the application. The performance parameter is used together with a performance setpoint to steer the resource allocation. The QoS manager monitors the applications' performance periodically, and resource allocation/deallocation is performed in case of the performance is either under or above the setpoint.

The proof-of-concept QoS runtime system was implemented on top of FreeRTOS, and a multi-task jpeg decoder was evaluated. The jpeg decoder was set to inject a setpoint value describing its desired rate of picture decoding. For a selected picture rate, the QoS manager used sleep states on a multi-core platform to allocate resources. By adjusting the setpoint, the impact on the power dissipation of the system was noticed as unused resources could be shut down.

Author's contribution: In order to extend Paper I and II with a more generic performance parameter, the author suggested the more explicit declaration language as a way of tailor applications to the resource allocation. The author implemented the QoS runtime manager on FreeRTOS and the communication interface to the applications. Co-author Dag Ågren contributed with a FreeRTOS port for the multi-core ARM Cortex-A9 platform, and a core-to-core communications infrastructure. Furthermore, the author contributed with the necessary evaluations on the platform as well as the writing of the paper and the presentation.

1.6 Paper VI: Energy Efficiency and Performance Management of Parallel Dataflow Applications

Energy efficiency in computer systems is based on runtime allocation/deallocation of resources. Modern processors usually provide two methods of CPU resource allocation: Clock frequency scaling and CPU sleep states. Allocating resources for applications is required in order for applications to gain a desired performance. In contrast to the controller in Paper I and II which assumes a single input, single output, we extend the work with the notion of QoS requirements to allow the usage of both clock frequency scaling and sleep states as suggested in Paper V. The new multi input, multi output controller is minimizing the power dissipation of CPU by using a model for the power required to reach a given performance setpoint for the applications.

While clock frequency scaling increases performance relatively linear for any application, using sleep states to activate/deactivate cores is only useful if the application is programmed for parallel execution. To to reach a good balance between clock frequency and the number of active cores, the inherited parallelism in the applications must be known. A completely sequential application can only increase the speed-up by increasing the clock frequency, but a parallel application also benefits from adding more cores. The work in Paper VI presents an approach to find the optimal number of cores and the optimal clock frequency for a set of applications with different levels of parallelism. The controller includes a non-linear optimization solver which firstly uses application performance as input parameter to determine the QoS. Secondly, the controller uses application defined parallelism in order to determine how many of the available cores are useful for providing application speed-up if needed.

Since measuring application parallelism is usually not a trivial task, dataflow tools are exploited for automatically extracting the level of parallelism. The dataflow tool PREESM was used to generate a signal processing filter with different levels of parallelism per program phase; the filter contained a completely sequential phase and a parallel phase for filtering video frames. PREESM was then used to extract and inject the value of the parallelism during runtime to the QoS manager.

Author's contribution: The extension to allow multiple input, multiple output control was discussed between the author and the co-authors. Based on the recommendations from the co-authors, the author suggested to formulate the clock frequency scaling and sleep state balance as a non-linear optimization problem. The author integrated the solver and communications infrastructure of the QoS manager in the Linux environment. In order to determine the power dissipation of the chip based on the current clock frequency and sleep state setting, the author created an analytical power model of the CPU. The author also created a mathematical model describing performance as a function of parallelism in the applications. This model was then used to determine the level of speed-up when either increasing clock frequency or increasing the number of cores. The extraction and injection of the parallelism-value into the QoS manager was done by co-authors Erwan Nouges and Maxime Pelcat from INSA de Rennes. The co-authors used the already existing PREESM tool developed at INSA de Rennes to determine and extract parallelism, and to automatically generate c-code from the dataflow program. The author contributed with the write-up and presentation.

1.7 Paper VII: Accurate Energy Modeling for Many-Core Static Schedules

According to the current trend the number of processing elements on a chip increase, the manufacturing technology shrinks and the chip temperature increases. All these factors contribute to increased static power, which can only be reduced by shutting down parts of the chip dynamically. Sleep state based power management is currently available, but the latency of utilizing such a mechanism is orders of magnitude larger than clock frequency scaling. A significant latency when enabling/disabling cores on a multi-core system can result in energy waste rather than energy savings because of mispredictions in core wakeup/shutdown actions.

We investigate how the latency of sleep state based power management systems affect the decision making and the energy consumption. The shutdown and wakeup latency was measured with a set of parameters, under different conditions and on different platforms. With the obtained results we set up a static scheduler with an optimization solver to determine the best power management strategy based on the given applications and time resolution. The decisions are made based on a power model and the latency measurements. Results indicate at which time granularity the feasibility of using the current sleep state mechanism, and at which time granularity clock frequency scaling is the more viable option.

Author's contribution: The author implemented a benchmark framework for measuring the elapsed time of shutting down and waking up a core in the Linux environment. One kernel module and one user space application was implemented to trigger the sleep state mechanism, and the timing results from several samples was obtained. Co-authors Jörg Keller and Patrick Eitschberger from FernUniversität Hagen implemented the static schedule optimizer to model the preferred power saving technique based on the application and the timing granularity. By using a set of schedules, the author executed real-world experiments with the same parameters as were used in the schedule optimizer in order to verify the precision of the models used in the optimizer. The author contributed with the write-up and presentation.

1.8 Paper VIII: Performance Monitor Based Power Management for big.LITTLE Platforms

Recently new heterogeneous processors with one set of energy efficient cores and one set of high-end cores called big.LITTLE have appeared on the market. The purpose of such hardware is to allow applications with low performance demands to execute on the small energy efficient cores, while applications requiring high throughput can use the big high-end cores. With this approach, applications can be mapped on the most suitable core in order for the system to save energy and still guarantee sufficient performance.

While the hardware offers means of energy savings, software is currently not able to utilize the big.LITTLE architecture efficiently. Current power managers are purely workload-based, which means that an increase in clock frequency is issued as soon as the workload level reaches a given threshold. Furthermore, as the clock frequency is increased beyond a limit, the LITTLE cores are automatically switched to the big cores. Software is currently programmed to execute as fast as possible to allow high throughput. In combination with current power managers, this results in an execution strategy called “Race-to-Idle”. As applications are executed fastly in order to reach idle state, the big cores are mainly used even if the LITTLE cores would be sufficient to reach the desired throughput.

In Paper VIII we detail the drawbacks of the Race-to-Idle strategy in big.LITTLE systems. A general case video decoder is used to demonstrate the increased energy efficiency of regulating the clock frequency according to application performance rather than workload. The QoS manager presented in Paper V and VI is used to regulate the decoder framerate according to the user QoS requirements and the energy savings is evaluated.

Author’s contribution: The paper was written in order to stress the importance of proper resource allocation especially in heterogeneous big.LITTLE systems. The author adapted the controller implementation presented in previous papers for the big.LITTLE architecture. A new power model was derived by the author and an interface for application priorities was also adopted by the author. The author contributed with the write-up and presentation.

1.9 Paper cohesion

The origin of the research problem is presented in Paper I and a simple example on how to reach power proportionality is stated. Paper II further evaluates the control methods proposed in Paper I and extends the focus on the controller. From this point, the evaluated results are used as input to the more generic QoS manager presented in Paper V. This paper is focus on the more theoretical aspects of application-to-resource interfaces and is designed as an input for the later implementations. As the work is extended to multiple input multiple output controllers for increasing power proportionality on OS level, technologies such as task migration (Paper III) and workload consolidation (Paper IV) are studied. The multiple input multiple output controller (Paper VI) is finally integrated into the QoS framework discussed in Paper V, and realized as a real implementation. With the QoS manager implemented and integrated with the multiple input multiple output controller, Papers VII and VIII present specialized cases which explore selected important details in more depth. Figure 1.1 illustrates the cohesion of the papers and how the papers are categorized.

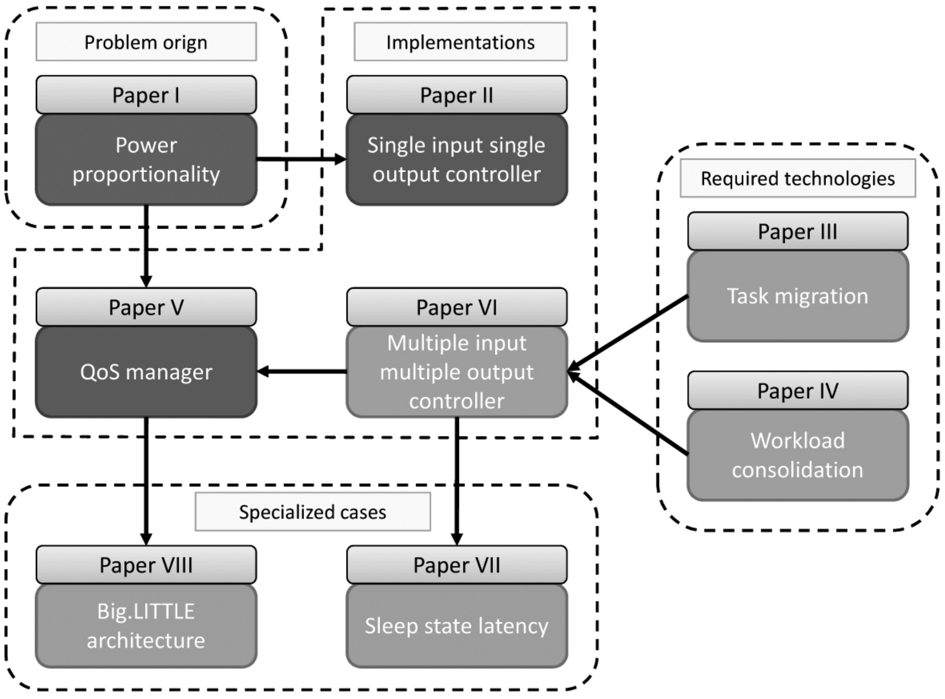


Figure 1.1: Illustration of paper cohesion. Yellow labels state the paper category. Red boxes indicate problem orig. Purple boxes indicate pure implementations. Green boxes indicate implementations as part of larger implementation. Orange boxes indicate specialized implementation or evaluations. Arrows indicate work input.

Chapter 2

Introduction

"No one should be asked to handle this trip."
— Raoul Duke, Fear and Loathing in Las Vegas - 1998

Hardware evolution has traditionally been the ultimate solution for computing systems to follow the trend of Moore's law, and allow continuous performance increase. The clock frequency scaling wall [14], which prevented computer systems from reaching a stable clock frequency above 3-4 GHz, was avoided by constructing multi-core systems. With multiple processing units, the performance of the systems were increased by introducing parallel programs to increase the computational throughput. Later, the memory wall [54] which represented a bottleneck in the memory bus as more cores emerged. The memory wall was, however, broken by introducing multiple memory buses (NUMA) [48] and network-based interconnects (NoC) [65] between the cores and the main memory. These types of connections expand by construct as more cores are added to a system, and the single-bus problem can be avoided.

The current road bump – CPU the power wall [14, 75, 87, 89] – is a new challenge currently being addressed. The CPU power wall allows no more dissipation of power with a fixed chip area due to limitations in the semiconductor material. As more transistors are squeezed into a smaller area of silicon, the power density increases [14, 63], and ultimately only a fraction of the full chip can be used at one time. This phenomenon is referred to as "*dark silicon*" [20, 85]. Traditionally, the power density problem was solved by creating more energy efficient transistors with, for example, lower voltage levels. This led to a scaling in transistor efficiency in proportion to the transistor density called "*Dennard scaling*" [16]. With current manufacturing techniques, Dennard scaling is no longer applicable [89] since the efficiency of the transistors are no longer in line with the transistor density. This leads to increased power density, which causes chips to malfunction due to the extreme temperatures caused by the large amount of power dissipated from a small area. *Hence, the performance of a computer system can no longer increase unless the energy issue is solved.*

While performance alone is a sound motivation breaking the CPU power wall, energy efficiency is an increasingly popular topic due to other physical limitations and cost issues. The very usability of battery operated mobile devices is completely dependent on its energy efficiency. With a limited ability to reach this goal, the users have been forced to adapt a daily re-charge routine for battery operated devices. Energy efficiency plays the same important role in high-end consumer desktop systems, but instead of the battery re-charge problem, the issue manifests itself notably in the electrical bill, excessive heat generation and unpleasant noise levels of the devices. For supercomputing centers to reach Exascale performance, the current energy efficiency of the computing systems requires a power feed of roughly 20 MW which is not realistic for a single facility [80].

Finally, the rate of expansion in computer systems is not in line with sustainable development and the ecological impact. Large scale datacenters consumed in the U.S. in 2013 100 billion kilowatt-hours of electricity, which corresponds to over 2% of the total electricity consumed in the U.S. With the current increase in energy demand, datacenters are predicted to release more carbon and consume more energy than the global aviation industry by the year 2020[1]. The Federal Energy Management Program have started an initiative to reduce the energy consumption in datacenters by increasing the energy efficiency of the datacenters facilities. According to the 2007 EPA Report to the U.S Congress, the energy consumption in datacenters could be reduced by 20%-40% by means of improving the datacenter infrastructure (not including the servers). According to the U.S. Department of Energy, an initiate was executed in 2014 to improve datacenter buildings, and is expected to reduce the energy consumption by 20% in the next ten years[2].

However, while the initiate only focuses on the building infrastructure, the largest energy consumer in the datacenters are the servers. Figure 2.1 shows the distribution of the energy consumption in three modern datacenters [52]. As the facility infrastructure including the cooling represents a significant part, the IT load – the servers – consume over 50% of the energy in each case. With the emerging CPU power wall, the end of Dennard scaling and increased focus on the man-made ecological impact, hardware solutions become inadequate for solving the global energy problem in computer systems.

[1]Statistics according to the U.S. Office of Energy Efficiency & Renewable Energy: http://energy.gov/eere/femp/resources-data-center-energy-efficiency

[2]https://www4.eere.energy.gov/challenge/sites/default/files/uploaded-files/Better_Buildings_Data_Center_Overview-FAQ.pdf

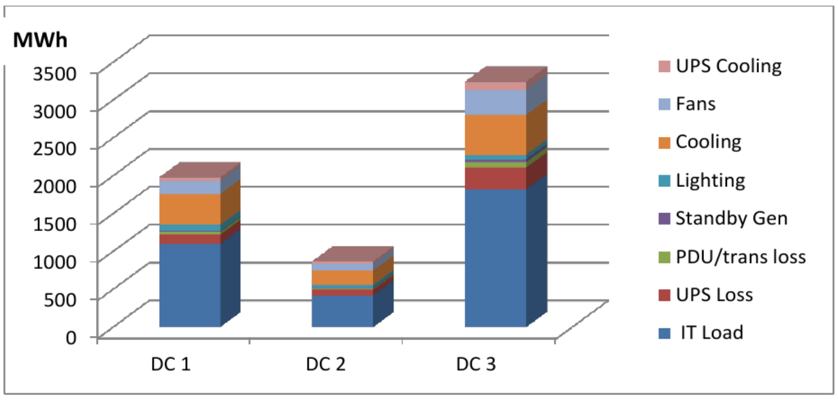


Figure 2.1: The distribution of energy consumption in three large scale datacenters (2014) [52].

2.1 Software Coordination

The work in this thesis, focuses on improving the energy efficiency in computer systems by increasing software involvement. While hardware solutions are very fast and have a low latency, software can comprehend a much larger intelligence base used for achieving a more efficient hardware utilization. The suggested hardware-software co-design allows the software to guide the hardware to maximize the energy efficiency.

The key goal for software applications is to not allocate more hardware resources than what is demanded by user satisfaction. By minimizing unnecessary resource allocation, user satisfaction is maintained while minimum energy is wasted. This is a demand pointing to a multi-objective solution, and possible a subjective interpretation with the following scenarios:

1. **Power-constrained.** The CPU is executing workload with a power cap for limiting the power envelope. Energy consumption is usually minimized with this strategy while the user accepts performance degradation. This solution is usually implemented as the ”*powersave*“ function in laptops and includes use cases such as web browsing without performance constraints.

2. **Time-constrained.** The CPU is executing workload with a given deadline to obtain at least a required lower-bound performance. The hardware has no restrictions. Use cases such as video decoding requires a steady performance in terms of framerate in order to satisfy the user.

3. **Operation-constrained.** The CPU is executing a given amount of bulk work without time limit and without limitations on the hardware.

Video encoding or source code compilation are considered as use cases in which no specific deadline is specified. A very long execution time will, however, waste energy simply by executing for a long time even though the power dissipation is low on average.

This thesis provides an investigation into how software-hardware coordination is used to create energy efficient computing systems. The presented work provides an interface for user defined software for more close communication with the hardware. With the provided facilities, software can be made *energy aware*, i.e. the software is aware of the required resources for proper execution. The following keypoints summarize the content of this thesis:

- An investigation into the power dissipation of modern multi-core processors.
- Recommendations to increase energy efficiency for program execution.
- An interface between the applications and the hardware in form of a runtime system.

2.2 Research Area

Energy efficiency in computer systems is a wide spread area crossing many domains such as mobile, desktop and server systems using different levels of hardware complexity. On the fundamental level, research in transistor technology[10, 68] has provided the stepping stones for building low power processors to fit in hand held devices. Clock frequency scaling [45, 35] is a method dating back to the 90's, allowing the early Pentium processors to reduce the power dissipation by scaling down the clock frequency when the performance is not needed. In more recent platforms, the clock frequency scaling has been complemented with CPU sleep states [37, 38] to further lower the power dissipation. As for large scale cloud solutions, energy efficient solutions have allowed to manage the workload with an optimal amount of server nodes [5, 51], which allows to physically shut down unused machines to save energy.

This thesis primarily focus on the runtime system within a computing node i.e. using the operating system's view of resources and its power management capabilities. Compared to other runtime systems [6, 8, 70], the focus is on modelling the physical system with a realistic, yet simple model to allow both portability and performance. The usability and a fast learning curve of the runtime system has also been a primary target, but rather than complete transparency [78] to the programmer a slight trade off between programmer effort and the effectiveness of the runtime system has been made. This thesis gives insight into how to reason about energy awareness in software and how to interface this awareness to the runtime system. It contains an approach for system modelling and optimization of resource allocation in real-world many-core devices. A practical case study of a complete ecosystem containing applications and a runtime system is finally presented, and experiments on modern many-core Linux platforms demonstrates its effect on energy efficiency.

2.3 Contribution of the Thesis

This thesis is divided into nine chapters which cover the technical background in the area of many-core energy consumption, recommendations for energy aware software and the implementation of the power manager.
Chapter 3 introduces the concept of energy efficiency in modern many-core systems, starting from the basic electrical characteristics in semiconductors towards the currently available power saving features in hardware. The content in this chapter stresses the importance of power proportionality – in other words all power dissipated in a computer system should be used for useful computations. The relation between power and energy is addressed to show how anomalies in power dissipation can be beneficial for energy consumption and vice versa.
Chapter 4 addresses the first dimension of energy efficiency in this thesis: *the energy-aware mapping.* The chapter describes the relation of power dissipation to the location of workload execution on a multi-core chip. The practical aspects of using power management in conjunction with different mapping policies are presented, and practical experiments demonstrate advantages and disadvantages for different policies.
Chapter 5 introduces the second dimension of energy efficiency in this thesis: *the energy-aware execution.* While the mapping question is fixed to the spatial dimension, workload execution concerns the timing. The trade-off between power and time is discussed in this chapter, which are the two contributing factors to energy consumption. Furthermore, the currently used method for determining performance based on workload levels is criticized and a new application-specific performance metric is suggested.
Chapter 6 ties science and engineering together with the design of a new type of power manager based on QoS metrics. The power manager is capable of monitoring the application-specific performance and allocates only the necessary amount of resources to an application mapped onto an optimal set of cores. Design choices for the power manager are motivated based on theoretical findings and practical issues in modern multi-core hardware.
Chapter 7 investigates more deeply into the heart of the power management construction. The admission control of the power manager is detailed in order to emphasize its feasibility in real-world systems. Design questions

such as algorithm complexity, scalability, accuracy and response time of different components are answered and further experiments are conducted to support practical aspects of the implementation.
Chapter 8 details the setup of describing the system as an optimization problem. Results from Chapter 4 and 5 are direct inputs for setting up the problem. The choice of optimization solver is motivated in detail with respect to the nature of the problem, complexity and response time. Moreover, simulations are used to predict the energy consumption for a set of applications in a system driven by an optimization based power manager. Finally, the latency of the power management mechanisms used in current hardware is investigated in order to clarify practical shortcomings in real-world systems.
Chapter 9 details the real-hardware implementation used in this thesis. A set of case studies include experiments to compare the default environment shipped in most Linux distribution with the new power manager on real hardware.
Chapter 10 finally concludes the thesis and initiates the discussion for including energy-awareness in future software frameworks.

Chapter 3

Energy Efficiency in Many-Core Systems

"*Congratulations, you've just left your family a second hand Subaru!*"
— Saul Goodman, Breaking Bad - 2008

The interplay between performance and energy consumption in many-core systems is a continuous mix of task scheduling, mapping and power management. It is important to understand the causes and the effects of the power dissipation in microprocessors in order to optimize the energy efficiency, since the power dissipation is an ever present factor as work is being processed and even as the chip is idle. The main components of the CPU power dissipation is discussed in this chapter in order to pinpoint the most significant factors for optimizing and obtaining an energy efficient execution.

3.1 Power Breakdown and Energy Consumption

Power is being dissipated by transistor switching as work is executed on the CPU and by leakage in the semiconductor material. The sources of power dissipation in a processing element can be categorized into two distinct parts: **dynamic** power P_d and **static** power P_s. The total power dissipation is the sum of both components as: $P_{tot} = P_d + P_s$. Both the dynamic and the static part can be individually broken down into several components defined in the following sections.

3.1.1 Dynamic power

Work execution on a microprocessor imposes chip activity in form of transistor switching. This activity is driven by the clock signal of the microprocessor, and is required to flip transistors on the chip to execute machine instructions.

The dynamic power is dissipated when the load capacitance C of the circuit gates and wires is charged and discharged to a voltage V. The contained charge is then $Q = C \cdot V$ (the unit is Coulombs). Since the charge-discharge cycles are executed at a clock frequency f, the charge moved per second is: $C \cdot V \cdot f$. The charge is delivered at voltage V; hence the power dissipated by the charge/discharge with a frequency f is $C \cdot V \cdot V \cdot f$. Most microprocessors are, however, very complex systems with a huge amount of transistors. Each cycle does not usually charge/discharge the complete capacitance of the CMOS chip but only a certain fraction of the capacitance. This fraction is usually referred to as the *activity* α of the chip.

The final relation of the dynamic power is shown in Equation 3.1.

$$P_d = \alpha \cdot C \cdot f \cdot V^2 \tag{3.1}$$

where the capacitance C is equal to the charge divided by the voltage $\frac{Q}{V}$. The electrical current is defined as the charge delivered per time: $I = \frac{dQ}{dt}$ and similarly $Q = \int_{t_1}^{t_2} I dt$ thus for one second $Q = I \cdot t$ (*ampere seconds*). The unit for dynamic power is hence derived as: $\frac{Q \cdot V^2}{V \cdot s} = \frac{A \cdot s \cdot V^2}{V \cdot s} = A \cdot V = W$ (Watts). In case the complete chip capacitance is charge/discharged during a cycle, the activity factor α is equal to 1.

As seen from the equation, the dynamic power is linearly dependent on the clock frequency but quadratically dependent on the core voltage. A low core voltage is hence preferred since the power dissipation is then significantly reduced.

3.1.2 Static power

Besides the dynamic power dissipation as a result of transistor activity in the chip, static power as a result of semiconductor imperfections is also dissipated. The static power is a combination of several factors resulting in leakage currents in the transistors, which is a current flow though the transistor even though it is closed.

One source of leakage current is the *subthreshold current* [44]. The subthreshold current is an undesired flow of current as the transistor gate voltage is below the threshold voltage V_t. The subthreshold current I_{sub} flows between the source and the drain because of imperfections in the semiconductor material in combination with the operating conditions such as temperature. The magnitude of I_{sub} depends on the manufacturing technology of the semiconductor, i.e. how precise the transistor is able to cut the current flow at the threshold voltage. Secondly, the source voltage range of the transistor influences the subthreshold leakage. A very narrow voltage range results in a more imprecise transistor and the definition between high- “on” and low- “off” is more difficult to determine. Because of this reason, microprocessors cannot currently operate on an arbitrarily low voltage. Equation 3.2 from [44] details the subthreshold leakage current.

$$I_{sub} = K_1 \cdot W \cdot e^{-V_{th}/n \cdot V_\theta}(1 - e^{-V/V_\theta}) \tag{3.2}$$

where K_1 and n are architecture specific constants, W is the transistor gate width and V_θ is the thermal voltage explained in Section 3.1.3.

An effect of shrinking the manufacturing technology to a very small scale is the increased *tunnelling current* [76]. The tunnelling effect is a result of the Uncertainty principle [26] and the wave-particle duality of matter. As the distance between two materials is decreased to a very short distance, the wave functions of the electrons partially overlap and the probability of an electron tunnelling across the insulator material increases. The other significant source of tunnelling current is the band-to-band tunnelling [2] effect. This is a non-quantum mechanical gate-induced drain leakage in which a high electric field between the transistor bands can cause the current flow from the gate to the drain. Both tunnelling effects increases the static power since they cause leakage currents in the transistors.

3.1.3 Thermal Influence

Chip temperature influences the static power by increasing the leakage currents in the transistors. By increasing the leakage current, the temperature further increases, which in turn increases the leakage current as a positive feedback loop. As seen in Equation 3.2, the subthreshold leakage current is exponentially dependent on a parameter V_θ, *the thermal voltage*. The thermal voltage represents the average energy of individual electrons. V_θ is proportional to $\frac{kT}{q}$ where k is Boltzmann’s constant, q is the electron charge and T is the temperature. This means that the thermal voltage is linearly dependent on the temperature, but the leakage current is exponentially dependent on the thermal voltage.

The leakage current effect was demonstrated by running an ARM CPU on constant clock frequency while not executing any workload. The CPU temperature was continuously increased with an external heating device while the power dissipation was measured. Figure 3.1 shows the power dissipation of the ARM chip and the power dissipation of the complete board as a function of temperature. As seen in the figure, the power dissipation increases exponentially (also confirmed in the work of Martinez et al. [55]) as the temperature increases and reaches a maximum at 80 °C. At this temperature the CPU is automatically stalled as a safety functionality to decrease the chip temperature.

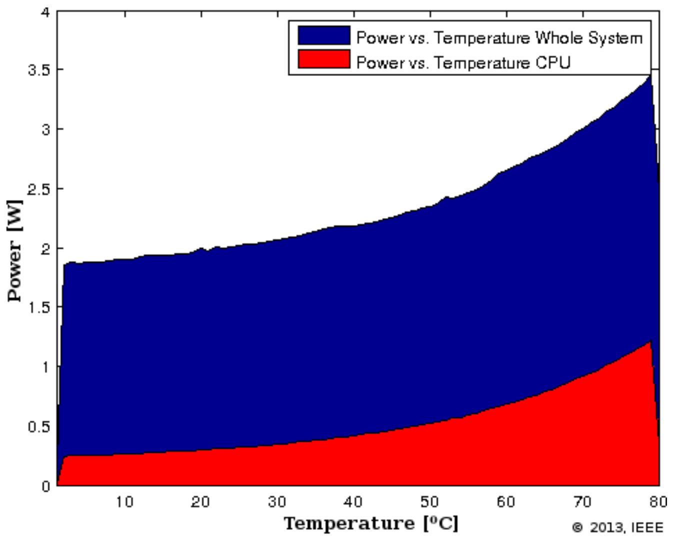


Figure 3.1: Influence of temperature on power dissipation for an ARM system.

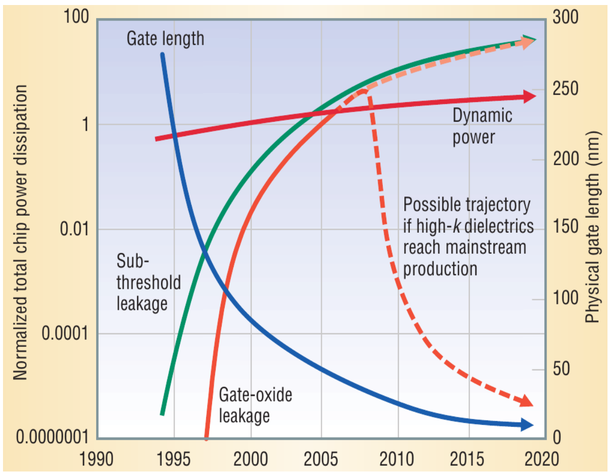


Figure 3.2: Total static- and dynamic power breakdown and the trends [44].

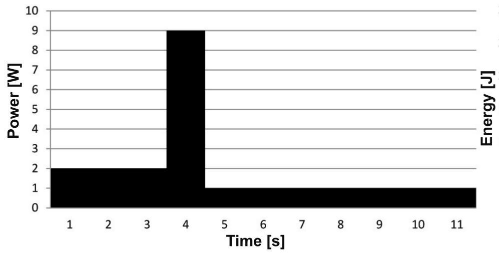
In previous generation microprocessors, the static power could be ignored since the dynamic power was dominating the total power dissipation i.e. $P_d \gg P_s$ [10]. However, with smaller transistors, lower voltage, higher clock frequencies leading to higher temperatures and more transistors on the chip, the static power is currently becoming more significant and is expected to dominate the total power dissipation in next generation microprocessors

[4, 44, 76]. Figure 3.2 shows the history of the power breakdown from typical microprocessors as well as the future predictions. As illustrated, in the early 90s up to the early 00s, the static power could be completely ignored since the dynamic power was orders of magnitude larger. This statement is, however, no longer true.

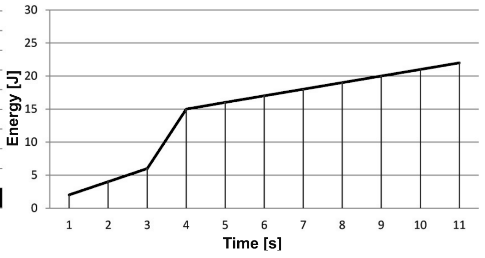
3.1.4 Energy consumption

The energy consumption of a microprocessor is the result of the power dissipated over a determined time window. Energy is measured in Joules (J) which is equal to Watts*Seconds (Ws). Since the power dissipation is a dynamic and instantaneous value, the energy consumption can be measured as the average power multiplied by the execution time as $E = P_{avg} \cdot T$. Equivalently, the energy is defined as the power integral over a defined time: $E = \int_{t_1}^{t_2} P(t) \cdot dt$

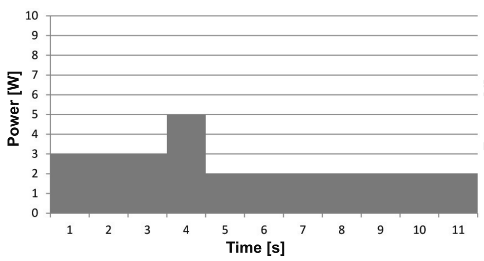
With the added time dimension, the minimization of energy consumption becomes a two-variable optimization problem. Figure 3.3a and 3.4a illustrates a scenario in which an 11-second execution includes one high power peak at $t = 4$, but with a low overall power dissipation. Similarly, Figure 3.3b illustrates the same length execution with a lower power peak at $t = 4$ but with an overall higher power dissipation. The total energy consumption is 22% higher in Figure 3.3b even though the instantaneous power dissipation at $time = 4$ is lower. The question stands *whether to execute the work fast in order to decrease time or slow in order to decrease the power dissipation.* This is further discussed in Chapter 4.



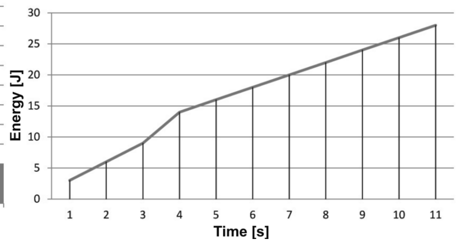
(a) System 1: Power dissipation.



(b) System 1: Energy consumption.



(a) System 2: Power dissipation.



(b) System 2: Energy consumption.

3.2 Power Proportionality

To achieve the most energy efficient execution, all dissipated power should come as a direct result of processing work and nothing should be dissipated as waste. In such a theoretical system, every transistor switch and every clock signal generated should be directly translated into useful work. An optimal system is assumed to have a power dissipation completely proportional to the amount of executed work – the *power proportionality* of such a system is therefore equal to 1.

Cloud server systems have adapted the notion of proportionality in order to build flexible platforms both for the cloud users and the cloud owners. The user can purchase a selected amount of processing power proportional to a rental fee, and the processing can spread across several physical machines transparent to the user. The cloud owner can, on the other hand, control the server machines according to the current rental status of the user to disable all unused machines in order to save power.

Since the perfect proportionality is practically not possible, an as high proportionality factor as possible should be considered in order to maximize the energy efficiency. The extraction of the proportionality factor is, however, generally not straight forward, and depends on the intentions of the software. Consider a webserver handling requests on demand from external users. The amount of work applied on the webserver is highly dependent on user activity, the website content, time of day etc. The system must therefore be scalable to provide the necessary resources during high activity and to shut down parts of the system during low activity to save power.

A control mechanism based on a PID controller was presented in Paper I [29] and Paper II [30] which simulated the adaptation of resources to the current amount of work, in this case webserver *requests per second*. The control mechanism was tunable in order to adapt to different scenarios by using a set of parameters. Figure 3.5 shows a simulation with poorly tuned parameters for a real-world workload scenario. As seen in the figure, the resources in form of processing elements in the cloud system were not able to well adapt to the very peaky nature of real webserver traffic patterns. By tuning the control parameters, the improved result shown in Figure 3.6 demonstrates increased power proportionality. The improved controller is better able to adapt the resource use to the incoming workload.

For any type of power proportional computer system, the hardware resources must be on demand adaptable according to the executing software. Furthermore, a software part controlling the hardware is required to efficiently collect input from the software to increase the intelligence of the resource allocation. The following sections present the currently used hardware power management capabilities in modern multi-core systems, and how these mechanisms can be used to increase the power proportionality.

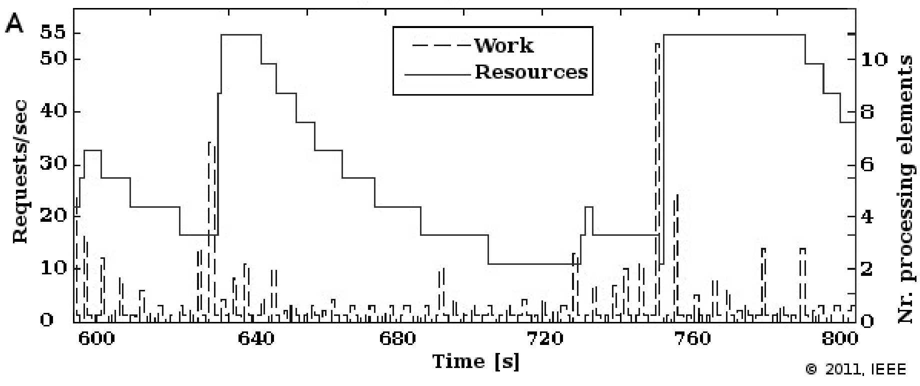


Figure 3.5: Low power proportionality due to poor parameter tuning (resource curve not able to adapt well to the work curve).

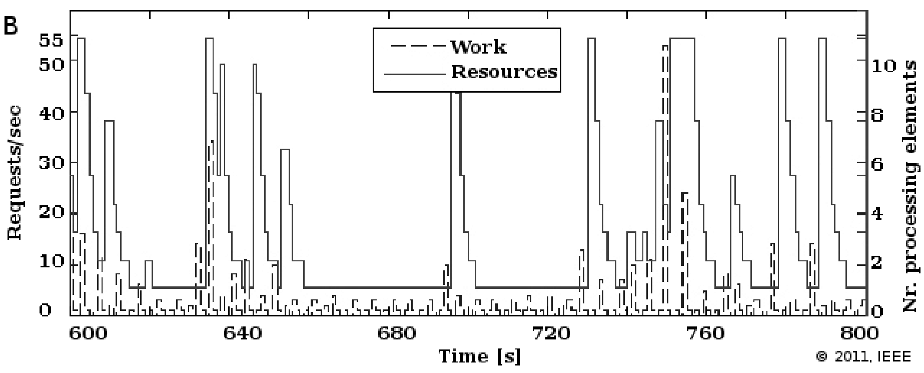


Figure 3.6: High power proportionality due to better parameter tuning (resource curve able to adapt to the work curve).

3.3 Hardware-Based Power Management

In order to maintain a power dissipation more proportional to the amount of work processed, the system requires hardware support to scale the hardware resources on demand at a low enough granularity. The two most common hardware mechanisms found in modern multi-core systems are Dynamic Voltage and Frequency Scaling (DVFS), and Dynamic Power Management (DPM). Both functionalities are described in this chapter as well as their impact on both dynamic and static power.

3.3.1 Dynamic Voltage and Frequency Scaling (DVFS)

The traditional power saving mechanism, DVFS, was already implemented in single core systems to increase power proportionality. The technique is used to dynamically scale the CPU clock frequency and the core voltage according to the workload in order to minimize power waste. Since the power dissipation of the CPU is linearly dependent on the clock frequency and quadratically dependent on the core voltage (Section 3.1.1), DVFS is a suitable technique to significantly reduce the dynamic power. Scaling down both voltage and frequency also reduces the temperature since the power output of the chip is reduced. This leads to reduced static power since the thermal voltage in the chip is reduced.

While the clock frequency can, in theory, scale down arbitrarily, the core voltage setting must respect the current clock frequency in use. A low voltage setting in combination with a high clock frequency can lead to CPU instability since the threshold voltage in is not reached fast enough and the distinction between high- "1" and low- "0" becomes more vague. In practice, CPU manufacturers provide a table indicating which voltage setting is appropriate for which clock frequency. The operating system utilizes the table to automatically switch between frequency and voltage settings completely transparent to the user. For ACPI compliant devices, a certain frequency/voltage combination is referred to as a *P-state* or Performance state. Table 3.1 shows the P-states for an Intel Core i7-3770 processor. In this example, the P-states range from 0 to 15, but the available settings usually varies between CPU types. The voltage setting for this particular processor is in the range [0.895 1.058] depending on the clock frequency setting.

Frequency	Voltage	P-state
3.4 GHz	1.080 - 1.058 V	P_0
3.3 GHz	1.060 V	P_1
3.2 GHz	1.040 - 1.458 V	P_2
3.0 GHz	1.005 - 1.010 V	P_3
2.9 GHz	0.990 - 0.995 V	P_4
2.8 GHz	0.975 - 0.980 V	P_5
2.7 GHz	0.965 V	P_6
2.6 GHz	0.955 V	P_7
2.4 GHz	0.935 V	P_8
2.3 GHz	0.925 V	P_9
2.2 GHz	0.920 V	P_{10}
2.1 GHz	0.915 V	P_{11}
2.0 GHz	0.905 - 0.910 V	P_{12}
1.8 GHz	0.900 - 0.905 V	P_{13}
1.7 GHz	0.900 - 0.905 V	P_{14}
1.6 GHz	0.895 - 0.900 V	P_{15}

Table 3.1: P-states for an Intel Core i7-3770 processor (Ivy Bridge).

To illustrate the effects of P-state selection, the quad-core i7 CPU was stressed to maximum capability while switching between the P-states. Figure 3.7 shows the power dissipation of the CPU for each P-state setting as an exponentially increasing power output with respect to the clock frequency setting and core voltage. This implication leads to a higher power difference when switching from a high P-state to a medium P-state, and a lower power difference when switching from a medium P-state to a low P-state. With this behavior, the efficiency of using DVFS decreases as lower P-states are used since the power savings become continuously smaller. Nevertheless, the profile of the power dissipation curve differs between CPU architectures and types, which means that the efficiency of DVFS also varies between different hardware devices.

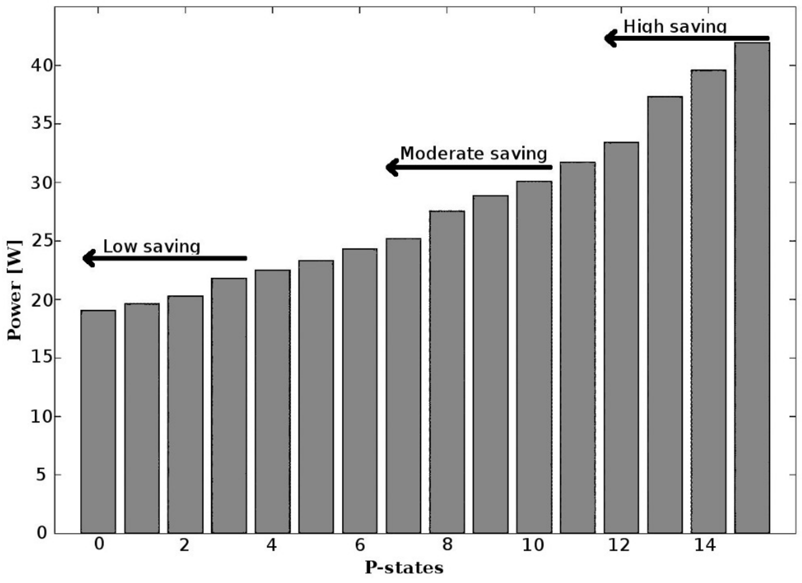


Figure 3.7: Total CPU power dissipation of an Intel core i7-3770 in different P-states during full load.

DVFS implementation The DVFS mechanism is implemented under Linux in a *frequency governor*. The governor monitors the CPU workload and, based on a set of policies and settings, adjusts the P-state of the CPU. Different governors can be implemented for different systems, different scenarios etc. and several governors can exist in the same system, but only one is active at a time. Typical governors are *Performance*, *Userspace* and *Ondemand*.

The policy of the Performance governor is to simply select the highest P-state of the CPU exclusively. The Userspace governor utilizes inputs from the user in form of clock frequency settings. No automatic clock frequency scaling is performed with Userspace without the user's explicit request. Ondemand [79] is the most typical governor used to dynamically and automatically select clock frequencies during runtime. Clock frequencies are selected with Ondemand as follows:

- CPU workload is monitored.
- In case the workload exceeds a threshold limit, the governor switches to the highest clock frequency.
- The governor decreases the clock frequency step-wise until the lowest feasible setting is found.

Figure 3.8 illustrates this behavior; as the workload increases enough, the Ondemand governor selects the highest clock frequency to minimize the response time in case of heavy workload. The governor then scales down the clock frequency to the lowest feasible frequency. In case the workload again exceeds the threshold, the clock frequency is increased to the maximum and the same steps are repeated.

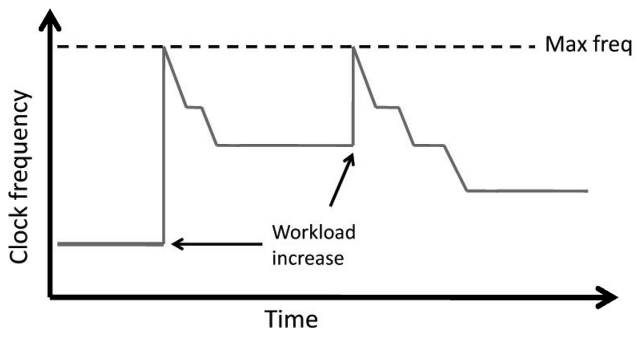


Figure 3.8: The Ondemand frequency governor scales up the frequency to the maximum value as the workload threshold is reached.

3.3.2 Dynamic Power Management (DPM)

Dynamic Power Management (DPM) manages the low power states, also called sleep states, of the CPU. Instead of reducing the dynamic power, DPM is used to minimize the static power by disabling parts of the chip, thus reducing the leakage currents in the transistors. As a part of the chip is shut down, the current feed to the transistors is cut and the leakage is reduced.

Similarly to the performance levels, the shutdown levels are referred to as sleep states or *C-states* according to the ACPI standard. A C-state is a standardized definition of the components inside the CPU being disabled, and the activity of the CPU and its context. The availability of C-states also depends on the CPU type, model etc., but the most commonly used C-states are listed below:

- C0: This is the highest C-state. The complete CPU is active and continuously executing instructions with no halts.
- C1: The C1 state is usually used as a CPU core does not receive work in the work queue. The core is then halted by executing the halt instruction (HLT) after which the clock signal to the core is gated. The bus interface unit and advanced programmable interrupt controller remain however clocked. The core is then regularly woken-up to check whether new work has arrived. The C1 state must be supported by all ACPI compliant hardware.
- C1E (Enhanced C1): The C1E state has similar capabilities as the C1 state, but is capable of lowering the supply voltage as well.

- C3: The C3 state improves the power savings further by completely stopping the internal clock signals. L1 and L2 caches are flushed into the L3 cache and snoops are ignored.
- C6: In C6 all core clocks are stopped. The core state is saved in a static RAM outside of the CPU. The voltage to the core can be completely cut off and the core is fully shut down.

To demonstrate the impact on the power savings, the power of a quad-core Intel i7 CPU was measured whilst in idle state without workload. The CPU was executing on a range of different clock frequencies and the maximum allowed sleep state was step-wise increased from C0 to C6. Figure 3.9 shows the power dissipation of the CPU.

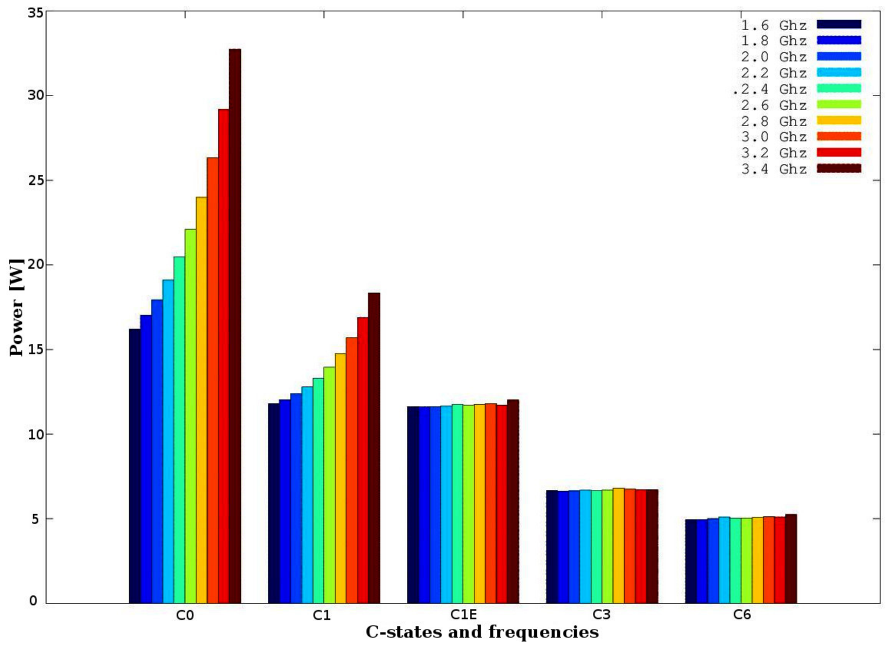


Figure 3.9: Total CPU power dissipation of an Intel core i7-3770 in different C-states during idling.

In C0 the CPU is actively idling which means that instructions are continuously executed, mostly in the OS idle loop. As seen in the figure, the clock frequency has a significant impact on the power, since increasing the clock frequency leads to an increased number of instructions executed. When allowing C1, the CPU is only executing instructions when checking the work queue for arriving workload. This leads to lower power, but also a lower influence of clock frequency scaling since the CPU is halted for a significant amount of time. C3 and C6 further decreases the power dissipation since more parts of the chip is shut down, and the static power is further reduced.

DPM implementation DPM is implemented in Linux using the CPU hotplug functionality, which disables a CPU core on request by either the user or the kernel. The hotplug functionality was initially intended as a facility for replacing CPU cores without shutting down the system. As the importance of reducing static power increased, CPU hotplug was instantiated as a power saving feature.

A CPU core is shut down upon request and hidden from the OS scheduler after which the core is shut down. For ACPI compliant devices, the C-state used by the hotplug varies between CPU types and settings. The internal clock is, however, usually halted and a wake-up can only be performed by a physical inter-core interrupt signal from another core. Utilizing hotplug in Linux requires a sequence of user space, kernel space and architecture specific instructions, which is explained below:

1. A shutdown command is issued by the user (or the kernel)
2. The selected core is locked in order to not accept incoming jobs
3. A message "CPU_DOWN_PREPARE" is sent to the kernel
4. A callback function in the kernel receives the message and migrates all jobs to another core in case the core is not idle
5. A message "CPU_DEAD" is sent to the kernel
6. A callback function in the kernel receives the message and flushes the cache, interrupts are disabled and the cache coherency is switched off
7. Architecture specific assembly routines are called to physically shut down the core

A wake-up is performed similarly but in the opposite order and the wake-up is triggered by an inter-core interrupt. The wake-up also includes a set of callbacks to restore the core and to initialize the idle thread on the core.

Since the CPU hotplug is driven by kernel messages and callbacks, the latency of accessing the functionality can influence the efficiency of the hotplug functionality as a power saving feature. This issue is further investigated in Section 8.4. Further details regarding the hotplug mechanism is found in the article from Mwaikambo et. al [56].

3.4 Summary

Power is continuously dissipated in a microprocessor as a result of active calculations and by leakage in the semiconductor material. While power dissipation cannot be completely eliminated, the power proportionality factor of a system can determine how much of the dissipated power is used for actual work, and how much is wasted. To increase the power proportionality, modern CPUs include hardware mechanisms for regulating the power according to the amount of work to execute. Hardware alone contains, however, only a limited amount of intelligence, and software implementations are required to increase the efficiency of the mechanisms. The following chapter details the crucial factor: *workload mapping*, for achieving effective power regulation.

Chapter 4

Workload Mapping

"*Then start asking the right questions!*"
— Rust Cohle, True Detective - 2014

Modern processors use multiple processing units, called multi-cores, to distribute the workload. As the workload is processed in parallel, an increased number of operations can be executed in a smaller time window compared to a single-core system. This composition can be used to either increase performance, or to lower the clock frequency while maintaining constant performance compared to a single-core system. By lowering the clock frequency, the dynamic power dissipation is reduced as explained in Chapter 3, and the energy consumption can be reduced for constant time executions. Still, the static power is continuously increased as more cores are added to the system since more transistors leaking current are added. The mapping of the workload is an issue of handing the trade-off between static and dynamic power dissipation by either spreading the workload to many cores with low clock frequency or consolidating the workload to few cores with high clock frequency.

This chapter presents a detailed analysis of the task mapping and the dynamic re-mapping called task migration. Two different mapping strategies are discussed and real-world experiments with both strategies are presented.

4.1 Relation to Energy Efficiency

As recalled from Section 3.1.4, the energy consumption of a computer system is calculated as the power dissipated over a time window. Therefore, the minimal energy consumption is obtained when the time integral over the total power dissipation is minimized. As the total power dissipation is a combination of static and dynamic power, the minimal energy consumption is obtained for a fixed power case as: $E_{min} = Minimize : (P_s + P_d) \cdot time.$

The relation between workload mapping and energy efficiency is manifested in the balance between static and dynamic power. The question becomes whether *it is worth decreasing the dynamic power using clock frequency scaling at the cost of increasing the static power by adding cores.* For the sake of comparison, two completely opposite mapping strategies are considered:

1. **Balanced mapping**: The workload is always distributed as evenly as possible over all processing elements.
2. **Consolidated mapping**: The workload is always packed onto as few processing elements as possible.

Figure 4.1 illustrates both mappings and the power dissipation. The left part illustrates the balanced mapping in which four cores are executing the workload completely in parallel at 400 MHz. The static and dynamic power, in this case, is equal for all cores since all cores execute the same amount of workload. In the right case in Figure 4.1, the workload is mapped to only one core. In order to maintain the same performance as the balanced case, the active core must execute at four times higher clock frequency. The static power for the first core is thus increasing since the temperature of the total system is higher than in the balanced case, and the dynamic power is higher since the clock frequency is increased.

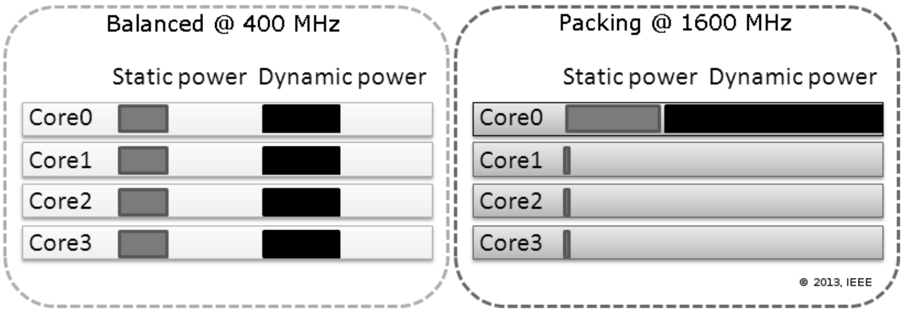


Figure 4.1: A balanced execution (left) is performed on all cores on a low clock frequency. A consolidated execution (right) is kept to as few cores as possible on a high clock frequency. The relative power is illustrated.

Naturally, the scenario illustrated in Figure 4.1 is an ideal case in which the workload is perfectly parallel, no OS interrupts are present, the workload is not re-mapped once executing and no mapping overhead is present. In order to gain a better understanding of the optimization involved in workload mapping, the mapping process was investigated in more detail. Practical aspects of workload mapping and task migration is presented in the following sections, together with the implementation of a new type of Linux scheduler used for evaluations.

4.2 Task Migration

Task migration refers to as the re-mapping of workload between different processing elements, or cores. The workload is usually quantified into work items called *tasks*, *threads* or *processes* depending on the currently used runtime system. In this thesis, the work is referred to as a task since this concept can be applied on both real-time systems, accelerator-based systems and desktop systems.

A task is a defined work quanta which can be executed in parallel with other tasks on a multi-core system in case no inter-task dependencies exist. A task can establish communication with other tasks, and data from one task is allowed to be used by another task. In this case, the task using the input is dependent on the first task hence defining an implicit order in which the tasks must be executed.

A task can be mapped for execution on a core, and re-mapped during runtime to another core based on decisions from a scheduler or other entity handling the task migration and mapping. The source core and destination core for the task migration depends on several factors such as core utilization, task type, task communication and the underlying platform. The goal of the scheduler issuing the task migration is to map the task execution on a combination of cores which follows a defined policy; such as maximum throughput, minimum energy consumption, minimum heat generation etc.

4.2.1 Linux run-queue selection

Linux schedulers use run-queue (RQ) selection to distribute tasks over the available cores. Each core uses a dedicated RQ which contains the collection of tasks currently runnable on the core. Placing tasks on a RQ is equivalent to scheduling a task on the RQ's corresponding core. Because of a shared memory and a symmetric OS view used in Linux, the scheduler can map any task to any core while maintaining a global task state view.

The RQ selection functionality places newly *created* tasks, as well as newly *awakened* tasks, on a suitable RQ based on the given policy. As a new task is created, the scheduler selects a RQ onto which the task is placed. The task is then scheduled on the core according to the scheduling policy. Tasks not currently runnable i.e. *waiting* or *blocked* tasks are not placed on a RQ but are placed on a special wait queue [50]. Since waiting tasks are not assigned to any specific RQ during the wait state, the scheduler places the task on a RQ upon wakeup. Because the initial RQ placement will affect the performance of the task in form of cache usage, this data is taken into account when scheduling a newly awakened task. The placement of an awakened task is hence dependent on the core's *idle percentage*, which represents the time elapsed in the idle state compared to the execute state.

The default Linux Completely Fair Scheduler (CFS) [41] places an awakened task on the RQ it was previously executing on in case the idle percentage of the RQ is above the safety margin. In case the target RQ is overloaded, the scheduler places the task on another RQ with an idle percentage above this safety margin. For a fully utilized system the scheduler places the task on the initial target RQ i.e. the task's previous RQ.

4.2.2 Platform dependence

Modern multi-core systems have obtained a diverse outlook in form of hardware configurations. As more cores are added, the platforms are referred to as many-core systems in which the core composition usually varies from the traditional multi-core systems. Cores are usually interconnected with a network like connection instead of the traditional bus because of the ever present memory bottleneck or Von Neumann bottleneck [7] present in such systems. Recent many-core platforms such as the Tilera [88], Xeon-Phi [62], Single Chip Cloud [36], MPPA [15] and the Adapteva [83] all use a Network-on-Chip (NoC) connection for inter-core communication.

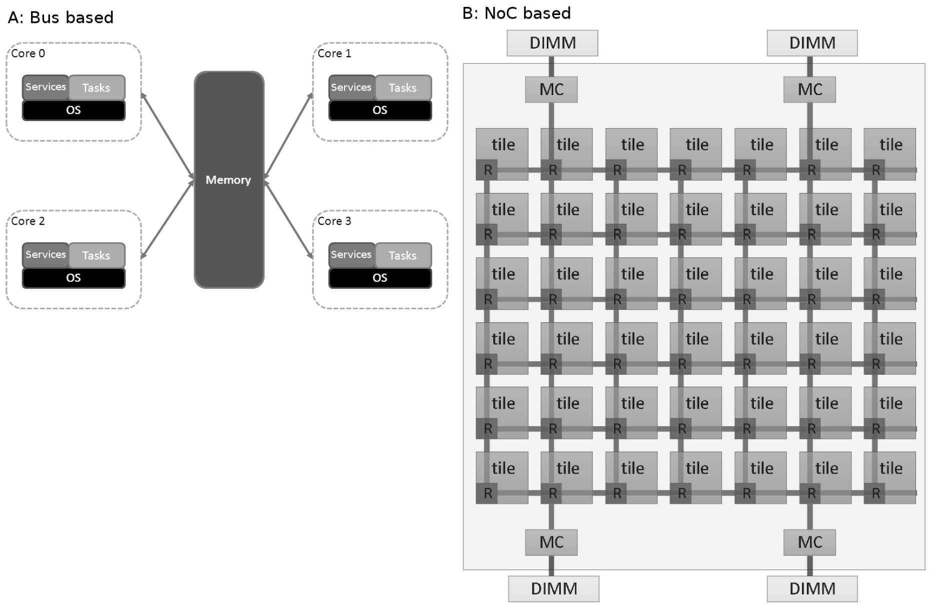


Figure 4.2: Structure of a bus-based system and a NoC-based system.

The hardware structure, number of cores and communication structure influences the behavior of utilizing task migration, since migrating a task requires data transfer between the processing elements. Figure 4.2 shows two different multi-core systems: (A) A bus based systems in which the inter-core communication is handled by a single bus, (B) A Network-on-Chip based system in which the inter-core communication handled by an on-chip network. All cores connected to the network can access the shared memory (DIMM) split into several locations and handled by message controllers (MC).

While the NoC type interconnect increases the bandwidth and the scalability compared to the bus based system, the communication time becomes more unpredictable since the communication packages are transmitted on the on-chip network according to a routing algorithm. In case the network is congested, the sending a data package might involve using a longer route than in a non-congested network.

A task migration mechanism was simulated using a many-core NoC simulator to pinpoint the parameters influencing the overhead. A model of the Intel SCC [36] was used as reference platform. The processor is clocked to 533 MHz and the network clock frequency is set to 800 MHz. A local L1 cache is used for all core while a L2 cache is shared between all cores. The task migration was simulated to **(1)** detach a task from the source core, **(2)** transmit the task data over the on-chip network, **(3)** attach the task on the target core. Since a shared memory system was considered, the complete task context was not transmitted but only the reference pointers to the task. This means that the amount of data to transmit is very low.

The task migration was simulated as follows:

1. L1 data cache of the source processor is cleaned.
2. The data is sent over the network.
3. The target core receives the code and data, which is loaded into the private L1 cache.

In the first experiment different cache miss-rates were used to model the task migration overhead. Figure 4.3 shows the results from using a miss-rate of 0 to 100% for the L1 cache and various parameters for the L2 cache.

As seen in the figure, when keeping the L1 miss-rate under 50% a relatively constant migration time can be expected. When the miss-rate exceeds this number, the L2 miss-rate will start to influence the results depending on the L2 miss-rate. By disabling the L2 cache, the migration time can increase with up to 10x the initial time, but with a L2 miss-rate of 50% the time is expected to not be influenced.

Secondly, the task migration distance to the memory controller was evaluated. A physically longer distance between the memory controller and the core could influence the task migration time since the packet uses a longer route. According to the results in Figure 4.4, the task migration overhead was, however, not influenced by the distance significantly. This is because the small amount of data involved in migrating a task uses a very small fraction of the network capacity, and the slow external memory almost completely dominates the overhead. In summary, the task migration overhead is mostly dominated by the cache miss rates and not the distance to the memory controllers.

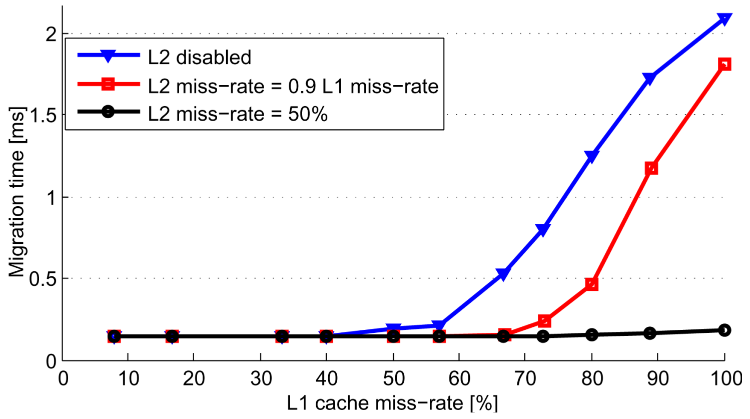


Figure 4.3: Task migration overhead using different cache miss-rates.

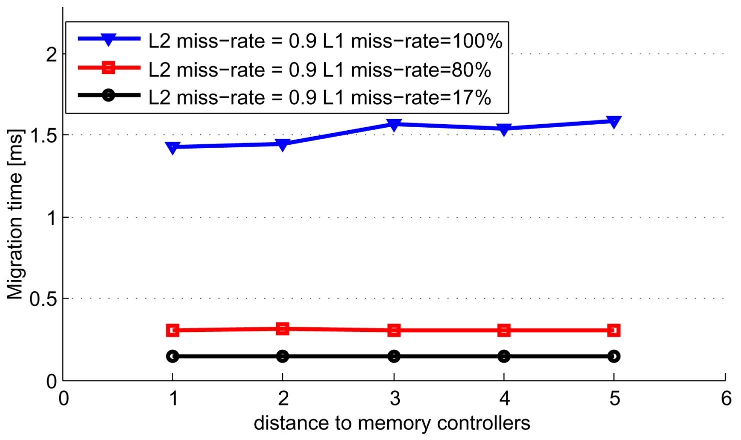


Figure 4.4: Task migration overhead using different migration distances in a NoC system.

4.3 Workload Consolidation

Since the task mapping directly influences the resource allocation and thus the power dissipation of the CPU, an approach to modify the task mapping strategy was investigated. The default mapping strategy of the CFS scheduler [41] is to balance the workload as evenly as possible over all cores. This allows a minimal clock frequency since the workload is processed mostly in parallel. The CFS strategy is therefore aimed to minimize the dynamic power dissipation while allowing a static power dissipation from all cores.

Figure 4.5 illustrates the mapping strategy of CFS using four tasks, each utilizing 20% of the CPU core.

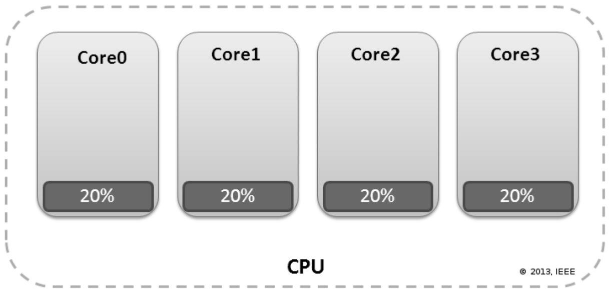


Figure 4.5: Task mapping of the Completely Fair Scheduler (CFS): the workload is evenly placed [41].

The complete opposite strategy was therefore investigated – this strategy involves mapping as many tasks to as few cores as possible while not overloading the core. With this approach the idle cores can be shut down and the static power is completely eliminated from the idle cores. Task packing or *load consolidation* was suggested in the Overlord Guided Scheduler (OGS) [77] and implemented by authors Fredric Hällis and Robert Slotte [1]. Figure 4.6 illustrates the load consolidation by mapping all tasks to a single core. Since all tasks utilize the core by 20% the total core utilization is 80% which means that the core is not overloaded. In case several cores are actively

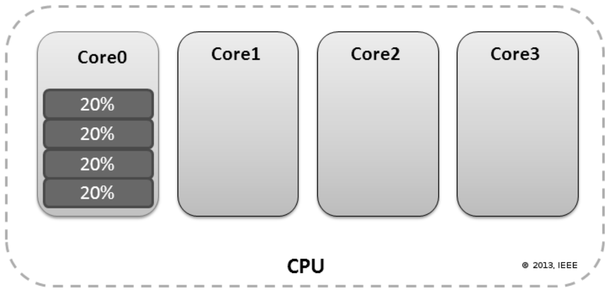


Figure 4.6: Task mapping of the Overlord Guided Scheduler (OGS): the workload is consolidated to few cores [77].

processing load, the tasks are mapped to the most loaded, non-overloaded core as illustrated in Figure 4.7. After the task is removed from Core2, the core can be shut down and the static power is decreased.

[1] Available at https://github.com/rslotte

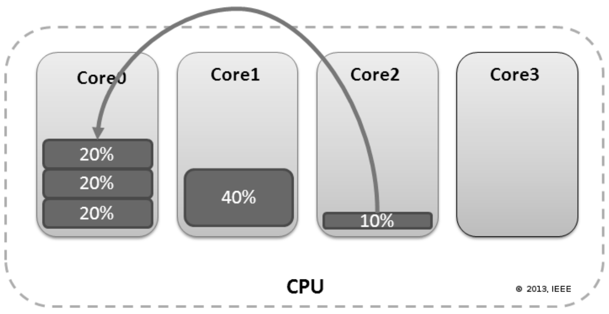


Figure 4.7: The OGS migrates the workload to the highest loaded- non-overloaded core.

4.3.1 Experimental results

The energy consumption of using the OGS was evaluated and compared to the CFS. A set of benchmarks were executed while the total energy consumption was measured with an external measurement device.

Spurg-Bench[2] was used as basis for evaluation. Spurg-Bench is an open-source benchmark for measuring load based performance on multi-core systems. The benchmark generates a selected number of threads; each of them able to apply a selected load level on a CPU core. The load level assigned to a thread represents a percentage of the full capacity of the CPU, e.g. four threads each with the load 50% can either run on two cores at 100% load or on four cores at 50% load each.

The main operations of a Spurg-Bench thread are: execute, and sleep, and the time ratio between execution and sleep is monitored by the Linux kernel and appears as the normal load level of the thread. As workload, the user can select a certain number of operations to execute, which are evenly divided among the created threads. For our experiments, we chose 100k floating points as the operations executed by the Spurg-Bench threads and load levels in range [10 90]% (for each thread). Spurg-Bench finally measures the execution time of completing all of the operations. The experiments were conducted on an Exynos 4412 SoC based on the quad-core Cortex-A9 CPU. Both the CFS and the OGS were integrated into Linux version 2.6.35.

Since the static/dynamic power ratio varies with the temperature according to Equation 3.2, the test bench was executed in various ambient temperatures: **(1)** Without active cooling, **(2)** With active cooling, **(3)** In a freezer (-20 °C). Figure 4.8 shows the results from the test. Each pair of curves shows the energy consumption for a constant-operation experiment. For each curve-pair, a crossover point is found at which the CFS strategy becomes more energy efficient than the OGS strategy. Depending on the ambient temperature, the point appears at different load levels.

[2]Available at https://github.com/ESLab/spurg-bench

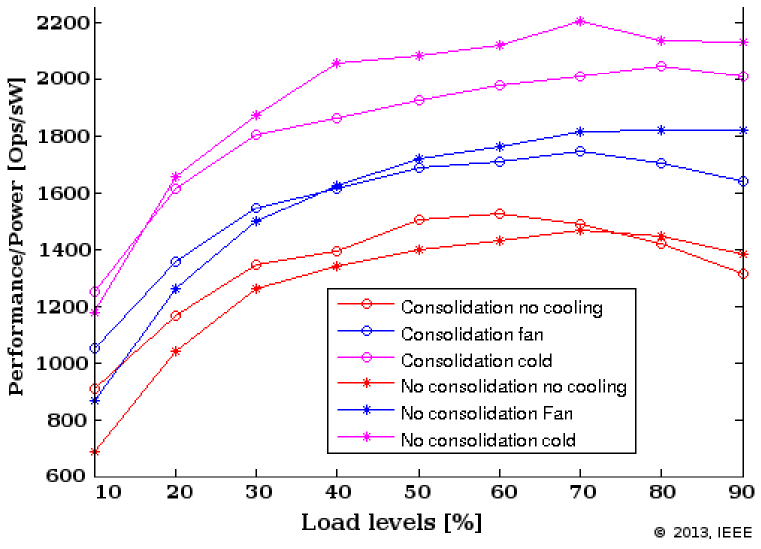


Figure 4.8: A comparison between CFS and OGS. Y-axis shows energy consumption in Performance/Power (higher is better) for three scenarios using different ambient temperatures.

With two completely opposite mapping strategies used in CFS and OGS, the energy efficiency still depends on the workload applied to the system. As the CFS occasionally causes an unnecessarily high static power dissipation, the OGS increases the dynamic power dissipation more than the CFS. With different hardware architectures using different number of cores, the crossover point is clearly volatile. Results from the experiments suggest that a middle way between fair scheduling and load consolidation should be defined according to the type of workload and the hardware platform used.

4.4 Summary

Task mapping is vital to power management in many-core processors because the distribution of work execution influences the total power balance between static and dynamic power. An energy efficient execution should minimize the total power by selecting the optimal number of processing elements for mapping the present workload on. With current architectures, experiments have shown that the overhead of communication using task migration is not significant unless the L2 cache misses are of great order. Practically, unless the power management system is operating on sub-milliseconds granularity, the task migration overhead is not a breaking point. The following chapter presents an approach to reduce the total product by introducing additional intelligence in the applications.

Chapter 5

Generic Performance Metric

"*Are you Herr Professor Heisenberg? I am here to prove you wrong!*"

— Grete Hermann - 1934

In order to efficiently balance the task execution in a many-core system and perform resource allocation, the runtime system requires knowledge about the performance of the applications and their relation to resource usage. The balance of workload is dependent on **a)** the behavior of the applications, **b)** available hardware resources and **c)** a balancing policy.

Traditionally, the allocation of resources and the balance of task execution is purely based on the level of CPU utilization called the *workload* level. In this chapter the concept of workload is defined and the shortcomings of using the workload level as metric for resource allocation are pinpointed. Furthermore, the concept of application specific performance is introduced as a more descriptive metric used for resource allocation.

5.1 The Definition of Workload

Workload has traditionally been the metric according to which task scheduling and mapping has been based. The workload is overall a metric upon which to measure the utilization of the CPU, i.e. the usage factor. Every task currently mapped on a certain core influences the workload level, and a common load balancing strategy is to limit the amount of task executions on a single core in order to not cause overloading.

Optimally the load contribution of each task should be obtained in order to optimize the mapping. However, tasks usually contain unpredicted behavior using I/O and memory access which causes difficulties in predicting the exact load. A common strategy has therefore been to measure the combined load of all tasks on a single processing element instead of individual task load. The total load can be used as a measurement of processing availability or "*space left on the core*". The following paragraphs summarize a set of methods to measure the load in Linux.

Load window On the finest granularity level inside the OS, a CPU is either actively executing instructions or idle in the idle loop. A load window is a measurement of the ratio between the active and the idle status of a CPU over a given time window. By adding the abstraction of load, the CPU can be modelled as semi-active listed as a percentage; this percentage determines the available capacity of the CPU.

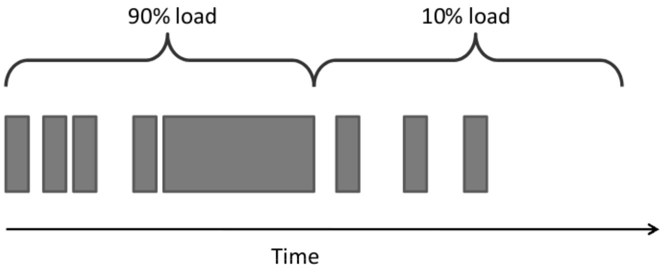


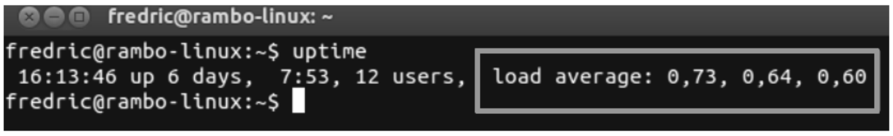
Figure 5.1: Load calculated as a ratio between active and idle for a defined window (Gray rectangles represent an active CPU).

Figure 5.1 illustrates two time windows with gray rectangles representing execution and blank slots representing idling. The left time window is loaded to 90% and the right window is loaded to 10% based on the ratio between active and idle. Measurements by load windows is usually a sound method for illustrating the load to the user since the visualization is considered as human interpretable.

Run-queue length By measuring the load according to a load window, a defined time scale must be used since the average is always related to the time window. A more fine grained method is to measure the length of a run-queue i.e. the number of runnable tasks in a RQ. A higher number of runnable tasks indicates a heavier load on the CPU and a lower number indicates a lighter load.

Even though the RQ length is per-ce not usable in an illustrative context, scheduling mechanisms and mapping decisions on a fine time scale can use the RQ length as load metric. As explained in Section 4.2, multi-core systems use a dedicated RQ per core. This allows load balancers to determine the task mapping when using several cores.

Load average More slow phased methods to measure the load is by load averages. Linux provide access to load average using the `top` or `uptime` commands. Figure 5.2 shows the output from `uptime` in which the load averages are indicated by the rectangle. The load averages are presented as triplets using moving windows of 1,5 and 15 minutes. Load averages



```
fredric@rambo-linux:~$ uptime
 16:13:46 up 6 days,  7:53, 12 users,  load average: 0,73, 0,64, 0,60
fredric@rambo-linux:~$
```

Figure 5.2: Load displayed by using the `uptime` command in Linux.

is usually not a feasible measurement for rapid scheduling decisions, but more useful in statistical analyses, load predictions etc. Figure 5.3 shows the load average for fully utilizing the CPU with the `stress` benchmark for 30 minutes after which the CPU was idling for 30 minutes. As seen in the figure, the one minute window fastly reaches the maximum load average and the 15 minutes windows reaches the peak after 30 full minutes.

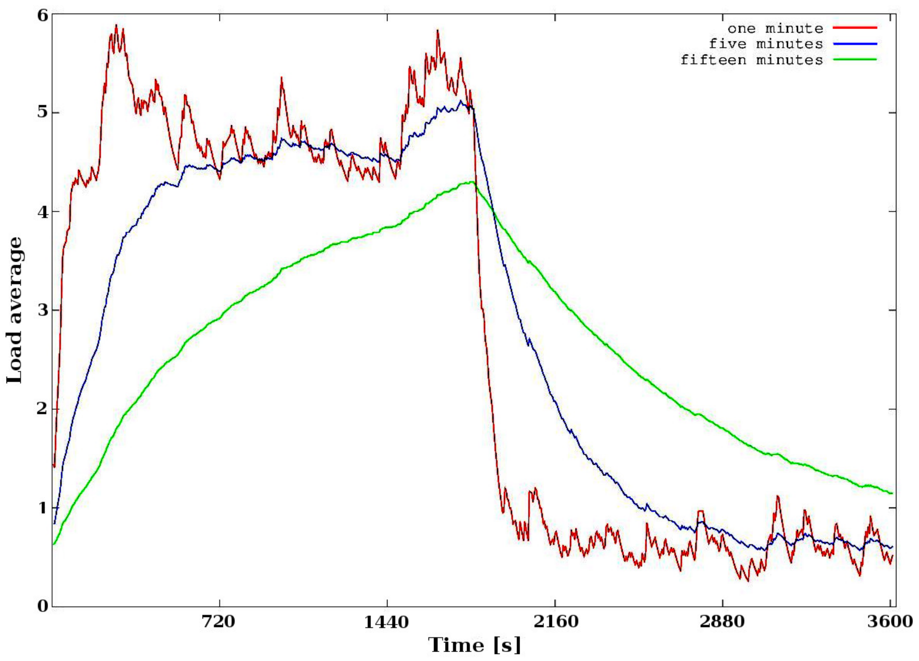


Figure 5.3: Load calculated as an average based on the system uptime.

5.2 Problems with Workload

The problem of using workload as a basis for resource allocation is the lack of software insights. While workload is a measurement of resource utilization, the allocation of resources is not based on software performance requirements. This means that resources are allocated only according to the indirect impact of the software.

The frequency governors presented in Section 3.3.1 regulate the clock frequency according to a workload threshold in order to keep the workload on a desired level. By using workload thresholds as basis for clock frequency regulation, software imposing a high workload also impose a high clock frequency even though not needed. On the other hand, software containing heavy I/O resulting in a low workload might suffer from performance degradation as the workload threshold is never met.

5.2.1 Race-to-Idle

A practical example of unnecessary resource allocation is the concept of Race-to-Idle [70] in which the CPU is using the maximum clock frequency to maximize the performance and minimize the energy consumption. Since maximizing the performance minimizes the execution time, this execution strategy was considered energy efficient in older generation smart phones. The early smart phone, e.g. the iPhone3, used an ARM Cortex-A8 based single core CPU. The power dissipation for different clock frequencies for such a CPU was measured and shown in Figure 5.4. As seen in the fig-

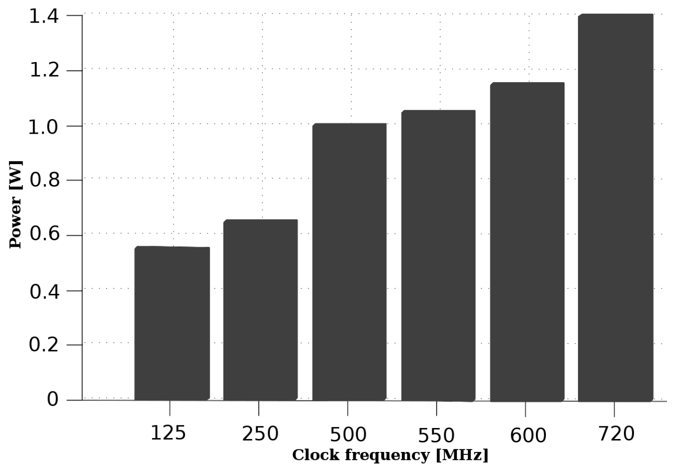


Figure 5.4: Power dissipation for a Cortex-A8 CPU using different clock frequencies (full load).

ure, the highest clock frequency (720 MHz) causes roughly 1.4W of power dissipation. When scaling down the frequency roughly 3x (250 MHz), the power dissipation is only reduced by 2x (0.7W), which means that the total energy consumption is less when executing at the higher clock frequency (if assuming that the speed-up is linear to the clock frequency).

Moreover, the same experiment was conducted using a modern multi-core ARM Cortex-A15 CPU and the results are shown in Figure 5.5. As the CPU is clocked to 1800 MHz and uses four cores for execution, roughly 5W of power is dissipated. In contrast to the Cortex-A8, when scaling down the frequency 2x (900 MHz) the power dissipation is reduced by roughly 4x (1.3W). This means that the energy consumption is lower using the lower clock frequency for a longer execution time.

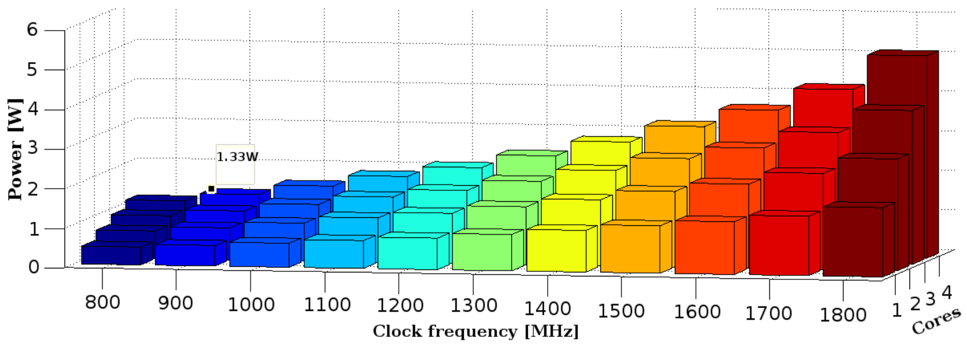


Figure 5.5: Power dissipation for a Cortex-A15 CPU using different clock frequencies and different numbers of cores (full load).

As Race-to-Idle was an energy efficient option in last generation processors, the strategy is less efficient using modern processors. This is because current generation microprocessors use higher clock frequencies leading to higher dynamic power and more cores leading to higher static power. With a more exponential power curve, (Figure 5.5) Race-to-Idle becomes less efficient and the highest clock frequencies should only be used in case the performance is required.

Race-to-Idle is, however, an easily implementable execution principle since the resource allocation is completely transparent to the programmer – the only task of the programmer is to assess the functionality of the software, and the responsibility of resource allocation is left to the workload-based frequency governors. The result of such implementations is a system executing periodically on a high clock frequency and periodically idling. A more energy efficient execution strategy is to continuously execute on an intermediate clock frequency and eliminate the slack time idling.

5.3 QoS – The Generic Performance Parameter

In computer systems, *application performance* is the parameter directly related to user satisfaction. The metric is described as throughput, latency, execution time, memory usage etc. – generally any kind of metric describing the form of performance required by the user is considered. As long as the user expectations are satisfied, the software is fulfilling its purpose and over allocation of resources is considered as waste.

5.3.1 QoS-aware execution

Instead of Race-to-Idle, computer systems should allocate resources according to software performance. Such an example is easily given by considering a typical video decoder in a video player. The decoder translates decoded video frames into a frame format viewable on a display. The execution of a video player is typically performed as follows:

- Video frames are decoded from a source.
- A decoded frame is put in a buffer of size n. In case the buffer is full, the decoder is waiting.
- The video player picks a frame from the buffer every $\frac{1}{25}$ of a second (25 frames per second) for the output display.

As soon as the frame decoding is initiated, the workload increases since the CPU becomes busy. By using workload as basis for resource allocation, the clock frequency is increased due to the busy CPU. With increasing clock frequency, the workload is still kept high since the decoder is continuously fed with new frames to decode. The CPU is hence decoding frames on the highest clock frequency at a *too* fast rate as illustrated in Figure 5.6 (A). When the frame buffer is full, the decoder is halted and the CPU remains idle until the buffer has been emptied by the video display. In this scenario, the frames are decoded at a rate of e.g. 50 fps whereas 25 fps is sufficient for displaying the video.

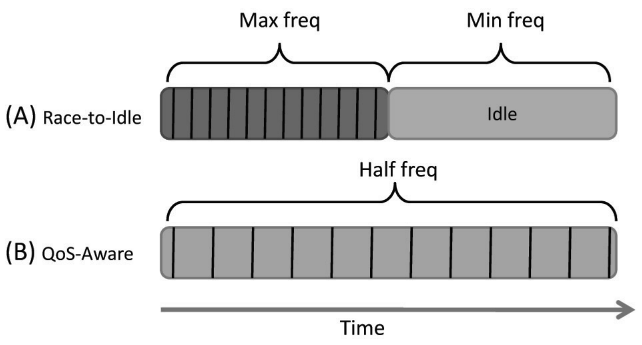


Figure 5.6: A: Race-to-Idle execution using the maximum clock frequency during computation and the minimum clock frequency during idling.
B: QoS-aware execution using a mid-range clock frequency for the whole execution and no idling.

Even though the lowest clock frequency can be used during the idle periods, the active periods are wasting power due to the very high power dissipation when using high clock frequencies on modern multi-core CPUs (see Figure 5.5). Part (B) in Figure 5.6 illustrates the QoS-aware execution in which the CPU is executing only "**as fast as needed**". By stretching the execution over the whole time window and eliminating the slack time, an intermediate clock frequency can be constantly used. In comparison to the power dissipation obtained from Figure 5.5, executing at 900 MHz for time t is significantly more energy efficient than executing at 1800 MHz for a time $\frac{t}{2}$.

5.3.2 Power management with QoS metadata

Since workload is solely a measurement of CPU utilization, direct application performance cannot generally be derived, hence the resources cannot be efficiently regulated. Instead, meta-data in form of direct performance should be used as basis for resource allocation.

Resources should be allocated according to the current performance level achieved by the application compared to a pre-set aim called the *setpoint* S. The setpoint indicates at what performance level an application *should* execute at. In the video decoder case, a suitable setpoint is 25 fps since the video output display is anyway operating at this frequency because of pre-defined video standards. With a defined setpoint and the current performance Pr explicitly monitored, the QoS can be expressed as:

$$QoS = \frac{Pr}{S} \tag{5.1}$$

as a percentage $> 0\%$. A QoS value in range [0 99]% indicates lack of performance, and more resources should be allocated in order to increase the QoS value. QoS values $> 100\%$ indicates resource excess, and the resources should be scaled down to save power.

The QoS metric is measured as the ratio between the requested setpoint and the actual performance. This makes the notion unitless and application performance in any metric can be used as basis for resource allocation. With the gained insights into resource allocation, next chapter presents the construction of the QoS manager used for energy efficient execution.

5.3.3 Related performance metrics

Eyerman et al. [22] claim that no single throughput metric is fundamentally generic for multiprogram workloads. Performance should instead be expressed as related to the internal single case-study; a direction adopted in this thesis. A high-level language CQML [1] was suggested for describing QoS requirements integrated in UML. CQML links a high level QoS description to system performance, and can describe system actions based on the result. Applications specify a performance setpoint and a lower bound acceptable performance level in context of the application. Applications then monitor own performance and signal this value to the QoS manager periodically. Similar notations inspired by the language has been used in this thesis to describe QoS in applications, but more focus has been put on the link between applications and hardware resources in a single computer system.

Hoffmann et. al propose *heartbeats* [27] as a generic performance parameter. The heartbeats are setup by including a set of heartbeat API calls in applications, which are used to monitor the application performance. By calling the heartbeat API on suitable places in the applications such as large loops, a notion of the update interval between API calls is created. The heartbeat API registers multiple applications and the outside system monitors the heartbeat of each application separately. Heartbeats is a suitable candidate, and fully compatible as a performance parameter in our system. An application can register a setpoint heartbeat after which the heartbeat monitor is used to derive the actual performance in heartbeats. Earlier work by Vetter et al. [86] presents a similar approach, but by including performance assertions directly into the code. Based on the assertions, the application can adapt itself in case significant performance is not achieved. The system allowed, however, only internal monitoring of the performance, and a runtime system was not in the scope.

5.4 Summary

Software should not Race-to-Idle on modern microprocessors, because the total power increase of executing at the highest clock frequency is usually more than the reduction in execution time. With the energy consumption as a product of power and time, less energy is consumed when executing at a lower clock frequency for a proportionally longer time. As Race-to-Idle is the currently supported execution strategy, the notion of QoS has been introduced to the applications in order to support user defined execution speed. By using QoS as the metric for regulating the performance, clock frequency scaling with relation to software execution requirements is possible, and only the necessary amount of resources are allocated for the execution. The following chapter presents the interface between the applications and the resources, which enables the QoS-aware execution.

Chapter 6

Bricktop: The QoS Manager

"*Hence the expression: as greedy as a pig.*"
— Brick top, Snatch - 2000

The notion of QoS was introduced in previous chapter as a concept for more specifically describe application performance compared to measuring the workload level. With a more detailed description of resource requirements, the hardware can be more closely scaled according to the software demands. This creates a more energy efficient execution since resource excess is minimized compared to the traditional approach.

In order to provide the runtime resource allocation based on QoS, a new resource manager was implemented. This chapter describes the structure of the *Bricktop QoS manager*. *Bricktop* was implemented as a centralized point for resource allocation, and is directly available to the applications. It uses DVFS and DPM to minimize the power dissipation for a system with a set of applications. A declaration language is used to specify QoS- and performance requirements in the applications as well as indications related to efficient hardware usage. *Bricktop* can be utilized by any application capable of declaring resource requirements, and capable of monitoring own performance.

The chapter is based on [34] (Paper V), in which further implementation details and measurement results are presented. Chapter 7 and 8 later cover the QoS manager parts in more detail while this chapter focuses on the overall structure.

6.1 Declaring the QoS Metric

The aim of QoS-based resource allocation is to more accurately regulate the hardware according to the actual software requirements. Because of this concept, the applications must follow two programming paradigms:

1. *The applications must be able to declare* **performance** and **QoS requirements**. The performance requirement is a defined setpoint according to which the application is allocated resources, and the QoS requirements determine the maximum acceptable deviation from the setpoint as a percentage. For example, a video decoder requesting 25 fps with a QoS level of 95% defines performance values between 23.75 fps and 26.25 fps as acceptable. In case the performance is below the lower QoS limit, more resources are allocated, and in case the performance is above the upper QoS limit, resources are de-allocated to save power.

2. *The applications must be able to* **measure** performance. Since the interface between the applications and the QoS manager accepts arbitrary performance metrics, the application must control performance measurements. The implementation for achieving the measurement is application dependent, and the only requirement is to obtain a suitable metric representing the performance. Measurements should be periodical and the application should utilize the actual interface call to transmit the measurement data to the QoS manager.

Both QoS declaration and measurement values are transmitted to *Bricktop* via simple library calls, which automatically optimizes the resource allocation with DVFS and DPM (more details is next chapters). The communications infrastructure and the structural overview of the *Bricktop* environment is presented in the following sections.

6.2 Structure of the Runtime System

The *Bricktop* QoS manager is a centralized service to which one or more applications can connect, declare QoS requirements and establish a communications flow.

6.2.1 The *Bricktop* structure

Figure 6.1 illustrates the structure of the environment as a *sensor-controller-actuator* setup in which the sensors are providing information to a controller which regulates the actuators. These three parts are defined as follows:

Sensor A sensor is a unit capable of connecting, expressing QoS and sending measurements to *Bricktop*. A sensors is capable of measuring own performance with an implementation specific mechanism in the sensor itself. There are no requirements on how the sensor measures the performance as long as it is able to send it to *Bricktop*. In Figure 6.1 three sensors are connected to the controller: *web server*, *video player* and *LDPC decoder*. Each sensor is expressing performance using a unique metric; later translated into a generic QoS metric in the QoS manager.

Actuator Actuators are units capable of indirectly altering the sensors' performance by regulating resources. The number of actuators and the functionality of the actuators varies between platforms. In this thesis DVFS and DPM (Chapter 3) are assumed as available actuators. An actuator is regulated by setting the *magnitude* relative to maximum value e.g. setting the clock frequency to 1.4 GHz out of the maximum 1.6 GHz.

Controller The controller is managing the connection between sensors and actuators and adjusts the actuators based on the measurements from the sensors and defined QoS requirements. The input to the controller is firstly translated into a generic QoS metric after which the actuator magnitude is set. The controller optimizes the balance between power efficiency and performance based on control policies, input priorities and an optimizer further described in Chapter 8.

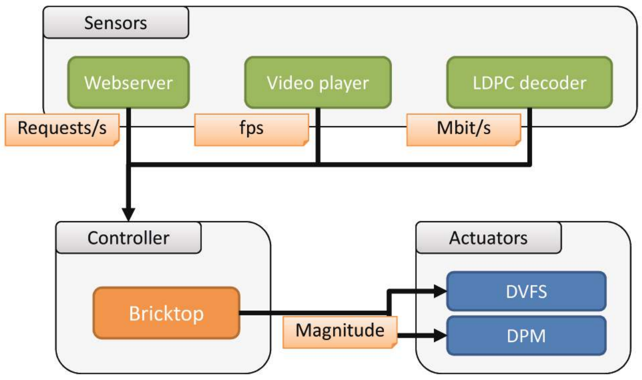


Figure 6.1: Overview structure of the *Bricktop* QoS manager.

The communications flow between sensors and the *Bricktop* QoS manager is handled in form of data packets which include a meta-data header and the measurement information. The header identifies the application, thus the performance metric used. It also contains a priority field which is used to declare the importance or the weight of the measurement data in case resource conflicts occur.

The other part of the data packet contains the measurement data provided by the application (sensor). The communications link between sensors and controller is handled by a queue structure capable of buffering a set of data packets referred to as tokens as illustrated in Figure 6.2. A sensor is sending a single token into the buffer in case the buffer is not full. The token is discarded otherwise since an outdated token contains outdated performance values unusable for the controller. The *Bricktop* QoS manager pulls all tokens periodically from the buffer and the optimization is performed on all received items within the period. In case no tokens are obtained, no resource allocation or de-allocation is performed within this period.

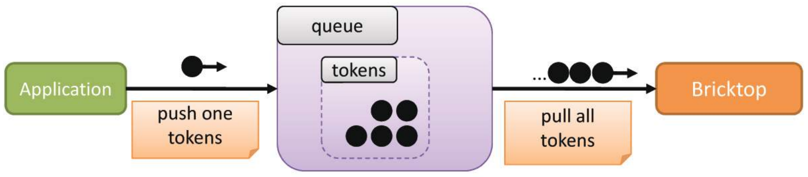


Figure 6.2: Communications flow from applications to *Bricktop*. A buffer is used to store data tokens, which *Bricktop* is regularly pulling.

6.2.2 The *Bricktop* controller

The controller part from Figure 6.1 is here detailed in Figure 6.3. The performance value from a sensor is transmitted to the *Bricktop* interface (Figure 6.3 [Green box]) which translates the value into a QoS value by comparing the ratio of the performance value $Perf$ to the setpoint S as $QoS = \frac{Perf}{S}$. Since the performance and the setpoint always have the same metric, the QoS value is unitless. For example $QoS = \frac{23fps}{25fps} = 0.92 = 92\%$.

In case the QoS value is *not within* the acceptable QoS limits, the derived QoS value is passed to a resource optimizer (Figure 6.3 [Orange box]), otherwise it is discarded. The task of the optimizer is to find the optimal resource allocation or de-allocation based on the current resource setting and the QoS value derived from the sensor(s). The optimization result is then passed to an update function (Figure 6.3 [Blue box]) which updates the actuator magnitude according to the result from the optimizer.

Finally, the output from *Bricktop* in form of actuator magnitude are passed down to the resource management of the OS and the resources are allocated. The actuator magnitude is defined in the form e.g.: ***use*** *2 cores* ***clocked*** *to 1.2 GHz.* In case of Linux based systems, the frequency governor (Section 3.3.1) and CPU hotplug (Section 3.3.2) interface are used for controlling the hardware.

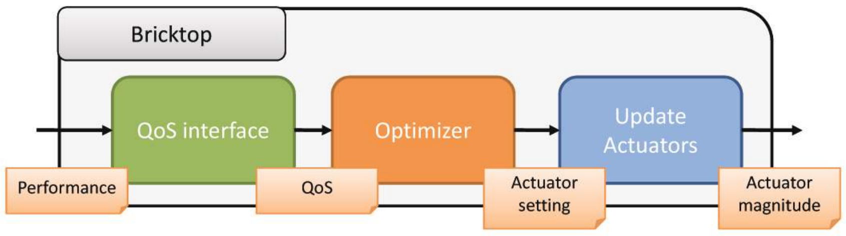


Figure 6.3: *Bricktop* consists of three main parts: QoS interface, power optimizer and actuator interface to the power saving mechanisms.

In order to achieve efficient resource allocation three important aspects of the controller must be kept:

1. Since two actuators are assumed in this thesis (DVFS and DPM), the controller must be able to *determine the most energy efficient balance between using DVFS and DPM while not violating the performance requirements in the applications.* Since DVFS and DPM have different impacts on both power dissipation and performance, the controller must use the provided information in the applications and the environment to regularly calculate the optimal balance.

2. *Resources must be allocated/de-allocated with a short latency.* Both the energy efficiency and the performance of the system is completely dependent of the capability of reacting fast enough in cases of non-adequate QoS in an application. The controller must therefore contain a control algorithm capable of quickly reacting to changes in the input.

3. *The controller must be stable.* A common consequence of tuning controllers for fast reaction time is controller instability. An instable controller will produce heavy alternations in the output signal due to the tuning parameters in the controller. While the controller is able to react quickly to the input signal, stability must be achieved for energy efficient and performance prone execution.

The next chapters provide more insights into the optimizer used to determine the DVFS / DPM combination. Chapter 7 presents an approach to model a multi-core system as a mathematical representation used for control decisions. Furthermore, Chapter 8 presents the design and implementation on a multi-variable optimizer used to regulate the resource balance based on the application inputs.

6.2.3 Related runtime systems

The PowerDial [28] approach allows graceful degradation in applications based on current application performance measured in heartbeats [27]. The system transforms application parameters (such as peak-signal-to-noise in a video) into dynamic control variables stored in the application itself. A callback function is inserted into the application using which the controller is able to adjust the control variables according to performance and policies. Figure 6.4 illustrates the PowerDial system in which a Performance Goal (setpoint) is used to drive a controller which selects the dynamic control variables in the applications. A heartbeat feedback monitors the execution and reports on the updated performance of the application. Also, the

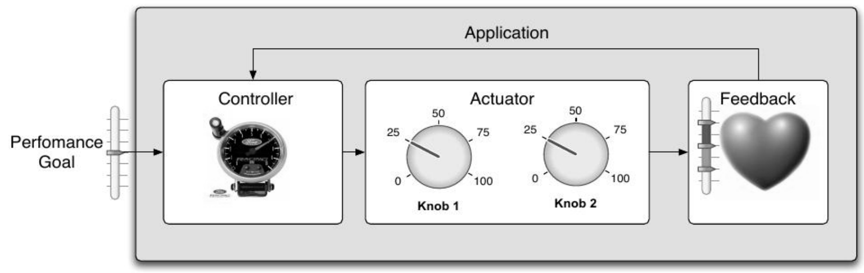


Figure 6.4: Overview structure of the PowerDial system [28].

work by Segovia [72] suggests graceful degradation of the application QoS by monitoring a happiness value from the application. Based on this value, the runtime system can degrade quality points in the application in order to achieve the requested QoS. The *Bricktop* QoS manager uses the same approach to treat input signals from applications: the performance is transformed into a generic parameter – QoS – upon which the controller acts. In contrast, *Bricktop* uses no graceful degradation in the applications, but hardware actuators to drive the power management.

StarPU [6] is a runtime system developed at INRIA Bordeaux. The development framework allows task kernels for heterogeneous architectures, for example CUDA or OpenCL, which in turn are scheduled by the StarPU runtime. The scheduling is based on a queue structure for each device and a performance model between the defined task and the hardware device. During initialization, StarPU is able to benchmark the tasks and their dependable parameters, such as data type and data size, on the available hardware. The initialization builds the performance model according to which StarPU selects the most performance efficient device to schedule a task on. The schedule is either completely transparent, or is influenced by the user who is able to select scheduling policy and task priority. StarPU is currently implemented for high performance in heterogeneous systems, while an energy efficient scheduling policy is under consideration. In contrast to StarPU, the *Bricktop* runtime system leaves the scheduling decisions completely to the OS, and the focus is on hardware resource allocation to minimize the power.

For efficiently regulating the power with both DVFS and DPM, a three step mechanism was used in the work by Ghasemazar et al. [23]. It firstly selected the number of active cores and secondly an optimal clock frequency for the active cores and finally task assignment. In a similar fashion, Marinoni et al. [53] choose to calculate the minimum frequency and the maximum sleep time allowed in a real-time scheduler to minimize energy consumption. HyPowMan [8] uses a set of policy experts to either optimize according to DVFS or DPM depending on processor state. In contrast to the mentioned runtime systems, *Bricktop* performs optimizations for both DVFS and DPM in a single run in order to determine the optimal *combination* of both methods. Hence, no power saving technique is favored over the other, since the decisions are based on the models.

Three design choices for a MPSoC runtime manager was presented in the work by Nollet et al. [58]. The complete system consisted of a *quality manager* which is capable of selecting operation quality points in the applications similar to PowerDial [28]. The quality manager is able to communicate with a *resource manager* which ultimately allocates the resources. Figure 6.5 shows three different setups:
a) Applications communicate only with the quality manager, which invokes the resource management transparently to the user.
b) Applications communicate with the quality manager, but also hint the resource manager using meta-data. This setup engages more precise resource control based on application input at the cost of increased complexity.
c) Applications communicate only with the resource manager. No quality operation points are allowed and application performance is only based on the resource allocation.

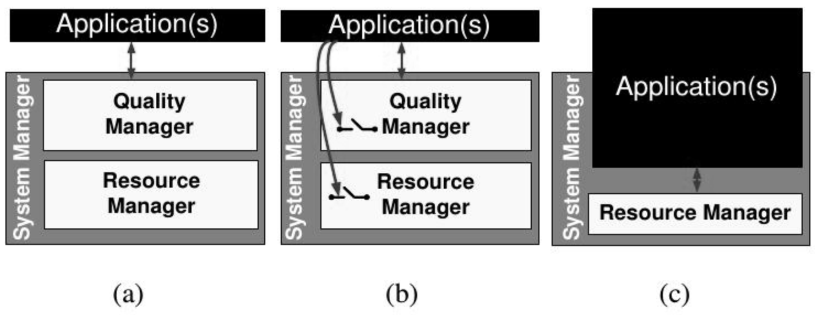


Figure 6.5: Three different design choices as defined in [58].

Bricktop adapts version **c)** using which application communicate directly with the resource manager and quality degradation of applications is not allowed as long as resources are available.

Chapter 7

System Identification

“*I’m sorry, Dave. I’m afraid I can’t do that.*”
— HAL, 2001: A Space Odyssey - 1968

Resource allocation in computer systems has a direct implication on the power dissipation of the platform. By multiplying the time of actively executing processing element by its power integral, the energy consumption can be determined. As recalled from Chapter 3, a many-core system can either provide resources in form of processing elements by waking up cores from a sleep state or by increasing the clock frequency of already active processing elements. Increasing the number of processing elements has a greater impact on the static power dissipation while the clock frequency impacts both the dynamic power and the static power.

In order to optimize the resource allocation, the underlying system should have an understanding of the implications related to power dissipation and application performance. A modelling concept is presented in this chapter which aims to describe the consequences of resource allocation with respect to power and performance. A power model is created in order to determine the power dissipation based on the actuator (DVFS and DPM) utilization called *magnitude* and a performance model is created to determine the effects on the speed-up. Both models are used in the *Bricktop* controller briefly described in Chapter 6 and further detailed in Chapter 8.

7.1 Modelling Power

By modelling the power dissipation of a microprocessor, the physical activities and power dissipation of the semiconductor can be more easily integrated into software algorithms in a computer program. A power model is used as a vehicle for understanding the correlation between resource allocation and power dissipation i.e. “*how much power does n-resources use?*”.

A representative, yet simple model describing this relationship is therefore required.

7.1.1 Previous work: Bottom-Up Approaches

Power has previously been modelled in multi-cores [11, 13, 21, 55, 64, 66, 70, 73, 74, 82]; the power models are usually constructed from bottom-up approaches according to which the power is estimation based on mathematical approximations of the platform and its behavior. An accurate power model which reflects both the software execution and external conditions such as temperature is usually difficult to create. Parallel software and OS scheduling in parallel systems is particularly difficult to model as power dissipation, especially with a real-world power manager executing inside the OS.

Cho et al. [11] presented a mathematical formulation on the interplay of parallelism and energy consumption. Their work provides analytical optimizations for tuning clock frequency and parallelism depending on application characteristics. The optimization algorithms was used to calculate values for the ratio between dynamic and static power for a given architecture together with application parallelism and performance to find the optimal configuration for energy efficiency. A similar approach is presented by Rauber et al. [66], in which the total energy consumption is modelled as a linear combination of the static and dynamic power. The dynamic power is based on the voltage and frequency level of each core which in turn depends on a set of tasks executing on the respective core. Each task is assumed to be homogeneous and completely independent of every other task. The model hence exploits a fork-join pattern in parallel tasks and the synchronization over to simulate the power dissipation.

C-3PO [70] is a power manager used to maximize performance under power constraints and minimize peak power to reduce energy. The power model used is based on a set of proportionality constants representing the chip and an estimated static component C, which is simply modelled as the idle power of the system. Applications are given a power budget, which is used for resource allocation in form of clock frequency and the number of cores. The manager activates cores to more parallel program and clock frequency scaling to more serialized applications. The work by Tudor et al. [82] focus more on the real parallelism of a set of tasks bounded by memory contention and data dependencies between tasks. The available parallelism is thus modelled as fraction of total parallelism with respect to the platform and the number of used thread. Power, on the other hand, was modelled as a linear combination of *idle power*, *active power* and *memory power*.

A further detailed performance model for microprocessors based on hardware and software characteristics is presented in Eyerman et al. [21] upon which the CPU power can be estimated. The model accounts for specific microarchitectural details such as pipeline characteristics, caches, branches etc. in combination with software behavior implications on the hardware such as instruction dependencies, instruction types and the number of instruction types.

7.1.2 Top-Down Modelling

In contrast to most related work, our power model was derived from real-world experiments during which the model was created according to the real power output instead of the physical composition of the system. The advantage of a top-down approach is a more realistic view of the complete system including cores, buses, memories, temperature, operating system and other software, which is very difficult to model using analytical bottom-up approaches. The approach requires, however, one set of experiments for each type of platform. This means that the experiment setup should preferably be easily executed, repeatable and sufficiently fast.

Experiment-based design As an example, a quad-core Exynos 4412 platform using the ARM Cortex-A9 CPU was modelled. To derive the power model, the system was fully loaded using the `stress` benchmark while both actuators (DVFS and DPM) were step-wise increased. The power dissipation was measured for each step until all combinations of clock frequencies and number of cores have been evaluated. Figure 7.1 shows the result from

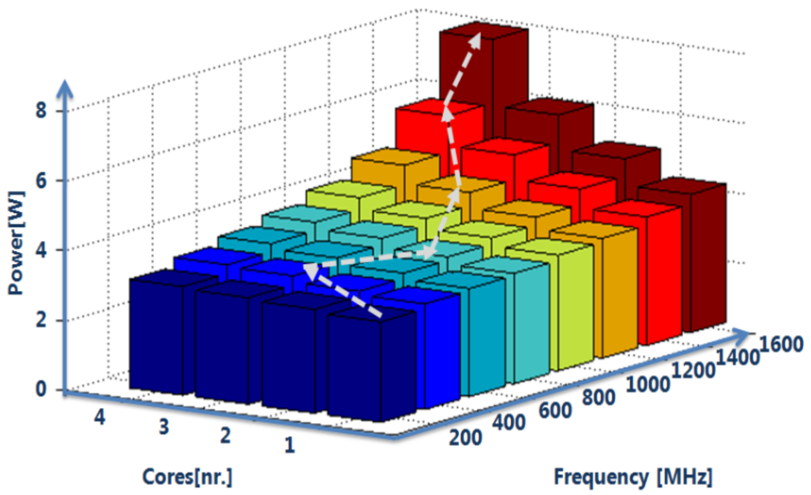


Figure 7.1: A top-down model of a quad-core ARM Cortex-A9 CPU. The figure shows the power dissipation of the CPU during full load using different configurations of DVFS and DPM.

this experiment: the power dissipation is low when using few cores and low clock frequencies, and increase according to different ratios as the actuator magnitudes increase.

From the experiments shown in Figure 7.1 it is clear that a high number of cores and a high clock frequency results in a very high power dissipation, and these combinations should be avoided as long as the performance is sufficient. The power model is hence used as a "*road map*" using which the control system can decide the most power efficient *path* from low- to high performance. The white arrows in Figure 7.1 illustrates one possible path as the performance requirements for an application increases from low to high.

Derivation of mathematical representation With the results shown in Figure 7.1, a mathematical representation was created in order to allow the integration in the controller. A similar approach as [69] was used, in which a two dimensional plane $(DVFS, DPM)$ was fitted as a function of the power dissipation. The control variables for DVFS and DPM were denoted as **q** and **c** respectively. Since these variables are only used as control variables in the optimization algorithm, the variables are unit-less and chosen in the range [1 - 8] where 1 is minimum utilization and 8 is maximum utilization of a specific actuator. The goal is to define a surface as close as possible to the data values in Figure 7.1, which includes the control variables.

The following third degree polynomial defines the surface:

$$P(q,c) = p_{00} + p_{10}q + p_{01}c + p_{20}q^2 + p_{11}qc + p_{30}q^3 + p_{21}q^2c \tag{7.1}$$

The parameters p_{xx} are fixed coefficients used to represent the quad-core Exynos 4412 use-case. Levenberg-Marquardt's algorithm [40] for multi dimensional curve fitting was used to find the optimal coefficients. This algorithm minimizes the error between the model and the real data by tuning the coefficients. Values for the coefficients are found in Table 7.1, and the obtained surface representing the mathematical model is illustrated in Figure 7.2.

Table 7.1: Coefficients for the power model.

p_{00}	p_{01}	p_{10}	p_{11}	p_{20}	p_{21}	p_{30}
2.34	0.0576	0.598	-0.0248	-0.1605	0.0097	0.0120

Model verification In order to obtain efficient resource control, the power model used to represent the system should be significantly accurate compared to the real-world power dissipation. An inaccurate model can predict a suboptimal actuator combination for a given application performance, for

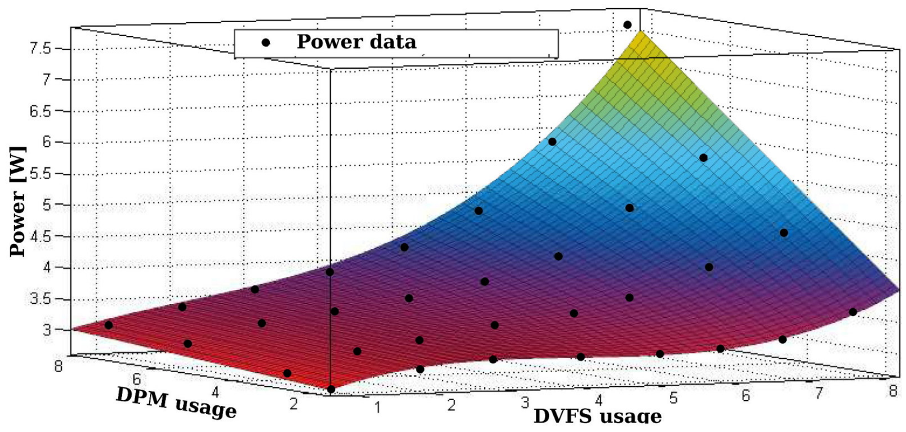


Figure 7.2: A mathematical representation of the power values in Figure 7.1 derived by using surface fitting methods.

example "*use 2 cores clocked to 1200 MHz instead of using 3 cores clocked to 1000 MHz*". The accuracy model was determined by calculating the difference between the real data and the derived model as the *model error*. The results are shown in Figure 7.3 in which the lines are the predicted model and the rings are data. For this use-case, a maximum error of 10.2% was obtained, but with an average error of 0.6% and with a computationally simple model, we considered the model feasible for our experiments. A more accurate model can be achieved by using a higher degree polynomial with the cost of increased computational complexity.

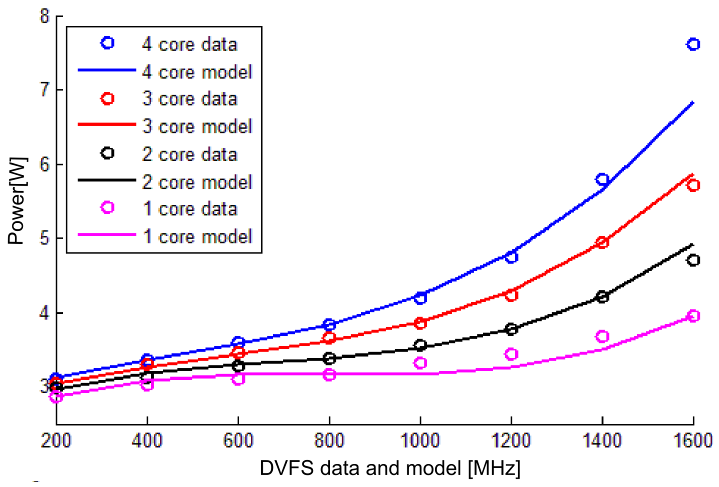


Figure 7.3: A comparison between the data points in Figure 7.1 and the mathematical representation in Figure 7.2.

Discussion

A top-down power model was chosen instead of the traditional bottom-up approaches based on analytical expressions used to model the hardware under different conditions. The top-down model is created by stressing the system to full load after which the real power dissipation is measured. A down side to this approach is clearly the required benchmark run for each new platform. On the other hand, the top-down model represents the real-world execution on a more practical level than a bottom-up model. Ambient temperature differences, for example, change the power curve significantly [39] – an effect difficult to model in practice with bottom-up models.

7.2 Modelling Performance

With resource-to-power model, *Bricktop* requires a model of application speed-up based on actuator magnitude to determine the most energy efficient actuator combination. With a performance model, the system is capable of allocating the *necessary* amounts of resources which results in *minimal* power dissipation.

7.2.1 Speed-up response of actuators

With two actuators (DVFS and DPM) considered, a multi-variable optimization routine is used to decide the optimal actuator configuration by considering the utilization parameters q (DVFS) and c (DPM). The speed-up is defined as application performance compared to the minimum actuator setting i.e. 1 active core clocked at the minimum clock frequency.

While acknowledging that predicting the exact speed-up of a general application is a difficult task because of, memory latencies, disk I/O, OS influences, user input etc., the abstraction of speed-up was risen to the level of two functions: clock frequency and parallelism. The speed-up using DVFS was modelled as a linear combination of clock frequency q as:

$$\text{Performance}(\text{App}_n, q) = K_q \cdot q \tag{7.2}$$

where K_q is a constant. This means that e.g. 2x increase in clock frequency models a double in speed-up.

Modelling speed-up with respect to the number of cores was, however, considered more difficult since the result depends highly on the inherited parallelism and scalability in the program. For example a sequential program does not increase its performance by adding additional cores, while a parallel application could save energy by increasing the number of cores instead of the clock frequency. Because of this issue, the notion of expressing parallelism directly in the applications was added as a used defined *P-value* in the range [0 1]. A value of 0.0 represents a completely sequential program phase and 1.0 is an ideal parallel program phase. With the P-value, Amdahl's law is used to model DPM:

$$\text{Performance}(\text{App}_n, c) = K_c \cdot \frac{1}{(1-P) + \frac{P}{c}} \tag{7.3}$$

where K_c is a constant and c is the number of available cores and P is the P-value. Amdahl's law models a high performance increases as long as the number of cores is low but decreases as the number increases as a logarithmic function. Hence, the speed-up becomes sub-linear as more cores are added, and eventually increasing DVFS instead of the number of cores becomes more energy efficient. The total performance is the sum of the two models as:

$$\text{Performance}(\text{App}_n, q, c) = \text{Performance}(\text{App}_n, q) + \text{Performance}(\text{App}_n, c) \tag{7.4}$$

7.2.2 Techniques for Obtaining Parallelism

Typical applications usually have different behavior depending on the phase of execution. For example whilst in an initialization phase, the application is single threaded and sequentially executing. On the other hand, the computational phase of the application can often be parallelized to utilize more cores. The different levels of parallelism in applications is accounted for by dynamic P-value injections during runtime. This means that the power manager is continuously aware of the resources and to what extent applications are able to utilize them.

While injecting dynamic P-values during runtime is simply a matter of updating the parameters in the power manager call, obtaining the correct P-value in the application can be a complex task. The parallelism in applications may depend on the usage of tasks/threads, application I/O, memory accesses etc. Many factors contribute therefore to the obtained performance scalability of the application in a many-core system. In case the parallelism is not accurately obtained, the power manager might not find the optimal solution. Therefore it is important to at least have a rough estimate of the parallelism, which can be done by several methods.

The work in [12, 57, 81] demonstrate methods to measure speed-up in parallel software by various benchmarks. The speed-up represents practical effects of increasing the number of cores in an application capable of parallel processing. Since the P-value reflects the "*usability*" of multiple cores, the level of speed-up determines how many of the available cores can be efficiently used. In more specific domains, tools such as Cilkview [25] can be used to predict the speed-up in parallel Cilk applications. The tool calculates the critical path in the software as the upper bound limit, and predicts a speed-up range for an application as a function of the number of cores as illustrated in Figure 7.4. The P-value is then extracted from the output as either an optimistic or pessimistic value.

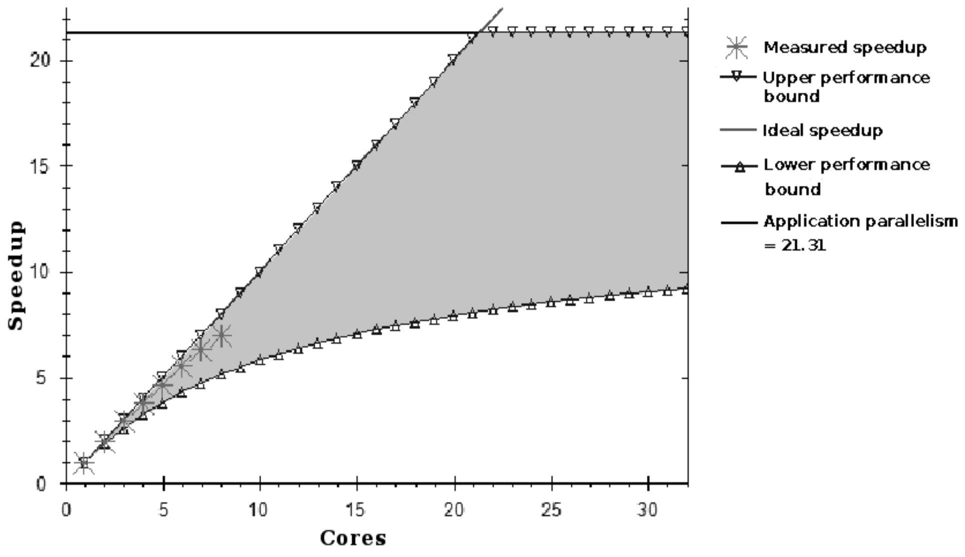


Figure 7.4: A Cilkview example showing the scalability approximation and measured speed-up for a Cilk-based quick-sort algorithm [25].

In the dataflow domain, dataflow tools can be exploited to predict the P-value in common dataflow networks. The following section presents an approach to practically highlight this property of dataflow programming.

Dataflow tools to the rescue!

Dataflow programming originally presented in [17] is a proposed solution to visualize computations as a set of independent nodes called actors. The actors are driven by data queues connected as in- and output ports to the actors called edges. Data flowing between actors is described as a quantized object called tokens. Each actor in a dataflow program can execute independently as long as data is available on the input ports; this construction explicitly exposes the potential parallelism in the program.

A commonly used dataflow construction is the Synchronous Data Flow (SDF) graph [19] in which each actor has a fixed data rate on the input and output port. SDF offers therefore strong compile time predictability, and transformation models can be used on the base model to allow application optimizations. One example is the Single Rate SDF (srSDF) transform which transforms the actor edges into homogeneous edges (data production rate = data consumption rate). The transformation is illustrated in Figure

7.5 in which the left part using different input and output rates is transformed into a network with only single data rate ports between the actors.

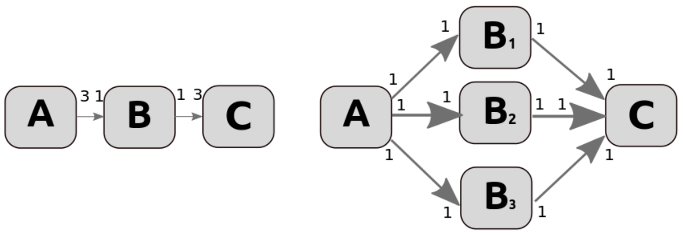


Figure 7.5: A SDF graph and its srSDF transformation – a multi-rate link is transformed into several single-rate links to enable parallelism.

This transformation exposes the parallelism more explicitly since the B-actors in Figure 7.5 can be independently scheduled on separate processing units. The execution is clearly defined as completely sequential in the A- and C-phases and parallel during the B-phase. The P-value is hence determined based on the number of independently executing actors and based on the available hardware.

P-value extraction using PREESM

A practical case study was demonstrated in Paper VI [33], in which the PREESM tool [61] was used to extract the P-value from parallel applications. PREESM is an opensource tool for dataflow network construction and rapid prototyping. The tool was developed at INSA de Rennes, France and is capable of automatic srSDF transform and c-code generation. Based on explicit architecture settings and scenario parameters, PREESM can generate applications tailored for specific hardware architecture without modifying the software functionality.

The tool was used in a case study for parallelizing and extracting the P-value of an image processing application. After parallelizing and optimizing the SDF graph, PREESM automatically generates a Gantt chart which illustrates the execution on the parallel hardware. Three different versions of an image processing application was generated in the case study: one **completely sequential**, one **completely parallel** and one **mixed-parallel**. After the P-value was extracted, the value was injected into the application phases accordingly.

Figure 7.6 illustrates the three cases. The first case is a sequential application with only one actor A active for the complete execution. The second case illustrates a dividable actor A_n which is able to execute completely in parallel. Lastly Figure 7.6 illustrates an application with a non-dividable

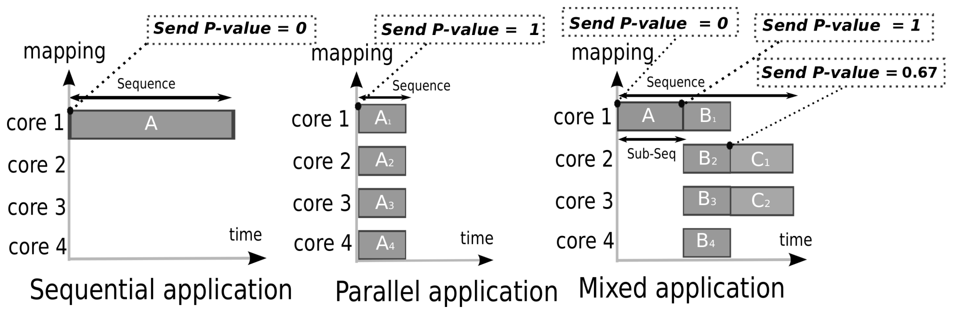


Figure 7.6: Declaring the P-value for each program phase in three examples.

actor A followed by a parallel actor B_n and finally a semi-parallel actor C_n.

A P-value equals 0.0 was extracted from the completely sequential version which indicates that adding additional cores will not affect its performance. P-value equals 1.0 was extracted from the parallel version which indicates that the performance will linearly improve by increasing the number of cores (for a quad-core system in this case). Finally the mixed-parallel version implemented both sequential, parallel and semi-parallel phases. A P-value representing the currently available parallelism was extracted from each phase according to Equation 7.5:

$$P = \left(\frac{\frac{1}{S} - 1}{\frac{1}{N} - 1} \right) \tag{7.5}$$

where S is the speed-up factor between the sequential and the parallel actor composition and N is the number of processing units.

7.3 Summary

Bricktop uses a model-based approach to determine the resource allocation based on the performance requirements and measurements from the applications. The models upon which decisions are made should therefore represent the real outcome of the resource allocation. Two models are considered: *Power* and *Performance*, according to which actuation decisions are made. The first model predicts the increase or decrease in power as the system allocates or de-allocates a selected bundle of resources, and the second model predicts the speed-up of the application as resources are allocated or de-allocated. With the provided system information, an optimization solver was used to determine the most efficient resource allocation or de-allocation for a given set of applications. This is presented in the following chapter.

Chapter 8

Multi-Criteria Power Optimization

"*He who controls the spice controls the universe.*"
— Baron Vladimir Harkonnen, Dune - 1984

With both models as input, *Bricktop* selects the power optimal combination of DVFS and DPM for the given set of applications. The selection is based on an optimization algorithm which, periodically and with low overhead, determines what actuator combination results in the lowest power dissipation while no QoS requirements in the applications are violated.

8.1 Related Work

Feedback based control is a many decades old approach to regulate systems based on a setpoint and measurement input. Such systems have been suggested in many papers such as [3, 18, 46, 47, 67] in which one-variable control systems are used to minimize power in multi-node computing systems.

The Napsac system [46] focus on a web cluster manager which, based on the request rate, allocates or de-allocates servers using sleep states. The algorithm uses the notion of *workload* to start up and shut down servers while also accounting for the latency of each action. Further, in the work by Leverich et al. [47] a control system for power gating individual cores in multi-core systems is suggested. The controller uses a high-low watermark based algorithm measuring the workload to wake up or shut down a core. Varma et al. [84] used a PID controller to scale the clock frequency according to the workload. The controller was capable of predicting future load patterns and proactively scale according to the prediction. Similarly to Paper I and II in this thesis, only a single output variable was used, which means that the controller can use a single transfer function, or simple if-then-else statements without multi-criteria optimization.

Control mechanisms for a multi-variable output systems were suggested in [8, 23]. Both works used DVFS and DPM control based on one or more input metrics. More specifically, the work by Ghasemazar et al. [23] selected the optimal number of active cores for a given throughput, after which the system fine-tuned the clock frequency with a feedback loop. Similarly, the controller approach presented by Bhatti et al. [8] chose, in sequential order, whether to optimize for DPM or DVFS. Our approach differs from the mentioned works because the *Bricktop* controller selects the optimal combination of DVFS and DPM rather than optimizing both methods one after the other. The problem is hence formulated as a *multiple-input multiple-output* problem rather than two *multiple-input single-output* in sequence.

Liu et al. [49] use multi-variable optimization methods to either minimize the total energy in a system or its maximum temperature. Side constraints included feasible task execution under timing constraints, respecting the thermal thresholds and keeping the core voltage in the allowed range. With a multi-variable problem, a multi-variable optimization approach was also considered in the work by Parolini et al. [60]. The work demonstrated an approach to optimize the power dissipation of a server system by DVFS and by regulating the CRAC units (air-cooling units) to cool the servers. Several side constraints were included such as thermal limits and application QoS based on the throughput of the total server system.

Complementary details have been used from the mentioned works to describe the optimization problem of using DVFS and DPM based on application performance. Such problems are either described as linear or non-linear depending on the setup of the problem, and the following section will present the setup in more detail.

8.2 NLP-Optimization

A Non-Linear Programming (NLP) programming problems are characterized as problems in which either the objective function or any of the side constraints contain variables of a higher degree. Both of these conditions are satisfied in our system with the third degree polynomial (in Equation 7.1) and the function of Amdahl's law (in Equation 7.3). Optimizing the DVFS and DPM magnitude is therefore a multi-dimensional problem with two control variables.

Figure 8.1 illustrates a common approach in solving multi-dimensional NLP problems. The search function is initiated at a starting point after which a search direction and a step length is defined for finding the next evaluation point. After performing a step, the new point is valid IFF *the* *value of the objective function is lower than in the previous step.* The result of the optimization method is to find the global optimum at which no new iteration will give the objective function a smaller value (illustrated with the red arrow in Figure 8.1). Several methods for finding the optimum exist such as: Newton, General Reduced Gradient (GRG) and Sequential Quadratic Programming (SQP). The common goal for each method is, however, to determine the search direction and step length in order to find the optimum with a low number of iterations and with good accuracy.

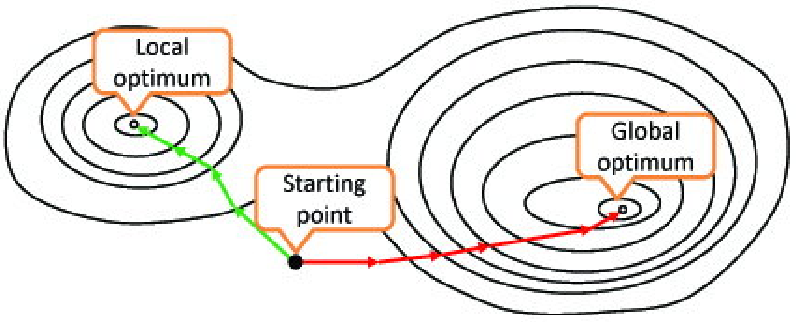


Figure 8.1: Illustration of solving a multi-dimensional NLP problem.

While linear optimization is more predictable and can guarantee a global maximum, NLP problems raise more concerns. Issues with NLP problems are firstly the inability to ensure global optimum. The optimization method cannot guarantee the avoidance of converging towards a local optimum. A local optimum is a point at which a step in either direction results in a higher objective function value, but an even lower value point exists in the global search space. This is illustrated as the green arrows in Figure 8.1. Converging towards the local optimum is a result of, for example, the choice in starting conditions. Secondly, NLP problems have a high complexity with respect to the control variables. Fortunately the optimization is iterated frequently, and a guaranteed global optimum is not a definite requirement for each iteration as long as the solution is *sufficiently* close to the optimum. As previously explained, only two control variables are used, which means a manageable complexity in practice.

8.2.1 Identification of optimization problem

The objective of the solver is to determine an actuator configuration which minimizes the power while still is providing sufficient resources to all applications. The impact of actuator magnitude on the power dissipation is derived from the previously defined power model (Section 7.1). Orthogonally, the impact on performance based on the actuator magnitude is derived from the performance model presented in Section 7.2. By combining both models, the minimum power dissipation for executing the applications can be determined.

As both the power model and Amdahl's law used in the performance model are clearly convex nonlinear, the problem is set up as a non-linear optimization problem. Recalled from Chapter 6, the required resources are given as a setpoint S, while the actual performance P_f is monitored and transmitted to the power manager in which the error value E is calculated. The power optimization problem is hence defined as follows:

$$\begin{aligned}\text{Minimize}\{\text{Power}(q,c)\}\\ &\text{Subject to:}\\ &\forall n \in \text{Applications}: E_n - (q+c) < S_n - Q_n\end{aligned} \tag{8.1}$$

where the variables: q and c are the actuators (DVFS and DPM respectively). "Power" is a given power model for the system in question (Section 7.1). S_n is the setpoint, E_n is the error value and Q_n[1] is the lower QoS limit. The optimization rule states to *minimize power while eliminating enough errors to reach at least the lower bound QoS limit.* This is achieved by setting the actuators (q,c) to a level sufficiently high for each application n.

- The setpoint S is set by the user to represent a practical performance aspect of the application which should be reached, for example "25" as frames per second in a video decoder.
- E is measured by the application – this is the current (real) performance.
- The QoS limit Q can be set by the user or obtain a default value for example 95%. This means that a 5% deviation from the setpoint would be treated as acceptable.

Initial evaluations of the optimization solver was performed in the Matlab environment using the `fmincon` implementation. The actuator response was visualized according to a set of errors E the P-value set to $P = 0.9$ (a semi-parallel application). The errors were steadily increasing from a starting point as a linear function, which replicates a system with continuously increasing load. The power model presented in Chapter 7 (Figure 7.1) was used for the experiments.

Figure 8.2 shows the actuator response for the mentioned use-case. As seen in the figure, DPM is chosen (cores are added) as the error value is very low. When DPM is roughly at 45%, the parallel proportion of only 0.9 causes the system to not benefit enough from activating more cores. DVFS is

[1] E and Q are normalized to the range in which q and c operate

instead used while the numbers of cores decrease. As the error rate hits 5.5, DPM is again increasing its utilization since the very high clock frequencies increases the power rapidly (compare to Figure 7.2), and the highest DVFS steps are used as the last resort since an application with $P = 0.9$ scales only to roughly 2 out of 4 cores.

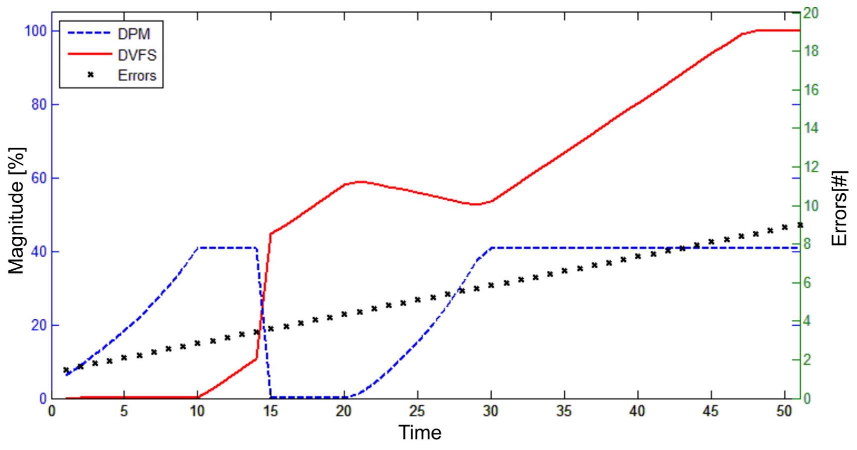


Figure 8.2: Actuator response from steadily increasing QoS errors (Plain SQP method).

As seen in Equation 8.1, only two two control variables: DVFS (q) and DPM (c) are used, which means that the complexity for solving the optimization problem is low. However, the response time of reaching a viable solution is crucial to the usability of the solver since the algorithm is being executed on a regular basis. Furthermore, the energy consumed by the solver alone must not exceed the energy savings obtained from optimizing the actuator usage.

8.2.2 Response time evaluation

In order to determine the response time and energy overhead of regularly solving non-linear optimization problems, a set of optimization algorithms were evaluated. The evaluation of the optimization algorithms was set-up in the MATLAB environment using the `fmincon` non-linear optimization solver. Our chosen baseline method implemented the SQP [24] algorithm with the plain objective function and side constraints given in Equation 8.1.

The baseline SQP was compared to a set of further optimized solvers:

1. SQP [24] with Gradient
2. Interior Point [43] with Gradient
3. Interior Point [43] with Gradient and Hessian

While increasing the computational complexity of the solver increases the number of instructions required in the algorithm, the solver might find a viable solution faster than the baseline solver and thus decrease the response time. One option is to provide the user defined gradient of the objective function to the solver. The gradient function approximates the search direction with a first order system, which can result in fewer optimization steps and a faster solution.

The gradient function was defined as $g = \frac{\partial f}{\partial A} = \begin{bmatrix} \frac{\partial f}{\partial q} \\ \frac{\partial f}{\partial c} \end{bmatrix}$ where f is the objective function and its derivative is defined for each actuator (q and c). By inserting the values from the objective function given in Equation 8.1, the derivatives are defined as:

$$\frac{\partial f}{\partial A} = \begin{bmatrix} p_{01} + p11q + p_{21}q^2 \\ p_{10} + 2p_{20}q + p11c + 3p30q^2 + 2p21cq \end{bmatrix} \tag{8.2}$$

The the analytical partial derivatives of the side constraints are defined by the partial derivative of all side constraints C with respect to the actuators c and q. $C = \begin{bmatrix} \frac{\partial C}{\partial c, \partial q} \end{bmatrix}$ where $\frac{\partial C}{\partial q, \partial c}$ are the first order derivative of actuators with respect to the side constraints.

Secondly, Interior point based methods were selected for evaluation. Interior point based methods can approximate the objective function both as a first order system and as a quadratic function with a Taylor-series expansion. The direction of the search function in the latter case is called the Hessian matrix H, which approximates the search function as a second order system as: $H = \frac{\partial^2 f}{\partial A^2} = \begin{bmatrix} \frac{\partial^2 f}{\partial c^2} & \frac{\partial^2 f}{\partial c, \partial q} \\ \frac{\partial^2 f}{\partial c, \partial q} & \frac{\partial^2 f}{\partial q^2} \end{bmatrix}$ where A is the actuators. The matrix contains the second order partial derivatives of the objective function defined in Equation 8.1. Insertion of the parameters gives the Hessian matrix:

$$\begin{bmatrix} 0 & (p_{01} + p_{11}q + p21q^2)(p_{10} + 2p_{20}q + p_{11}c + 3p_{30}q^2 + 2p_{21}cq) \\ (p_{01} + p_{11}q + p21q^2)(p_{10} + 2p_{20}q + p_{11}c + 3p_{30}q^2 + 2p_{21}cq) & 2p_{20} + 6p_{30}q + 2p_{21}c \end{bmatrix} \tag{8.3}$$

Evaluating the Hessian matrix further increases the computational complexity, but also might result in fewer iterations to find optimum.

The execution time for finding 70 solutions for all algorithm configurations was measured in the Matlab environment. Note that the execution times do not reflect the overhead of a real-world C or Python implementation, but merely displays the relation between algorithms. Table 8.1 shows the results. The plain algorithm used only the cost function and the side constraints given in Equation 8.1, while the other included the user defined gradient or Hessian. The SQP with Gradient input had the shortest execution time, and the Interior Point with Hessian input was clearly the most expensive algorithm. The SQP algorithm with the gradient function was therefore chosen as algorithm in our solver.

Table 8.1: Average execution times for 70 solutions.

Plain SQP	**SQP Gradient**	**Interior Point Gradient**	**Interior Point Hessian**
16.44 ms	13.97 ms	29.88 ms	287.27 ms

8.3 Simulation Based Evaluations

The complete system was simulated with a multi-core TrueTime [9] based simulator, which allows individual task scheduling per core, full DVFS support and per-core DPM. It is also a practical tool since it runs on top of Simulink and is capable of using the complete toolbox of Matlab. The purpose of the simulation was to determine **1)** how the parallelism in an application affects the energy **2)** the energy consumption of an optimization based mapping and the default Linux mapping. The power model for the quad-core ARM described in Section 7.1 was used as our general reference model for constructing the simulation environment, and the simulations were run for 200 seconds each.

8.3.1 Actuator response

The power manager was evaluated by executing a simulated video decoder. A simulated high definition (HD) video was processed with one type of frame with a static size. Video decoders with different levels of parallelism was simulated in order to study the actuator utilization. For illustrative reasons, the actuator response for a single use cases was plotted.

Figure 8.3 shows the corresponding actuator response for the video decoder. With a P-value of only 0.7 the system is able to only utilize DPM to 25%, and is unable to process the video even when executing at the highest clock frequency. The drop in QoS for the video decoder is listed in Table 8.2. The more parallel version ($P = 0.9$) is, on the other hand, able to use more cores to process the video without significant QoS degradation (as seen in the fifth column in Table 8.2), but is still forced to push DVFS to roughly 85% of the maximum clock frequency.

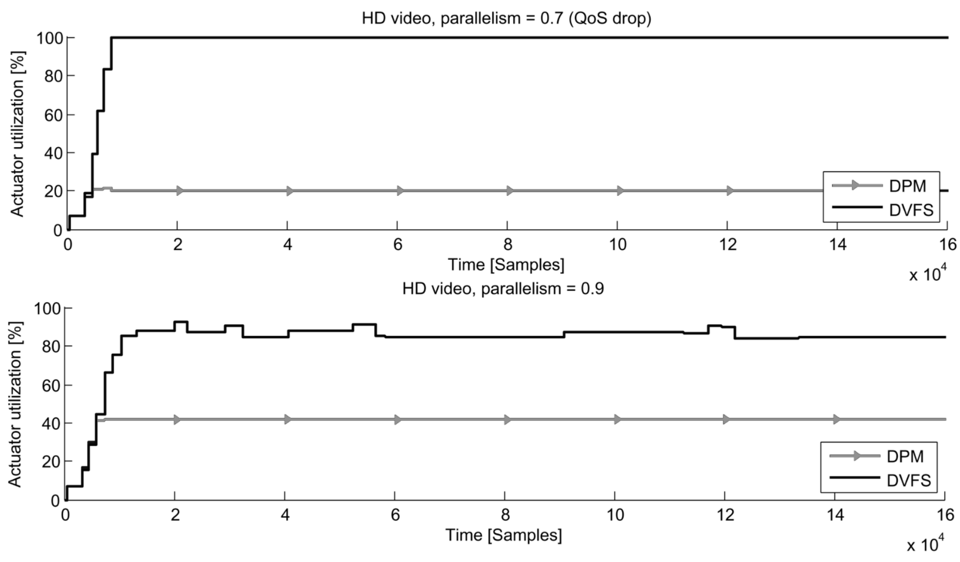


Figure 8.3: Actuator response for video decoder with two different P-values.

8.3.2 Energy consumption

The energy consumption of the decoder was evaluated by simulating implementations with nine different levels of parallelism in the range
[0.5 0.7 0.8 0.85 0.9 0.91 0.92 0.93 0.94]. This P-value was static for the whole experiment run. A SD (standard definition) and the HD (high definition) video was used in order to evaluate the response of using different resource requirements. Both cases was compared to the actuation policy in the default Linux Completely Fair Scheduler (CFS) [41], using the *Ondemand* (OD) frequency governor [79]. This setup is shipped as the default configuration in typical Linux machines. The behavior of the CFS+OD was replicated in the TrueTime environment with the policies:

1. Applications are scheduled on all cores (as far as the application scales)
2. Cores with no tasks are activated but idle
3. DVFS utilization is set by the OD governor with the following policies:
   (a) Clock freq. is set to the lowest possible according to the workload
   (b) The workload is too high if an `upthreshold` limit[2] is reached. Then, the frequency is increased to the maximum and step-wise decreased to the lowest feasible setting

The energy consumption for each simulation is shown in Figure 8.4. In the HD case, the energy consumption is highest around $P = 0.8$ and de-

[2]The upthreshold in Linux is usually set based on best practice for the system in question. Typical settings are around 80-95% of full workload (100%)

creases in both directions. This is, firstly, because a lower parallelism prohibits the system from using a sufficient amount of cores and idle resources are shut down to save power. The cost of disabling resources is a degradation in the QoS of the video playback as seen in Table 8.2, which might or might not be acceptable to the user.

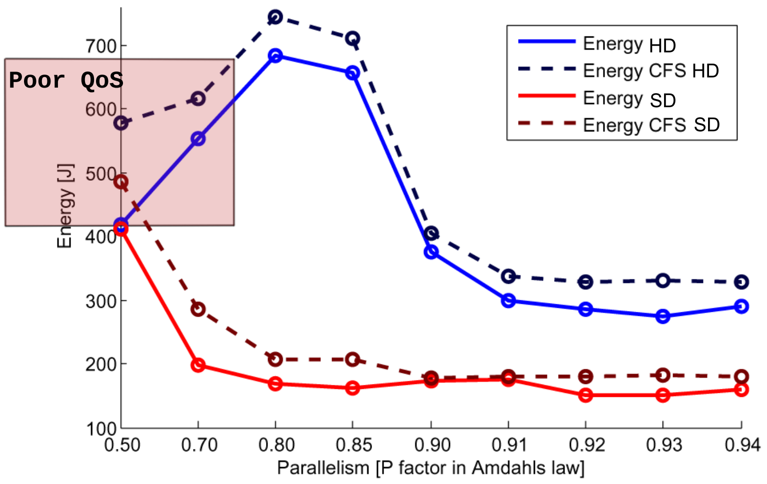


Figure 8.4: Energy consumption for HD and SD video compared with standard Linux CFS+Ondemand (Rings are data).

Secondly, increasing the parallelism allows the system to activate more cores and hence reduce the clock frequency. The static power P_s increased by activating the cores is significantly lower than the dynamic power P_d saved when decreasing the clock frequency, which results in energy savings. In the SD case, the resource requirements are much lower and a decreased clock frequency can occur already at $P = 0.7$ as seen in Figure 8.4, whereas the HD case requires a parallelism of at least $P = 0.9$.

However, the energy consumption reaches, in both cases, an energy plateau at certain points (roughly at $P = 0.91$ for the HD case and $P = 0.7$ for the SD case). At this point the parallelism of the application is not strongly worth improving – from an energy point of view – since mapping the application onto even more cores will only result in a static power increase approximately equal to the related dynamic power decrease i.e. $\Delta P_s \approx \Delta P_d$.

The optimized cased showed overall lower energy consumption than the default CFS+Ondemand case, which is due to two reasons:

1) Low scalability forces mapping onto only a few cores. For the CFS+Ondemand case no cores can be shut down and they dissipate waste power while idling.

2) With very high scalability the applications are scheduled on *too many* cores, which leads to an increase in P_s which is larger than the total P_d

decrease when lowering the clock frequency to reach the same performance i.e. $\Delta P_s > \Delta P_d$. In other words, the static power becomes more significant than the dynamic power.

Table 8.2: QoS (in %) for HD and SD case compared with the standard Linux CFS+Ondemand policy.

P-value	0.5	0.7	0.8	0.85	0.9	0.91	0.92	0.93	0.94
QoS HD	6.7	34.8	92.7	97.3	99.4	97.8	97.8	96.7	90.1
QoS-CFS HS	5.0	20.3	92.9	97.6	99.1	99.1	96.2	96.7	89.7
QoS SD	73.7	95.9	93.9	92.6	92.3	92.4	95.8	96.5	95.5
QoS-CFS SD	77.5	96.1	90.6	88.2	93.4	95.1	98.7	97.5	92.4

8.3.3 Discussion

To minimize the energy consumption two main parts should be optimized: **1)** Application execution **2)** Application mapping. An application should execute with a performance level such that no unnecessary resources are allocated. This means that the resource allocation should be regulated such that the minimum allowed performance level is achieved in the application. Secondly, the application should be mapped on the optimal number of cores based on the application parallelism and the ratio between static and dynamic power dissipation for the hardware architecture in question. *Unnecessarily high dynamic power is dissipated when the application is mapped on too few cores and an unnecessarily high static power is dissipated when the application is mapped on too many cores.* By optimizing both parameters, an energy optimal execution and mapping is achieved.

8.4 Switching Latency in Power Management

The current theoretical case implies an immediate actuation response and the resources are available to the applications without any allocation overhead. Practically, computer systems include physical limitations because of both hardware and software actions as resources are allocated. Switching latency in power management mechanisms represents the overhead between resource request and resource availability. With a significantly large latency, the performance monitoring used as basis for the power management becomes more frequent than the hardware resource allocation. This can cause inefficient resource management since the timing between demand and allocation is out of phase i.e. *resources are allocated when not needed and de-allocated when needed.* In the most extreme case, switching latency leads to controller instability which causes the resource allocation to oscillate between minimum and maximum uncontrolled.

The following sections describe latency measurements for DVFS and DPM using different configuration parameters. The obtained information is used to define the limits in which the controller is capable of operating, and further studies can exploit the information to create a predictive latency model in order to fine tune the controller.

8.4.1 Latency Measurements

DVFS and DPM contain software drivers used to manipulate the hardware resources via operating system calls. A generic latency is hence not expected because influential parameters include both software implementations, OS versions, driver versions and the physical hardware. However, the software-to-hardware call stack remains static for Linux based systems as:

- User or kernel requests an update to either DVFS or DPM.
- A set of OS calls are made to the kernel.
- The kernel requests hardware access with platform specific drivers.

After the procedure is completed, the hardware resources has been allocated. Since the main focus was set on mobile devices, a thorough investigation was conducted on the very procedure of DVFS and DPM on ARM devices. The latency of allocating resources was measured as the *elapsed time between requesting a resource (DVFS or DPM) and the return of the call which completes the process.*

The evaluations consists of two separate implementations: **1)** kernel space and **2)** user space implementations. The kernel space implementation accesses the functionalities either from a kernel module or via direct system calls. The user space implementation uses the `sysfs` interface, which is read by the kernel using filesystem I/O operations.

DVFS In a Linux based system the following core procedure describes how the clock frequency is scaled:

1. A change in frequency is requested by the user
2. A mutex is taken to prevent other threads from changing the frequency
3. Platform-specific routines are called from the generic interface
4. The PLL is switched out to a temporary MPLL source
5. A safe voltage level for the new clock frequency is selected
6. New values for clock divider and PLL are written to registers
7. The mutex is given back and the system returns to normal operation

The kernel space implementation issued direct system calls to the kernel to indicate a change in frequency and the user space implementation wrote the setpoint frequency value in a `sysfs` file. Since the resource access is, as previously mentioned, dependent on software mechanisms, the system was stressed to different load levels in order to demonstrate the effects of load present. The system was stressed with a step-wise increasing load using Spurg-Bench[3], and the latency was measured 100 times per load level. Figure 8.5 shows the average latency for an Exynos 4412 ARM chip under different levels of workload. When using the system call interface, the average latency decreases slightly when increasing the load (left part of Figure 8.5). On the other hand, the switching latency has a strong correlation to current clock frequency and target clock frequency in the user space implementation. As expected, the latency is shorter as the clock frequency transitions from 1600 to 200 MHz because most of the DVFS procedure (including the file system call) is executed at the higher frequency.

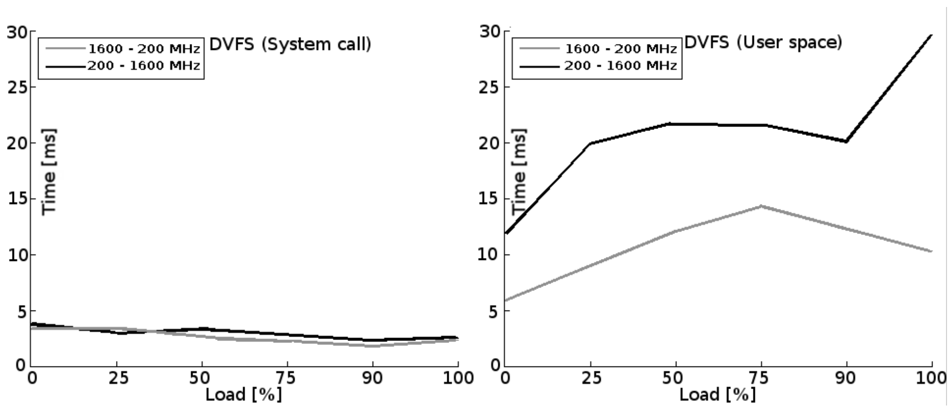


Figure 8.5: Average latency for changing clock frequency under different load conditions using system call and `sysfs` interface.

DPM As recalled from Section 3.3.2 in Chapter 3, the shutdown of a core is reliant on callback functionalities in the Linux kernel, which means that the system performance and current utilization will affect the response time of the kernel thread. The wake-up procedure is, similarly to the shut-down procedure, dependent on callbacks but with an inter-core interrupt to trigger the core startup. Both functionalities were, however, evaluated since procedure details still differ [56], for example the expensive procedure of creating kernel threads. Similarly to the DVFS measurements, DPM using the CPU hotplug implementation in Linux was evaluated on the same Exynos 4412 chip. The kernel space implementation consisted of a kernel module with direct access to kernel functions and the user space implementation consisted of filesystem I/O to the `sysfs` interface.

The average latency for shutting down the Exynos4 core in kernel- and user space respectively is shown in Figure 8.6. It is clear that the average

[3]https://github.com/ESLab/spurg-bench

latency for shutting down this type of core is rather constant and not significantly dependent on clock frequency as long as the implementation resides in kernel space. On the other hand, the user space implementation is more dependent on the load level of the system as the latency tend to double between 0% load and 100% load. The user space implementation is also more influenced by the clock frequency of the CPU.

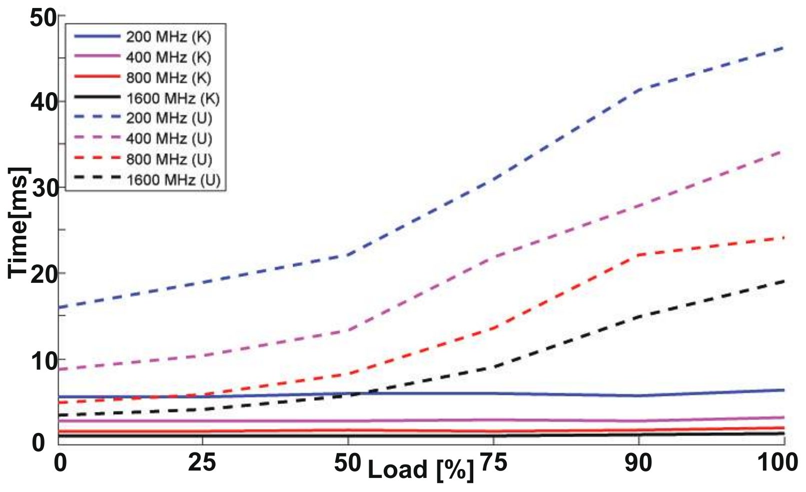


Figure 8.6: Average latency for shutting down a core under different load conditions using kernel and userspace mechanisms.

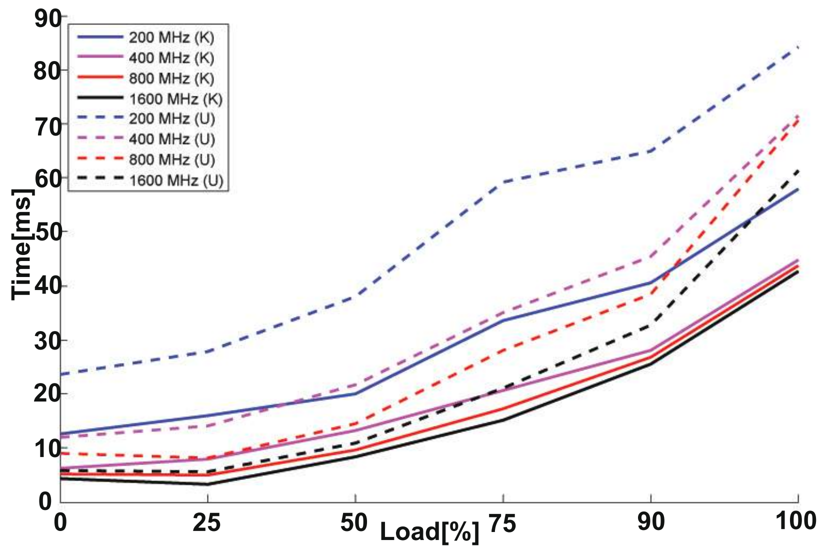


Figure 8.7: Average latency for waking up a core under different load conditions using kernel and userspace mechanisms.

On the contrary, the wake up time is more dependent on the load level in both the kernel space and the user space implementation. As seen in Figure 8.7, the kernel space implementation measures up to 6x higher latency for the 100% load case in comparison to the 0% load case. A similar ratio is seen in the user space implementation, but the latency is on average roughly 2x higher than the kernel space implementation. In summary, the latency for utilizing power management on modern microprocessors depend on several parameters such as clock frequency, workload, software implementations and lastly the hardware platform. Moreover, a latency in the tens of milliseconds range can be expected when using current implementation under normal system conditions. For managing the power on modern many-core CPUs, the power manager should either be implemented with low latency in mind on OS and hardware level, or the latency based on clock frequency and workload should be added as a parameter to the power manager.

8.5 Summary

A control-theoretical approach to minimize the CPU energy consumption has been presented in this chapter. The energy optimization is based on an architectural model describing the power dissipation of the microprocessor in question, and a performance model is used for describing application speed-up with relation to its parallelism. A set of optimization methods have been evaluated based on response time and simulations have demonstrated the feasibility of using the optimization based approach compared to the standard Linux CFS+Ondemand approach.

The latency of the actuators has finally been measured on mobile devices in order to improve the efficiency of the real-world implementation. Within the limits of this work, this information is used to determine the period of the update frequency used in the power manager. Future work remains to investigate whether it is possible to more accurately determine the optimal DVFS/DPM combination based on the difference in the latencies. Chapter 9 presents practical details regarding the mapping to real hardware, and a set of case study application executing under the Linux OS.

Chapter 9

Case Studies

"Welcome to the desert of the real."
— Morpheus, The Matrix - 1999

This chapter presents the real-world implementation of the *Bricktop* power manager and a set of applications used for evaluation. Applying the presented methods to already existing software and utilizing *Bricktop* as a power manager is applicable for the following systems:

- Software with declared QoS and measurable performance
- Software with measurable or approximated level of parallelism
- Implementation currently for C-based software, but core algorithm agnostic to the programming language
- Multi-core platforms with DVFS and/or DPM supported by a Linux distribution

This chapter shows experimental results from implementations executed on a quad-core Exynos 5410 ARM platform with the Ondemand frequency governor as reference point, and the *Bricktop* power manager used for comparison. The output of the experiments is the consumed energy for both systems with a constant pre-defined performance level.

Bricktop is mapped on top of the operating system and is available as a middleware to the applications as illustrated in Figure 9.1. Applications connected to *Bricktop* issue library calls for transmitting configuration and measurement parameters to *Bricktop* as shown in Listing 9.1:

```
#include <fmonitor>
.
fmonitor(<performance>,<P−value>);
.
```

Listing 9.1: Library call for transmitting performance values to *Bricktop*.

Applications can naturally use the operating system freely, and *Bricktop* can be bypassed completely if no performance requirements are needed in the application.

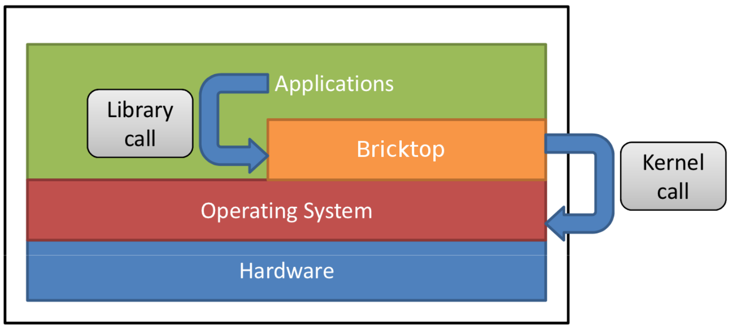


Figure 9.1: *Bricktop* used as a middleware on top of the OS.

The communication was established by using System V queues to push and pull data between *Bricktop* and the applications as illustrated in Figure 9.2. Applications push a data packet containing a header with application ID and priority followed by the performance and P-value. The message queue is polled by *Bricktop* regularly and all packets are pulled from the queue each period. The current infrastructure supports multiple applications, but is limited to a fixed priority between applications. Future research is thus needed for supporting multiple priority levels.

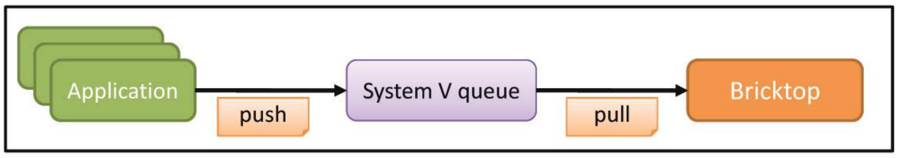


Figure 9.2: Applications are communicating with *Bricktop* using shared memory queues.

The period of the *Bricktop* controller should hence be set long enough to not suffer from the hardware/software latency, but also short enough to give an as fast as possible response. Since the SQP solver is the most computationally heavy part of the power manager we measured the elapsed time for obtaining one solution. This time was measured to 900 μs. Further based on results from [71] and [42, 59], the access time of DVFS and DPM can significantly impact on the chosen period of the power manager. Because the measurements by Schöne et al. [71] were conducted on Intel platforms, independent experiments were conducted on the chosen ARM platform (presented in previous chapter). A period of 60ms was finally chosen, which means that regulating the actuators will most likely be completed before the next period is reached.

The following sections shortly presents three use-cases: *MPlayer*, *LDPC-decoder* and *Facedetection* in which the energy consumption is measured for using the Ondemand frequency governor and *Bricktop*.

9.0.1 MPlayer

MPlayer is a free software and open source media player. The program is available for all major operating systems, including Linux and other Unix-like systems, Microsoft Windows and Mac OS X. MPlayer can play a wide variety of media formats and can also save all streamed content. Video decoding is executing in one or more threads controlled by the MPlayer process. As in any video decoder, the MPlayer threads requires varying CPU resources depending on the current frame type and its content under execution.

The selected videos was of type h.264 using resolutions 480p 720p and 1080p. Since the chosen platform was a quad-core system, the experiments were executed using four decoder threads.

Performance metric The selected performance metric for the MPlayer application was the framerate measured in "Frames per Second" or *fps*. MPlayer was modified such that the decoder outputs the framerate directly to *Bricktop*. Heavy variations and occurrence of I-frames requires significantly more hardware resources than frames with low variations. The requirement of MPlayer is to execute the decoder fast enough such that the output buffer never empties.

MPlayer was implementend using standard pthreads, and experiments using on four threads resulted in a scalability with the **P-value** of 0.96. The video output task reads the output buffer with a period of 25 fps, which means that the decoder filling the buffer must on average operate on a slightly higher framerate (for example 30 fps) in order to cope with occasional framerate under-runs. Figure 9.3 shows the results from the experiments on the ARM device using the Ondemand governor to the left and *Bricktop* to the right. Table 9.1 shows the energy consumption for each experiment.

Table 9.1: Energy consumption (in Joules).

Video resolution [size]	**1080p**	**720p**	**480p**
Ondemand	279.96	103.96	70.48
Bricktop	158.62	43.88	35.84

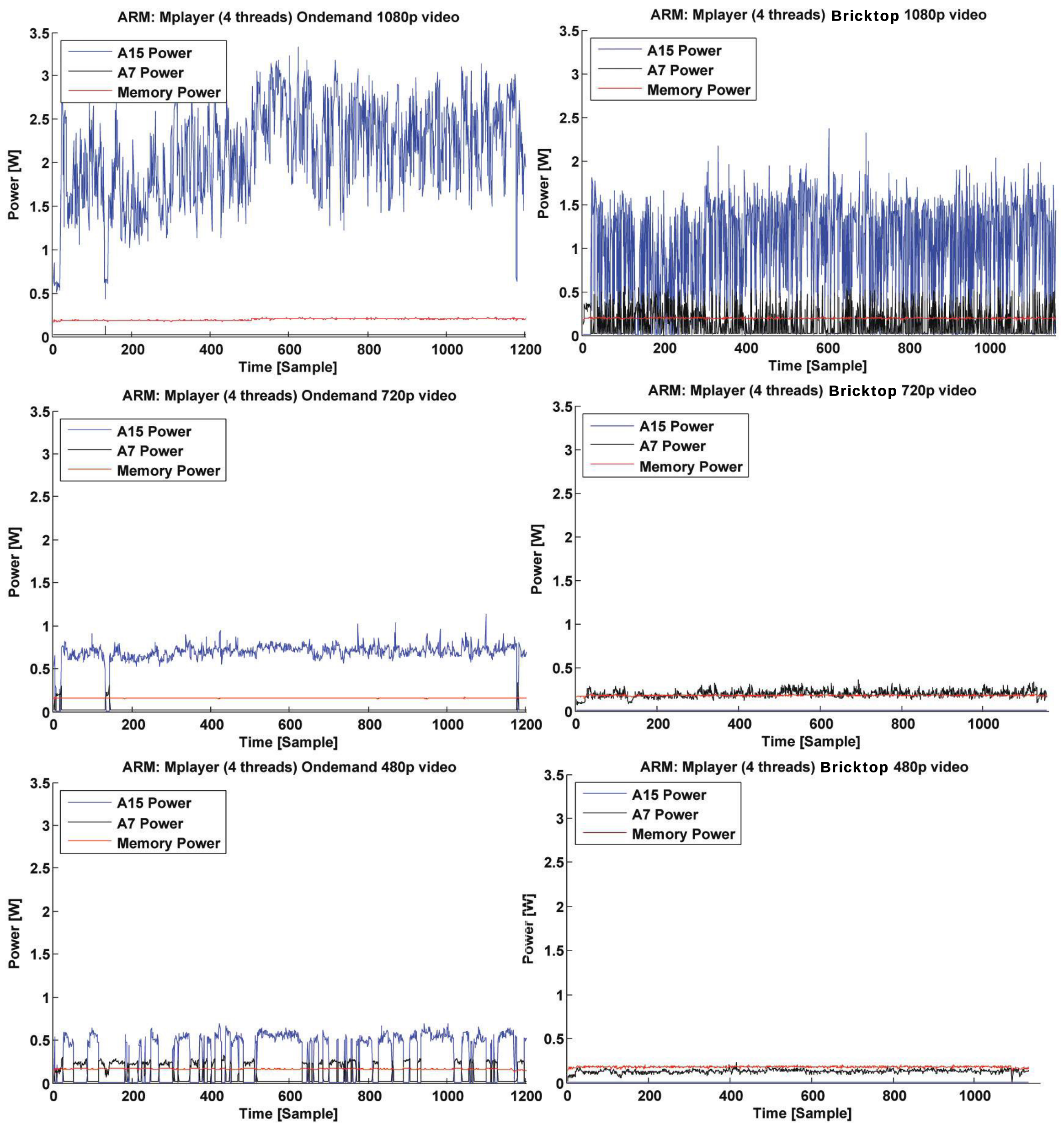


Figure 9.3: Power dissipation from MPlayer experiment

9.0.2 LDPC Decoder

LDPC (low-density parity-check) codes are a class of error correction codes used in various telecommunications standards in order to correct data corruption caused by noisy transmission links. An LDPC code is represented by a bipartite graph called a Tanner graph, which describes the relation between information bits and parity bits in a transmitted block of data called a codeword. Decoding LDPC codes is often performed using iterative message passing algorithms which consist of two major steps, the bit-node update, and the check-node update. These two steps are iterated back to back as many times as is required to successfully recover the original data, or until a certain maximum number of iterations have been performed.

Performance metric The performance metric from the LDPC decoder is represented as the throughput given in “Megabits per second” (or Mbit/s). The benchmarks were conducted with six discrete bitrate setpoints for the ARM device, and the input was distorted with white noise. The decoder was implemented using pthreads and handcrafted parts to utilize the neon engines as fully as possible on the ARM platform. The **P-value** was approximated closely to 1.0 for a quad-core system. Figure 9.4 shows the results from the experiments on the ARM device using the Ondemand governor and *Bricktop* for bitrates in range [1.25 - 7.5] Mbit/s. Table 9.2 shows the energy consumption for each experiment.

Table 9.2: Energy consumption (in Joules).

Throughput [Mbit/s]	**7.5**	**5.0**	**2.5**
Ondemand	315.64	127.15	39.70
Bricktop	193.74	78.22	22.25

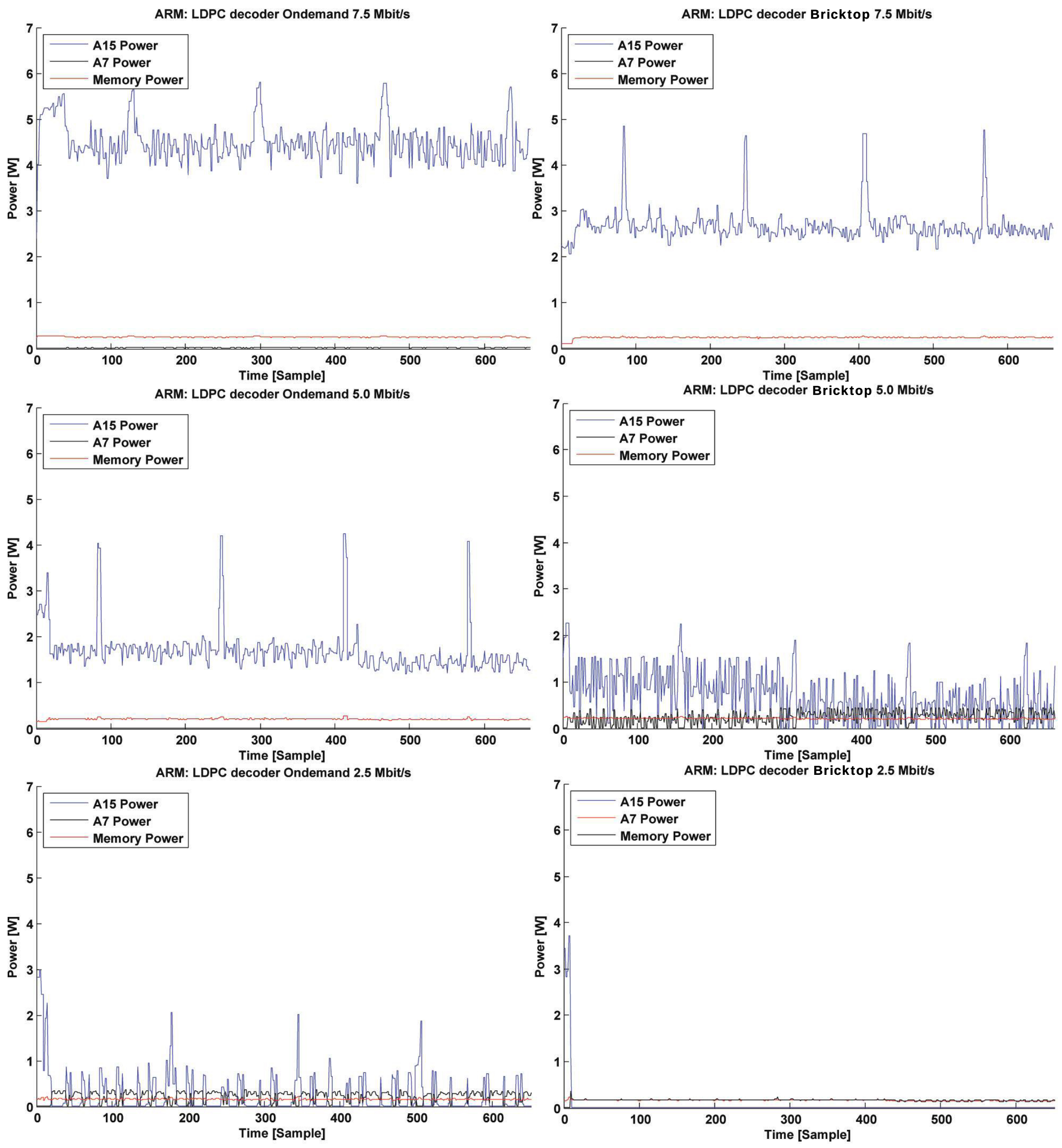


Figure 9.4: Power dissipation from LDPC decoder experiment

9.0.3 Face Detection Software

The benchmark consisted of a software capable of extracting the number and location of human faces from an image. The image can either be static or be read as input from for example a webcam.

The analysis starts by selecting the regions of interest in the image. A set of skin-like regions, which are considered face candidates, are extracted from the video frames. After orientation normalization and based on verifying a set of criteria (face symmetry, presence of some facial features, variance of pixel intensities and connected component arrangement), only facial regions are selected. To identify the faces, the face area is first divided into several blocks and then the LBP (Local Binary Patters) feature histograms are extracted from each block and concatenated into a single global feature histogram which efficiently represents the face image.

Face detection software is usually used in for example video surveillance systems. A video camera feeds the video to a face detection system, which, in real-time, detects (and even recognizes) the input faces. As a face is detected on the input stream (a webcam), the software draws a square box on the position of the face(s).

Performance metric The performance of the face recognition software was defined as the number of images being scanned for faces per second, i.e. the inverse of the duration for one image scan. This was referred to as “Faces per second” (fa/s). Since the input of the software was grabbed from a webcam stream, one image relates to one frame. With a higher fa/s, the time elapsed to scan one video frame is shorter, and the software will detect a face faster when appearing on the webcam. The software was implemented using the Intel Thread Building Block (TBB) which is a task based programming construct to more easily expose the parallelism in the system. The **P-value** was approximated to 1.0. Figure 9.5 shows the results from the experiments on the ARM device using the Ondemand governor to the right and *Bricktop* to the left. Table 9.3 shows the energy consumption for each experiment.

Table 9.3: Energy consumption (in Joules).

Throughput [fa/s]	**12**	**8**	**4**
Ondemand	104.34	67.57	25.07
Bricktop	79.20	23.91	13.16

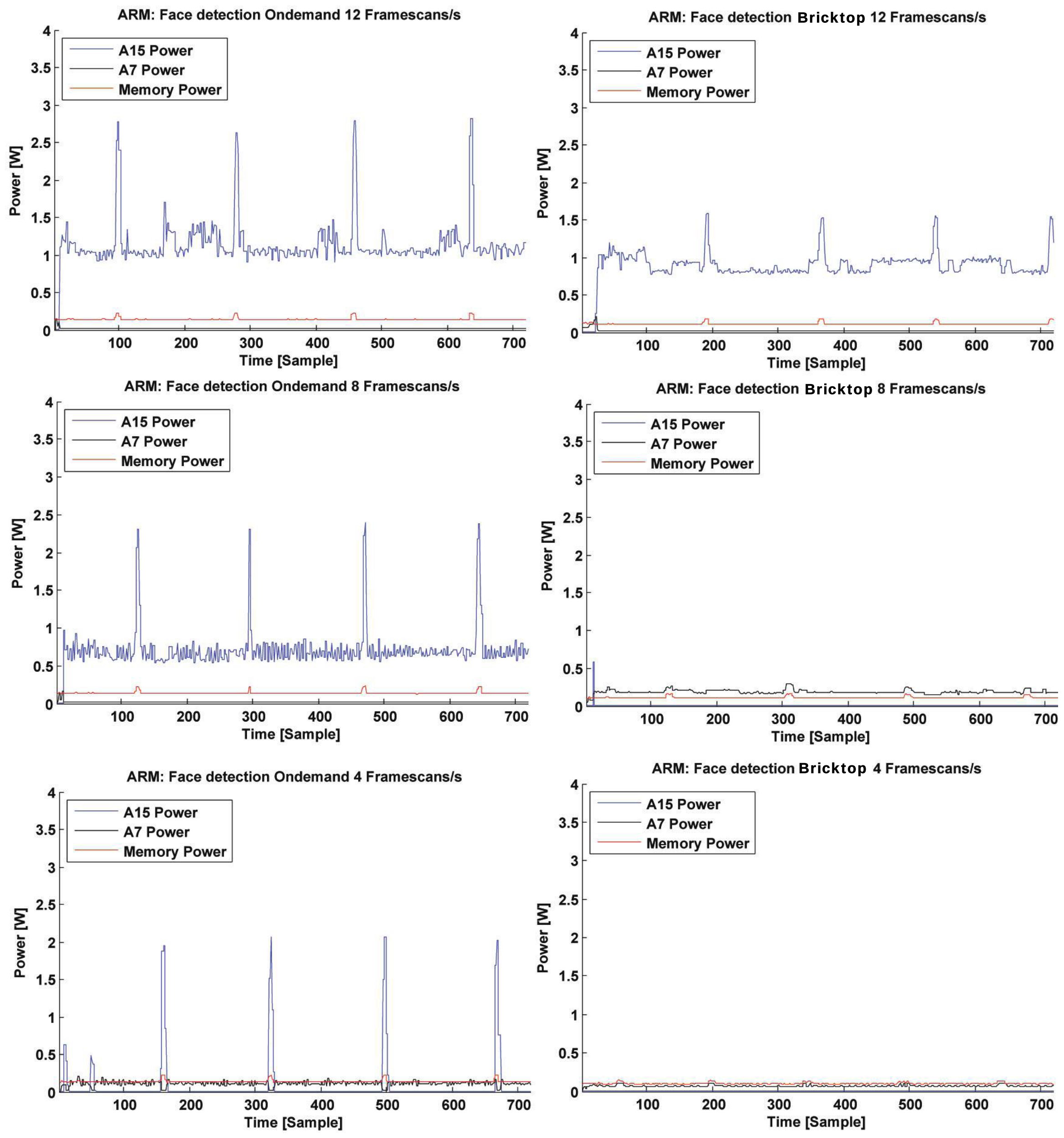


Figure 9.5: Power dissipation from Facedetection experiment

Chapter 10

Discussion and Conclusions

"*I just had one of those brain lernin experiences!*"
— Ricky, Trailer Park Boys - 2001

Energy efficiency in computer systems is not only a problem posed by operational costs apparent in the electrical bill and recharge intervals of mobile devices. Energy efficiency is currently also introduced as a limit for performance in current microprocessors because of the end of Dennard scaling [16]. Dennard scaling forecasts that the power density of transistors stays constant as the manufacturing technology decreases. This means that as transistors become smaller, energy efficiency is improved by transistor technology such that the power remains in proportion to the area. In recent hardware, this scaling no longer applies since the voltage range used in transistors no longer can be lowered. The end of Dennard scaling brings consequences such as dark silicon, which means that all processing power on a chip cannot be used simultaneously because of the limited power envelope. The statement in quote:

"*Moore's Law gives us transistors...*
Dennard scaling made them useful."

— William J. Dally, Nvidia - 2015

argues that energy efficiency is required also for gaining performance. Energy efficiency is thus no longer a hardware-only based area of research, but software must be included in reaching this goal.

We have demonstrated the effect of power proportionality on computer systems as measurement of *how much of the power* is used for actual processing and how much is waste power. In order to keep the power proportionality high, i.e. not waste energy, the hardware mechanism used to save power should be controlled on a higher level of abstraction by the application software via a runtime system. To achieve this goal, we propose to extend the interaction between software and hardware by creating an interface allowing applications to directly influence decision making in the runtime system. This interface allows *energy awareness* in applications. The contribution of this thesis is a framework with guidelines for creating energy aware software. The concept is usable in both legacy software and target specific programming environments such as in dataflow constructs. Two important recommendation have been submitted for creating energy aware software and a runtime system has been implemented to control the resource allocation.

1) Energy-aware mapping. Modern multi-core systems dissipate power based on two main factors: 1) The clock frequency, which influences mostly the dynamic and slightly the static power. 2) The number of active cores, which influences the static power. In traditional multi-core systems, applications are mapped onto all available cores (in case of sufficient parallelism in the application). This approach promotes the usage of *many cores with low clock frequency* rather than *few cores with high clock frequency* i.e. the minimization of dynamic power is favored over static power. With the current advancements in microprocessor technology, the static-to-dynamic power ratio is changing, and the static power is expected to dominate the total power dissipation in next generation processors. Instead of exclusively maximizing the parallelism in the system to reach the minimum clock frequency, this thesis proposes methods to find the middle ground between clock frequency and the number of cores to minimize the total power dissipation.

2) Energy-aware execution. The typical execution strategy *Race-to-Idle* aims at executing software at a maximum speed to minimize the execution time – this is currently the default behavior in most systems. Because of the power balance in modern multi-core architectures, this strategy is very energy inefficient. A QoS-aware execution strategy was therefore suggested in this thesis. It allows applications to execute only *as fast as required*, appointed by the user and by direct application performance levels. Using this execution strategy, only the necessary amounts of resources are allocated resulting in, on average, lower power dissipation, which reduces the energy consumption.

Energy-aware software: a new paradigm To utilize the energy-aware methodologies, energy-awareness should be a part of the natural development environment from programmer- to language- to compiler- and runtime. Meta-data-based approaches have for decades been a natural part of parallel programming such as pragmas in OpenMP, keywords in Cilk and initializations in OpenCL. In other words, the programmer is responsible for including a set of parameters in a program to increase its efficiency (in the traditional case, *performance*). To increase energy efficiency, the programmer should

be allowed to input information regarding the structure and the behavior of the program, to be used as control inputs for resource allocation. This thesis has presented a policy to extend the energy-awareness of software by including the following meta-data in the applications:
1) Application *Parallelism* to enable *energy-aware mapping.*
2) Application *Performance* to enable *energy-aware execution.*
A runtime system, *Bricktop*, was implemented to use the meta-data and continuously allocate the optimal amount of resources, which minimizes the energy consumption.

10.0.4 Future directions

The work presented in this thesis has been a mixture of scientific methodologies, implementation engineering and practical experimental design. Covering such a broad spectrum consequently leaves some stones unturned, but also opens new doors for future research. There are three main future directions, which are summarized below.

Methodology Describing power consumption in modern microprocessors becomes more and more important in order to fully utilize the power saving features in the hardware. A model of the system is helpful since mathematical functions can replace the real-life behavior and be used in software algorithms. As my thesis has suggested a top-down power model, more models are yet to be investigated, more exactly:

- Is there a better way to model the system?
- Are the experiments creating the model sufficient?
- Is the mathematical expression capable of representing the system with enough accuracy?

Although the optimization methodology was evaluated in this thesis, more aspects of it would increase its robustness and efficiency:

- Is there a better optimization algorithm than the ones evaluated?
- Is the algorithm stable under all conditions?
- How close to the global optimum must an algorithm guarantee for feasible action?

Implementation Secondly, implementing a theoretical framework in practice requires additional parameters regarding the platform. The work in my thesis has been assuming the Linux OS environment since it is the most popular platform for embedded systems and mobile phones, it is also the most popular server platform and it is fairly much used even as a desktop

OS. When relying on a pre-created software infrastructure, one must perform system programming according to its facilities. The main future work questions regarding the implementation are:

- What is the best interface between application and power management? Use library call, system call, `sysfs` or other?
- What is the optimal power management period to minimize the latency, minimize overhead and prohibit control instability?
- Is there a scalability problem when increasing the applications to a very large number?
- Can the actuator latency be predicted and integrated into the optimizer?

The integration of the DVFS/DPM latencies as well as the task migration overhead would allow the runtime system to more accurately predict the efficiency of using the power management techniques. As the latencies have been determined, it is left to future work to include then in the optimization algorithm for a more proactive control.

Experiment Experimental design on real hardware requires an actual implementation of the mechanism under investigation as well as a test bench onto which to apply the implementation. The test bench must represent the real use-cases to produce useful results for the research community. In this thesis, much of the applied evaluation tools have been inherited from other communities such as video processing and other streaming applications. These evaluation tools were chosen because of their popularity from the point of view of the everyday-user. While a significant set of tools have been used to evaluate the power management system, future work can be applied on the following questions:

- Is the current test bench complete or should different tools be used?
- Should the system be tested on more diverse hardware?
- Are the experiments representing real use-cases well enough?

Finally, as microprocessors become more diverse and heterogeneous models such as the Exynos 5422, the AMD Kaveri and the Parallella Adapteva appear on the market, the system should support platforms with different processors types. This addition does not only require power management, but also a modified scheduling algorithm since the energy efficiency is not only dependent on the *amount* of resources used but also the *type*. As some algorithms better fit, for example, a digital signal processor, other algorithms better fit general purpose processors. It is therefore the task of a runtime system to match this **Fitness** from software construction and compiler technique to the hardware architecture, and to coordinate the power saving features to maximize the energy efficiency.

Bibliography

[1] Jan Aagedal. *Quality of Service Support in Development of Distributed Systems*. PhD thesis, University of Oslo, Oslo, Norway, March 2001.

[2] P.C. Adell, H.J. Barnaby, R.D. Schrimpf, and B. Vermeire. Band-to-band tunneling (bbt) induced leakage current enhancement in irradiated fully depleted soi devices. *Nuclear Science, IEEE Transactions on*, 54(6):2174–2180, Dec 2007.

[3] Asim Ali, Rui Jia, Abdelkarim Erradi, Sherif Abdelwahed, and Rachid Hadjidj. Towards model-based management of database fragmentation. In *Presented as part of the 8th International Workshop on Feedback Computing*, Berkeley, CA, 2013. USENIX.

[4] M. Anis and M.H. Aburahma. Leakage current variability in nanometer technologies. In *System-on-Chip for Real-Time Applications, 2005. Proceedings. Fifth International Workshop on*, pages 60–63, July 2005.

[5] Adnan Ashraf, Fareed Jokhio, Tewodros Deneke, Sebastien Lafond, Ivan Porres, and Johan Lilius. Stream-based admission control and scheduling for video transcoding in cloud computing. In Pavan Balaji, Dick Epema, and Thomas Fahringer, editors, *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, page 482–489. IEEE Computer Society, 2013.

[6] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André; Wacrenier. Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurr. Comput. : Pract. Exper.*, 23(2):187–198, February 2011.

[7] John Backus. Can programming be liberated from the von neumann style?: A functional style and its algebra of programs. *Commun. ACM*, 21(8):613–641, August 1978.

[8] K. Bhatti, C. Belleudy, and M. Auguin. Power management in real time embedded systems through online and adaptive interplay of dpm

and dvfs policies. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pages 184–191, 2010.

[9] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-Erik Årzén. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3):16–30, June 2003.

[10] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen. Low-power cmos digital design. *Solid-State Circuits, IEEE Journal of*, 27(4):473 –484, apr 1992.

[11] Sangyeun Cho and R.G. Melhem. On the interplay of parallelization, program performance, and energy consumption. *Parallel and Distributed Systems, IEEE Transactions on*, 21(3):342–353, 2010.

[12] A. Cristea and T. Okamoto. Speed-up opportunities for ann in a time-share parallel environment. In *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, volume 4, pages 2410–2413 vol.4, 1999.

[13] LeandroFontoura Cupertino, Georges Da Costa, and Jean-Marc Pierson. Towards a generic power estimator. *Computer Science - Research and Development*, pages 1–9, 2014.

[14] Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, and Mark Horowitz. Cpu db: Recording microprocessor history. *ACM Applicative conference*, 10, April 2012.

[15] Benoît Dupont de Dinechin, Pierre Guironnet de Massas, Guillaume Lager, Clément Léger, Benjamin Orgogozo, Jérôme Reybert, and Thierry Strudel. A distributed run-time environment for the kalray mppa-256 integrated manycore processor. In *ICCS'13*, pages 1654–1663, 2013.

[16] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, Oct 1974.

[17] Jack B. Dennis and David P. Misunas. A preliminary architecture for a basic data-flow processor. In *Proceedings of the 2Nd Annual Symposium on Computer Architecture*, ISCA '75, pages 126–132, New York, NY, USA, 1975. ACM.

[18] Gang Ding. A control theoretic approach to analyzing peer-to-peer searching. In *Presented as part of the 8th International Workshop on Feedback Computing*, Berkeley, CA, 2013. USENIX.

[19] D. Messerschmitt E. Lee. Static scheduling of synchronous data-flow programs for digital signal processing. *IEEE Transactions on Computers*, pages 24–35, 1987.

[20] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[21] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. A mechanistic performance model for superscalar out-of-order processors. *ACM Trans. Comput. Syst.*, 27(2):3:1–3:37, May 2009.

[22] Stijn Eyerman, Pierre Michaud, and Wouter Rogiest. Multiprogram throughput metrics: A systematic approach. *ACM Trans. Archit. Code Optim.*, 11(3):34:1–34:26, October 2014.

[23] M. Ghasemazar, E. Pakbaznia, and M. Pedram. Minimizing energy consumption of a chip multiprocessor through simultaneous core consolidation and dvfs. In *ISCAS, Intern. Symposium on*, pages 49–52, 2010.

[24] Philip E. Gill, Walter Murray, Michael, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 1997.

[25] Yuxiong He, Charles E. Leiserson, and William M. Leiserson. The cilkview scalability analyzer. In *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '10, pages 145–156, New York, NY, USA, 2010. ACM.

[26] Werner Heisenberg. Über den anschaulichen inhalt der quantentheoretischen kinematik und mechanik. *Zeitschrift für Physik*, pages 172–198, 1927.

[27] Henry Hoffmann, Jonathan Eastep, Marco D. Santambrogio, Jason E. Miller, and Anant Agarwal. Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments. In *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, pages 79–88, New York, NY, USA, 2010. ACM.

[28] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. *SIGPLAN Not.*, 46(3):199–212, March 2011.

[29] Simon Holmbacka, Sébastien Lafond, and Johan Lilius. Power proportional characteristics of an energy manager for web clusters. In *Proceedings of the 11th International Conference on Embedded Computer Systems: Architectures Modeling and Simulation*. IEEE Press, July 2011.

[30] Simon Holmbacka, Sébastien Lafond, and Johan Lilius. A pid-controlled power manager for energy efficient web clusters. In Jinjun Chen, Wanchun Do, Jianxun Liu, Laurence T. Yang, and Jianhua Ma, editors, *Proceedings of the International Conference on Cloud and Green Computing (CGC2011)*, number 0, pages 721–728. IEEE Computer Society, 2011.

[31] Simon Holmbacka, Wictor Lund, Sébastien Lafond, and Johan Lilius. Lightweight framework for runtime updating of c-based software in embedded systems. In Rik Farrow, editor, *5th Workshop on Hot Topics in Software Upgrades*, page 1–6. Usenix association, 2013.

[32] Simon Holmbacka, Wictor Lund, Sébastien Lafond, and Johan Lilius. Task migration for dynamic power and performance characteristics on many-core distributed operating systems. In Peter Kilpatrick, Peter Milligan, and Rainer Stotzka, editors, *Proceedings of the 21st International Euromicro Conference on Parallel, Distributed and Network-based Processing*, page 310–317. IEEE Computer society, 2013.

[33] Simon Holmbacka, Erwan Nogues, Maxime Pelcat, Sébastien Lafond, and Johan Lilius. Energy efficiency and performance management of parallel dataflow applications. In Ana Pinzari and Adam Morawiec, editors, *The 2014 Conference on Design & Architectures for Signal & Image Processing*, page 133 – 141. ECDI Electronic Chips & Systems design initiative, 2014.

[34] Simon Holmbacka, Dag Ågren, Sébastien Lafond, and Johan Lilius. Qos manager for energy efficient many-core operating systems. In Peter Kilpatrick, Peter Milligan, and Rainer Stotzka, editors, *Proceedings of the 21st International Euromicro Conference on Parallel, Distributed and Network-based Processing*, page 318–322. IEEE Computer society, 2013.

[35] I. Hong, D. Kirovski, Gang Qu, M. Potkonjak, and M.B. Srivastava. Power optimization of variable voltage core-based systems. In *Design Automation Conference, 1998. Proceedings*, pages 176–181, 1998.

[36] J. Howard, S. Dighe, Y. Hoskote, and Vangal. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 108 –109, feb. 2010.

[37] Kai Huang, L. Santinelli, Jian-Jia Chen, L. Thiele, and G.C. Buttazzo. Adaptive dynamic power management for hard real-time systems. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 23–32, 2009.

[38] Kai Huang, L. Santinelli, Jian-Jia Chen, L. Thiele, and G.C. Buttazzo. Periodic power management schemes for real-time event streams. In *CDC/CCC 2009. Proceedings of the 48th IEEE Conference*, pages 6224–6231, 2009.

[39] Fredric Hällis, Simon Holmbacka, Wictor Lund, Robert Slotte, Sébastien Lafond, and Johan Lilius. Thermal influence on the energy efficiency of workload consolidation in many-core architecture. In Raffaele Bolla, Franco Davoli, Phuoc Tran-Gia, and Tuan Trinh Anh, editors, *Proceedings of the 24th Tyrrhenian International Workshop on Digital Communications*, page 1–6. IEEE, 2013.

[40] Kelly Iondry. *Iterative Methods for Optimization.* Society for Industrial and Applied Mathematics, 1999.

[41] M. Tim Jones. Inside the linux scheduler. `http://www.ibm.com/developerworks/linux/library/l-scheduler/`, Jun 2006.

[42] A.B. Kahng, Seokhyeong Kang, R. Kumar, and J. Sartori. Enhancing the efficiency of energy-constrained dvfs designs. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(10):1769–1782, Oct 2013.

[43] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th ACM symposium on Theory of computing*, STOC '84, pages 302–311. ACM, 1984.

[44] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, Dec 2003.

[45] Woonseok Kim, Dongkun Shin, Han-Saem Yun, Jihong Kim, and Sang-Lyul Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE*, pages 219–228, 2002.

[46] Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, and Randy H. Katz. Napsac: design and implementation of a power-proportional web cluster. In *Proceedings of the first ACM SIGCOMM workshop on Green networking*, Green Networking '10, pages 15–22, New York, NY, USA, 2010. ACM.

[47] Jacob Leverich, Matteo Monchiero, Vanish Talwar, Partha Ranganathan, and Christos Kozyrakis. Power management of datacenter workloads using per- core power gating, 2009.

[48] Yi Liu, Yanchao Zhu, Xiang Li, Zehui Ni, Tao Liu, Yali Chen, and Jin Wu. Simnuma: Simulating numa-architecture multiprocessor systems efficiently. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 341–348, Dec 2013.

[49] Yongpan Liu, Huazhong Yang, R.P. Dick, Hui Wang, and Li Shang. Thermal vs energy optimization for dvfs-enabled processors in embedded systems. In *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, pages 204–209, March 2007.

[50] Robert Love. *Linux Kernel Develoupment.* Addison-Weasly, 3 edition, June 2010.

[51] D. Lucanin and I. Brandic. Pervasive cloud controller for geotemporal inputs. *Cloud Computing, IEEE Transactions on*, (99), 2015.

[52] Rod Mahdavi. Case study: Opportunities to improve energy efficiency in three federal data centers. *U.S. Department of Energy's Federal Energy Management Program*, May 2014.

[53] M. Marinoni, M. Bambagini, F. Prosperi, F. Esposito, G. Franchino, L. Santinelli, and G. Buttazzo. Platform-aware bandwidth-oriented energy management algorithm for real-time embedded systems. In *ETFA, 2011 IEEE 16th Conference on*, pages 1–8, 2011.

[54] SallyA. McKee and RobertW. Wisniewski. Memory wall. In David Padua, editor, *Encyclopedia of Parallel Computing*, pages 1110–1116. Springer US, 2011.

[55] Francisco Javier Mesa-Martinez, Ehsan K. Ardestani, and Jose Renau. Characterizing processor thermal behavior. *SIGPLAN Not.*, 45(3):193–204, March 2010.

[56] Mwaikambo, Raj, Russell, Schopp, and Vaddagiri. Linux Kernel Hotplug CPU Support. *Proceedings of the Ottawa Linux Symposium*, pages 181–194, 2004.

[57] A. M'zah and O. Hammami. Parallel programming and speed up evaluation of a noc 2-ary 4-fly. In *Microelectronics (ICM), 2010 International Conference on*, pages 156–159, Dec 2010.

[58] Vincent Nollet, Diederik Verkest, and Henk Corporaal. A safari through the mpsoc run-time management jungle. *Journal of Signal Processing Systems*, 60(2):251–268, 2008.

[59] Jaehyun Park, Donghwa Shin, Naehyuck Chang, and M. Pedram. Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pages 419–424, Aug 2010.

[60] L. Parolini, B. Sinopoli, B.H. Krogh, and Zhikui Wang. A cyber-physical systems approach to data center modeling and control for energy efficiency. *Proceedings of the IEEE*, 100(1):254–268, Jan 2012.

[61] Maxime Pelcat, Jonathan Piat, Matthieu Wipliez, Slaheddine Aridhi, and Jean-François Nezan. An open framework for rapid prototyping of signal processing applications. *EURASIP journal on embedded systems*, 2009:11, 2009.

[62] Sreeram Potluri, Karen Tomko, Devendar Bureddy, and Dhabaleswar K. Panda. Intra-mic mpi communication using mvapich2: Early experience. *Texas Advanced Computing Center (TACC)-Intel Highly Parallel Computing Symposium*, April 2012.

[63] M.D. Powell and T.N. Vijaykumar. Resource area dilation to reduce power density in throughput servers. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 268–273, Aug 2007.

[64] Bharathwaj Raghunathan, Yatish Turakhia, Siddharth Garg, and Diana Marculescu. Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 39–44, March 2013.

[65] Amir-Mohammad Rahmani. *Exploration and Design of Power-Efficient Networked Many-Core Systems.* PhD thesis.

[66] T. Rauber and G. Runger. Energy-aware execution of fork-join-based task parallelism. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 231–240, 2012.

[67] Erik Reed, Abe Ishihara, and Ole J. Mengshoel. Adaptive control of apache web server. In *Presented as part of the 8th International Workshop on Feedback Computing*, Berkeley, CA, 2013. USENIX.

[68] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. *Proceedings of the IEEE*, 91(2):305–327, Feb 2003.

[69] M. Sadri, A. Bartolini, and L. Benini. Single-chip cloud computer thermal model. In *Thermal Investigations of ICs and Systems (THERMINIC), 2011 17th International Workshop on*, pages 1–6, 2011.

[70] H. Sasaki, S. Imamura, and K. Inoue. Coordinated power-performance optimization in manycores. In *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*, pages 51–61, 2013.

[71] Robert Schöne, Daniel Molka, and Michael Werner. Wake-up latencies for processor idle states on current x86 processors. *Computer Science - Research and Development*, pages 1–9, 2014.

[72] Vanessa Segovia. *Adaptive CPU resource management for multicore platforms.* Licentiate thesis, Lund University, Sep. 2011.

[73] Yakun Sophia Shao and David Brooks. Energy characterization and instruction-level energy model of intel's xeon phi processor. In *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ISLPED '13, pages 389–394, Piscataway, NJ, USA, 2013. IEEE Press.

[74] Hao Shen, Jun Lu, and Qinru Qiu. Learning based dvfs for simultaneous temperature, performance and energy management. In *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, pages 747–754, March 2012.

[75] P.S. Shenoy, Sai Zhang, R.A. Abdallah, P.T. Krein, and N.R. Shanbhag. Overcoming the power wall: Connecting voltage domains in series. In *Energy Aware Computing (ICEAC), 2011 International Conference on*, pages 1–6, Nov 2011.

[76] H. Singh, K. Agarwal, D. Sylvester, and K.J. Nowka. Enhanced leakage reduction techniques using intermediate strength power gating. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 15(11):1215 –1224, nov. 2007.

[77] Joachim Sjöblom. Power efficient scheduling for a cloud system. Master's thesis, Åbo Akademi University, Turku, Finland, September 2012.

[78] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive dvfs. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8, July 2011.

[79] Venkatesh Pallipadi Alexey Starikovskiy. The ondemand governor. In *Proceedings of theLinux Symposium*, 2006.

[80] M. Tolentino and K.W. Cameron. The optimist, the pessimist, and the global race to exascale in 20 megawatts. *Computer*, 45(1):95–97, Jan 2012.

[81] C. Truchet, F. Richoux, and P. Codognet. Prediction of parallel speedups for las vegas algorithms. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 160–169, Oct 2013.

[82] B.M. Tudor and Yong-Meng Teo. Towards modelling parallelism and energy performance of multicore systems. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 2526–2529, 2012.

[83] A. Varghese, B. Edwards, G. Mitra, and A.P. Rendell. Programming the adapteva epiphany 64-core network-on-chip coprocessor. In *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 984–992, May 2014.

[84] Ankush Varma, Brinda Ganesh, Mainak Sen, Suchismita Roy Choudhury, Lakshmi Srinivasan, and Bruce Jacob. A control-theoretic approach to dynamic voltage scheduling. In *Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*, CASES '03, pages 255–266, New York, NY, USA, 2003. ACM.

[85] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: Reducing the energy of mature computations. *SIGARCH Comput. Archit. News*, 38(1):205–218, March 2010.

[86] Jeffrey S. Vetter and Patrick H. Worley. Asserting performance expectations. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, SC '02, pages 1–13, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[87] O. Villa, D.R. Johnson, M. Oconnor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, Peng Wang, P. Micikevicius, A. Scudiero, S.W. Keckler, and W.J. Dally. Scaling the power wall: A path to exascale. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 830–841, Nov 2014.

[88] C.Y. Villalpando, A.E. Johnson, R. Some, J. Oberlin, and S. Goldberg. Investigation of the tilera processor for real time hazard detection and avoidance on the altair lunar lander. In *Aerospace Conference, 2010 IEEE*, pages 1 –9, march 2010.

[89] Liang Wang and K. Skadron. Implications of the power wall: Dim cores and reconfigurable logic. *Micro, IEEE*, 33(5):40–48, Sept 2013.

Part II

Original Publications

Paper I

Power Proportional Characteristics of an Energy Manager for Web Clusters

Simon Holmbacka, Sébastien Lafond, Johan Lilius

Originally published *Proceedings on Embedded Computer Systems: Architecture, Modeling and Simulation, 2011 IEEE International Conference*, pages 51–58. Samos, Greece.

Power Proportional Characteristics of an Energy Manager for Web Clusters

Simon Holmbacka, Sébastien Lafond, Johan Lilius
Department of Information Technologies, Åbo Akademi University
Joukahaisenkatu 3-5 FIN-20520 Turku
firstname.lastname@abo.fi

***Abstract*—Energy consumption is a major issue in data centers operating 24 hours a day, 7 days a week. The power dissipated by a web cluster is not proportional to the numbers of incoming requests if only DVFS (Dynamic Voltage Frequency Scaling) is used. This is because of the nonlinear power efficiency of DVFS, the large load fluctuation in web services and the typical CPU utilization rates of a server.**

This paper presents a system level controller making a cluster of low-power servers power proportional by managing the resources on the platform. Our controller uses sleep states to switch on or off CPUs in order to continuously match the current workload with the system capacity. Methods from control theory are used to drive the CPUs from and into sleep states. The power proportional characteristics of the proposed energy manager are studied for different workload patterns. Results from system simulation show that power proportionality is obtainable but only with appropriate parameters set on the controller.

I. Introduction

Energy efficiency and power density are key issues for data centers. These factors do not only affect the operational costs and ecological footprint, but have also an important impact on the possibility to construct or expend data centers.

The *Efficient Servers* project [1] evaluated the increase of electric power consumption of servers in Western Europe at 37% between 2003 and 2006 [2]. In 2007 the energy consumed in data centers in Western Europe was 56 TWh and is projected to increase to over 100 TWh per year by 2020 [3].

In current servers, there is a mismatch between the energy-efficiency characteristics and the behavior of server class workloads as their most common operating mode corresponds to the lowest energy-efficiency region [4]. When using DVFS as power management technique, an energy efficient server still consumes about half of its energy while idling.

With an average of 10 to 50 percent CPU utilization for servers [5] and the large load fluctuation found in typical web services [6], the use of slower but more energy-efficient cores could match the workload more efficiently with a much finer granularity than server-grade cores. A cluster of mobile processors can provide the same computational power as server-grade processors, but with a lower power density. The usage of mobile processors also aims at obtaining cheaper server facilities by minimizing the need of active cooling infrastructure.

Because switching on and off a CPU is orders of magnitude slower than changing its voltage and frequency, a cluster of such low-power CPUs needs an energy manager on system level i.e. a component controlling the whole cluster as one entity and continuously matching the current workload with the whole cluster capacity.

This paper analyses for different workload patterns the proportional characteristics of an energy manager that uses sleep states to dynamically adjust the system capacity according to the workload so that minimal performance penalty and maximum reduction in energy consumption is obtained.

II. Related work

Previous work has been done in the area of using sleep states to reduce the energy consumption of mobile processors. The authors in [7] are proposing a mixture of high-end Xeon servers combined with low-end mobile processors in order to achieve a fine granularity of system capacity in relation to the workload. All processing elements in the system uses sleep states to shut down the CPUs during low workload and thus reduce the energy consumption. Once the system recognizes an increase in workload, the system activates the processing elements in accordance with their different capacities and wake-up times. To determine the power proportionality, experiments were conducted on two different types of workload patterns, which results concluded in a power proportional system.

In our approach, the power management system uses control theory as basis for the capacity adaption. We argue that the use of the PID controller could, with correctly set parameters, create a near optimal adaption of system capacity to the workload. Moreover, we intend to use a cluster consisting only of low-end mobile processors to gain finer power granularity of the whole system.

The authors in [8] present a sleep state based power manager for server grade CPUs together with PSUs (Power Supply Units) in a so called *RAILS*-configuration (Redundant Array for Inexpensive Load Sharing). Smaller low-power PSUs are used instead of one powerful, since a RAILS-configuration will make the PSUs operate in their most energy efficient sweet spot. As the power need increases more PSUs are enabled to provide the sufficient power needed. Similarly to [7] the CPU cores are switched on and off according to the workload to give a better power proportionality of the system. The method of anticipating the workload curve was not mentioned, but an average of 74 % energy reduction was achievable according to the authors [8]. The power proportionality was determined based on the wake-up time for the core, and would in best case result in a linear function.

We have used the idea from both of the previous works together with the implementation of a PID controller [9] to adjust the capacity of the system. By using a larger number of low power CPUs we argue that – by having a finer granularity – we could achieve a higher energy reduction while keeping the power proportionality constant and obtaining a sufficient performance.

III. Power Proportional Web Cluster

A. System Level Power Management

We created a power manager which adjusts the system capacity dynamically in order to save energy. The manager uses sleep states to switch on and off cores according to the current need and according to the anticipated future workload. The simulated cluster uses ARM Cortex-A8 processors used in the BeagleBoard and its wake-up time was measured by experiments to roughly 800 ms. By using the measured capacity of a Cortex-A8 the energy consumption for a many-core cluster was simulated. The basic processing element in this paper is referred to as a *core*, since embedded systems with multi-core configurations have recently been available.

1) **Overview:** The outline of the framework is shown in Figure 1. The framework is based on input in form of requests made to the service. The framework shows the output based on data from a PID-controller and a feedback loop, which sends information to the compare-block regarding the needed capacity of the system.

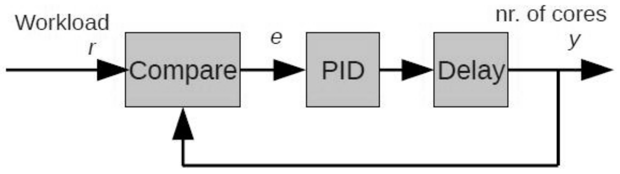


Fig. 1: Structure of the simulation framework

2) **System capacity:** System capacity is measured as the number of requests per second the system as a whole is able to handle. The compare block reads the workload with a certain sample rate and divides this number with the current capacity of the system. The ratio of this division determines the QoS (Quality of Service) output. When the capacity monitor in the compare-block notices a higher workload than the system is able to handle, it sends the ratio between the workload and the capacity to the PID controller in form of an error value. Similarly when the capacity of the system exceeds the workload, the monitor sends a negative error value to the controller which in turn switches off CPU cores.

Since switching on and off CPU cores is not instantaneous, the framework uses a delay to postpone the control signal to the workload comparator. For this function, the simulation framework implements a unit-delay block with a configurable delay length. This simulates the actual delay introduced in the change of CPU state.

The framework uses one *static* core. This core will constantly be active and is used to instantaneously handle the small amount of requests that are made between request peaks. The number of static cores is also configurable.

3) **QoS value:** A trade-off to energy consumption is the performance and response time of the system. By lowering the capacity of the system the performance will drop – this leads occasionally to an increased response time for certain requests. *Quality of Service* is the measurement on how well the system performs compared to a pre-defined value. Our simulations show a drop in QoS as soon as a request is delayed more than the selected deadline. The amount of delayed requests compared to non-delayed requests results in the QoS value. If every request is handled before their deadlines the QoS will be 100 %, if half of the requests are handled before the deadline the QoS will be 50 % etc.

4) **PID controller:** The controller block in Figure 1 includes a PID controller which, based on methods from control theory, adjusts the capacity of the system. The obtained difference between y and r is called the control error e, which is the *a priori* result from the capacity comparison in the previous block. The output y of the PID controller partly shows the amount of cores needed to achieve a sufficient performance and partly generates feedback data to the comparison block in the next time frame. The aim of the feedback loop is to minimize the control error and to thereby achieve equilibrium in the system.

The behavior of the PID controller is determined by setting P, I and D values in the controller. The value of P determines how fast the controller reacts on a change in the reference value r. The value I, which is the inverse time constant of the controller, determines the integral effect of the control error. The derivative part, which is adjusted by the parameter D, predicts the future input based on the previous input.

5) **Final energy consumption:** The system shows the power output as a multiple of the amount of active cores and their power dissipation. The cores are assumed to run on the highest possible clock frequency once activated, and retain this clock frequency until they are shut down. The final energy consumption is the sum over the power dissipation for all time frames in the simulation.

B. Power Proportionality

While the power manager shows a promising result in energy reduction, we need to investigate how well it scales in a growing web cluster. To be able to apply the power manager into a large cluster the proportionality of the workload compared to the power dissipated must be constant. This means that if the workload increases by a certain factor, the power dissipation should also increase with the same factor.

To measure the proportionality we created different workload patterns against which the power dissipation was compared. The behavior of the system was simulated by inserting the workload patterns into the simulation framework.

IV. Simulation data

Our simulations will be based partly on specially generated request patterns and partly on real web server data, which allows for a comparison of power proportionality in different situations. The first simulations are executed against trivial cases to evaluate and clearly illustrate the theory. Later the real web server data will show the obtained proportionality in a real-world scenario.

A. Request patterns

1) **Linear cone:** The first pattern to investigate energy proportionality is generated by requests made according to a linear cone as seen in Figure 2. Requests are made with certain increments and a selected step size. The increment determines how much the requests increase for each step, the length of which is selected by a step size.

For the energy to be proportional to the requests, the power dissipation for all time frames should increase linearly according to the workload curve. A linear increase in the power dissipation would scale the energy consumption well in a large web cluster.

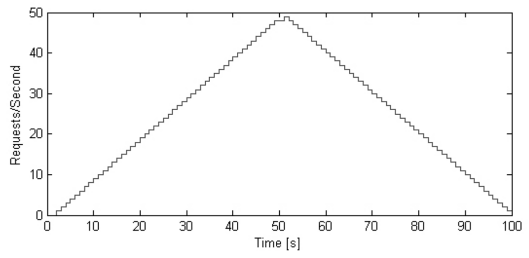


Fig. 2: Linear workload pattern

2) **Exponentially increasing cone:** The exponentially increasing cone in Figure 3 is created by incrementing the steps multiplied by a certain constant. The energy proportionality of an exponentially increasing pattern should be followed by a similar pattern in the power dissipation. By investigating different patterns, we will be able to determine the proportionality characteristics of the power management system.

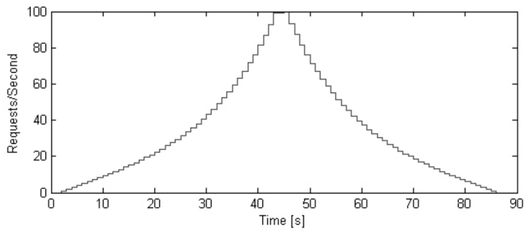


Fig. 3: Exponential workload pattern

3) **Web server requests:** We also used requests made to a Finnish web space provider [10] to compare the results of power proportionality in a real-world scenario. The request samples were obtained from 1 Nov. 2010 and simulated for 30 minutes and shown in Figure 4. The simulation data is freely available from [11].

B. PID-parameters

The values of the PID-parameters P, I and D determines how the controller should react to its input signal. The parameters were chosen based on a heuristic method, and tuned until desired result was achieved. The simulation framework supports currently only static PID-parameters, but could eventually be further developed to handle dynamic values. The value of the delay after the PID-block was chosen, based on conducted experiments, to 1000 ms in order to ensure the necessary delay of the wake-up time, which was measured to 800 ms. A sample time of 250 ms was selected as time frame for updating the output from the controller.

C. BeagleBoard power dissipation

To obtain values for the simulation framework and be able to run a proof-of-concept simulation, the power dissipation of one BeagleBoard revision C3 low-power platform was measured. The BeagleBoard is equipped with one ARM Cortex-A8 processor-based TI-OMAP3530 chip that does not require any forced cooling system or heat sinks. The system ran Ångström Linux kernel version 2.6.32 and was controlled through a remote serial console. The operating performance points (OPPs) of the TI-OMAP3530 chip were used to dynamically scale the clock frequency and voltage of the ARM subsystem. The OPPs were accessed through the Linux ACPI. To avoid unwanted energy consumption, the display subsystem of the TI-OMAP3530 was disabled. The BeagleBoard includes a resistor, which provides a way to measure the current consumption used by the board. The voltage drop across the resistor was measured for each OPP and the corresponding power was calculated. The obtained power values of the system running at respective voltage and clock frequency are displayed in Table I. To ensure that the load would remain constant during the measurements, the processor was stressed to 100 % utilization using a simple program that recursively counts Fibonacci numbers. The highest OPP (720 MHz) was used in the simulation framework to represent the power dissipation for the active core.

TABLE I: Measured power dissipation of the BeagleBoard

Frequency [MHz]	720	600	550	500	250	125
Voltage [V]	1.35	1.35	1.27	1.20	1.06	0.985
Power [W]	1.40	1.15	1.05	1.00	0.65	0.55

D. BeagleBoard load capacity

The system capacity is dependent on both the number of CPU cores in use and their capacity. We needed to determine the capacity of a BeagleBoard in order to run a realistic simulation.

Experiments were therefore conducted, which results defined the system capacity of one BeagleBoard. The test tool in use was Autobench [12] which generates requests to an Apache [13] server running on the BeagleBoard. The number of requests per second generated by Autobench started from a

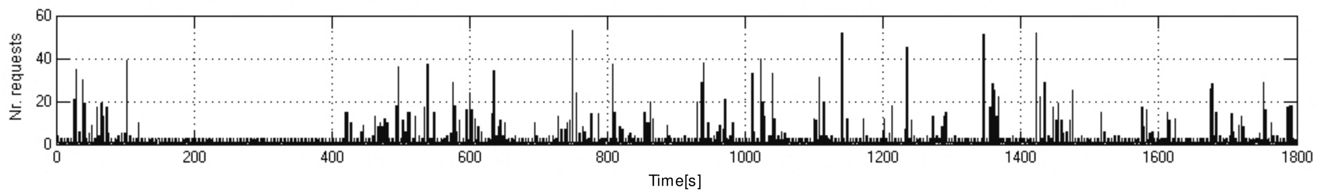


Fig. 4: Workload sample from [10] 1. November 2010

selected number and increased by specified increments. When the number of requests per second start to exceed the capacity of the host, delay-errors will start to occur as the selected deadlines for the requests are not met. The requests from Autobench are made to a selected file on the Apache server running on the BeagleBoard. Since a larger file will require more time to process, the capacity of the server is dependent on the size of the requested file. Our experiments show that a file size of 248 KB with a deadline of 1000 ms will result in the capacity of 5 requests per second – this number was used in the simulations. Related experiments in [7] result also in the capacity of 5 requests per second for the BeagleBoard which, as stated, would compare to a file size of 248 KB.

The file size was constant for each simulation with a maximum amount of 10 or 20 cores available. Using these numbers, the cluster would have a theoretical maximum capacity of 50 or 100 requests per second.

V. Simulation results

The framework for simulation was set-up according to the results from the experiments presented in the previous section. The simulations were run in three phases: using a linear cone, an exponential cone and real web server data.

A. Linear cone

The first simulation used the workload pattern of a linear cone. The step size of the cone determines how fast the request rate is climbing. A longer step size means that request rate will stay constant for a longer time. The step size for the first simulation was set to 5 seconds per step. Results from the first simulation is shown in Figure 5, which displays three graphs. The first graph (1) shows the power dissipation compared to the request rate. The second graph (2) shows the amount of cores switched on in the current time frame, and the last graph (3) shows the quality of service as a function of time.

The amount of cores in Figure 5 (2) shows to be spiking each time the step increases. This peak is a result of the delay (wake-up time) the cores introduce when switching from sleep state to active state. Because of the delay, the system will not react instantaneously to the increase in request rate (workload). As soon as the system notices the drop in QoS it needs to compensate for the delayed requests by switching on additional cores. After the system has processed the delayed requests, the need for the additional cores does not exist any more and they are shut off. This phenomenon occurs every time the step increases until a stable request rate is established.

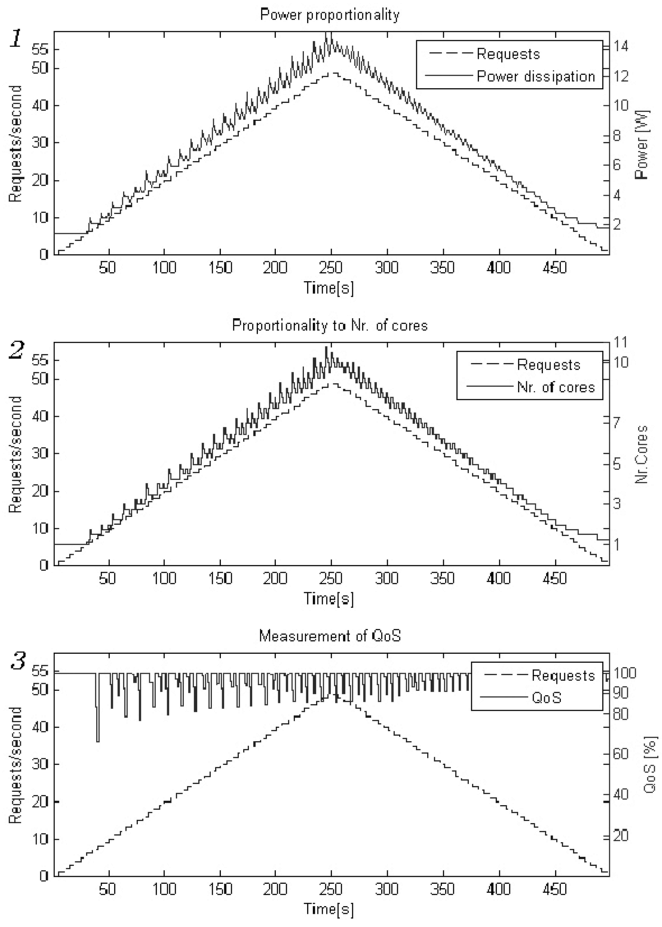
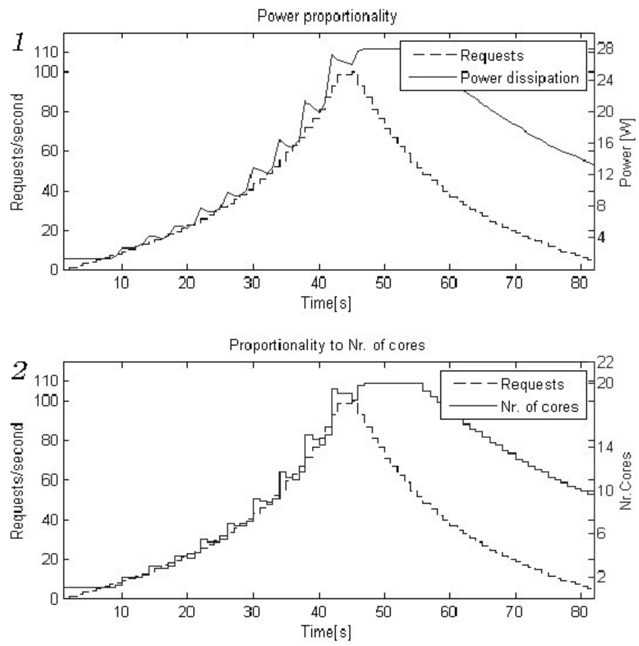


Fig. 5: Simulation of linear cone pattern

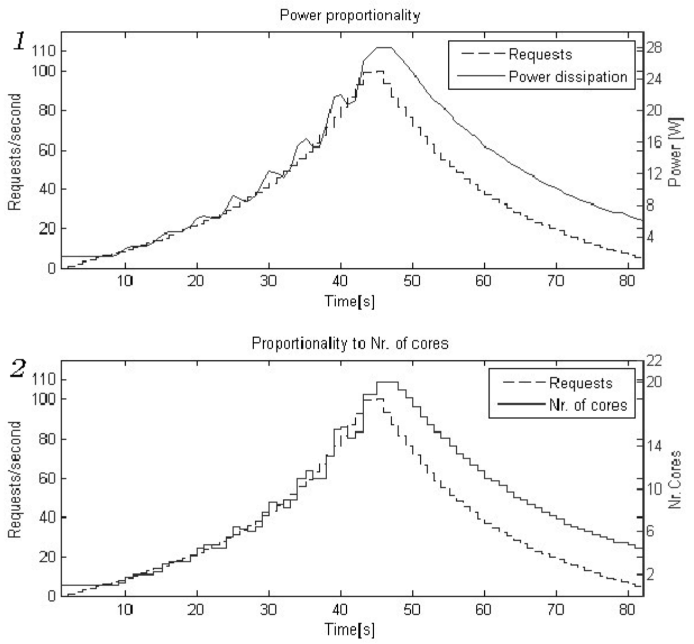
The power dissipation follows the amount of CPU cores since a CPU core is, in this model, either fully on or off. Similar peaks occurs therefore also in the power dissipation curve in Figure 5 (1) as the system compensates for the delayed requests.

Figure 5 (3) shows that the QoS value drops each time the step increases. This model, which strives to simulate a realistic case, cannot fully eliminate the QoS drop because of the delay of waking up cores. A strategy to reduce the QoS drop is to make the PID controller more QoS conservative by adjusting the parameters – this, however, also results in increased energy consumption.

Conclusions about the power proportionality can be drawn



(a) Simulation with initial parameters



(b) Simulation with improved parameters

Fig. 6: Simulation of exponential pattern with two different sets of parameters

based on the curve from Figure 5 (1). Aside from the fluctuation peaks when the steps increase, the power dissipation curve follows the workload in a linear fashion. By applying a low-pass filter on the power curve, we obtained the mean values for the fluctuating graph. The proportionality factor between power and workload is shown in Table II.

TABLE II: Measurement of power proportionality obtained from Figure 5 (1)

Time [s]	100	150	200	250	300	350	400
Req/sec	20	30	40	50	40	30	20
Power [W]	5.51	8.32	11.12	13.76	11.48	8.63	5.77
Prop. [Req/J]	3.63	3.60	3.60	3.63	3.48	3.47	3.47

Table II shows seven values derived from Figure 5 (1). The last row shows the final proportionality factor which is the ratio between the Req/sec and power, and thus uses the unit Requests/J. From the last row in the table we can see that the values of the proportionality does not fluctuate much – in fact the largest fluctuation, shown in Table II, results in a difference of 5 %.

B. Exponential cone

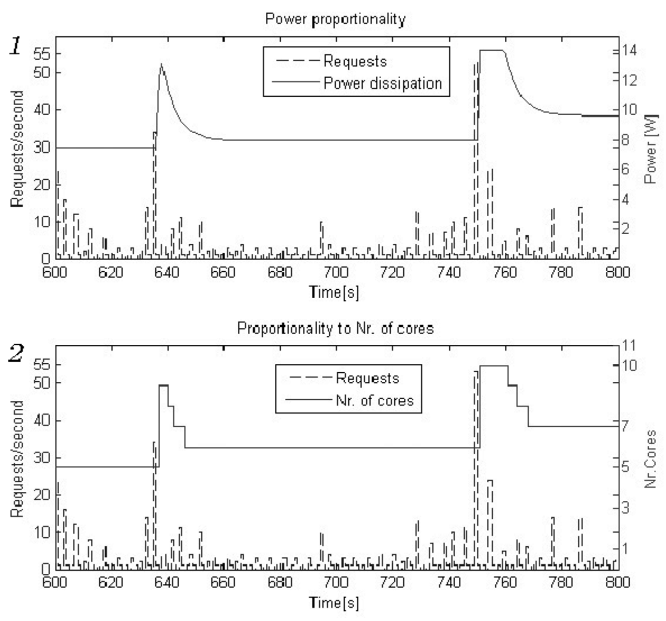
Secondly the framework was set-up with the same parameters as in the previous case, but with a different workload pattern. The second pattern was an exponentially growing request curve as shown in Figure 3. The workload used in this simulation has, in contrast to the previous simulation, a maximum value of 100 req/s to more clearly illustrate the behavior of the exponentially increasing pattern. To cope with 100 req/s we allow the system to use 20 cores instead of 10.

Figure 6a shows the result from the simulation. By using the same PID-parameters the controller fails to establish an effective output signal used for switching on and off the cores. The amount of cores in Figure 6a (2) shows to be insufficient as the curve increases. As the curve exponentially decreases, the control error remains high because of integrating property of the controller. The result is a slowly diminishing output which leads to wasted energy.

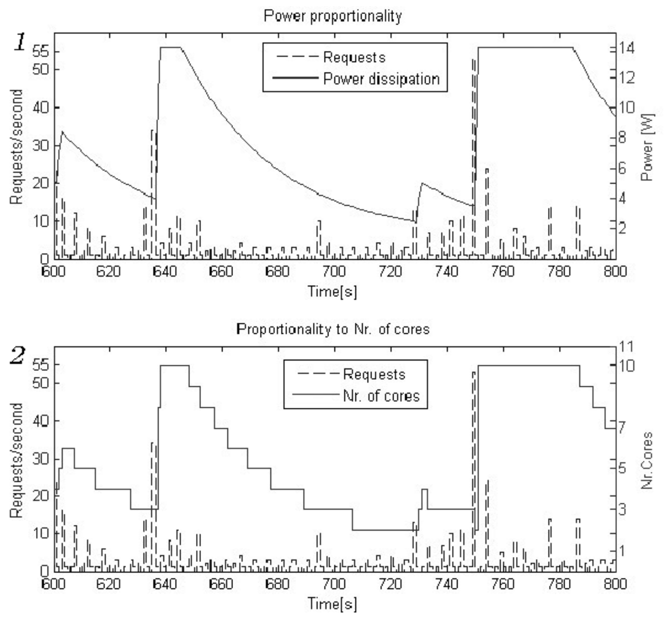
Because this simulation did not show a power proportional behavior we needed to alter the PID-parameters on the controller. After establishing a new set of parameters by experiments we run the simulation again. The new set of parameters achieved, with the same workload, better power proportionality as shown in Figure 6b. Table III shows both power dissipation and the power proportionality for both graphs in Figure 6. As seen in the table, case *b* (row 6) will show better proportionality than case *a* (row 4) because the power curve follows the workload more precisely.

TABLE III: Measurement of power proportionality obtained from Figure 6 (1)

Time [s]	15	25	35	45	55	65	75
Req/sec	40	60	80	100	80	60	40
Power 6a	4.20	7.70	15.75	27.65	26.95	20.30	15.4
Prop. 6a	3.65	4.08	3.76	3.38	1.79	1.22	0.69
Power 6b	4.20	7.70	15.40	28.00	18.55	11.90	8.40
Prop. 6b	3.65	3.82	3.61	3.34	2.60	2.08	1.55



(a) Simulation with exponential parameters



(b) Simulation with linear parameters

Fig. 7: Simulation of exponential pattern with two different sets of parameters

C. Web server data

The final simulation used the request log from a web server as workload. The PID-parameters for this simulation was chosen according to the previous simulation (exponential cone), but since the workload shows a maximum value of 50 req/s we allowed only 10 cores to be active simultaneously. The results from the simulation is shown in Figure 7a. The PID-parameters used for the exponential cone turned out to be unsuitable for controlling the workload from the web server, since the power dissipation will remain relatively constant and thus result in a poor energy management.

To achieve a better power proportionality we adjusted the PID-parameters according to the simulation with the linear cone. The results from this simulation is pictured in Figure 7b. By interpreting the curves in Figure 7 (1) we can see that the power proportionality of the system is highly dependent on the controller settings. The use of CPU cores pictured in Figure 7b (2) matches the workload better than the previous simulation. By using the new PID-parameters we achieved a better power proportionality as seen in Figure 7b (1) compared to the previous simulation showed in Figure 7a (1). The QoS was not included in Figure 7b and 7a since the numbers of cores remained high during the whole simulation and thus not resulted in any substantial QoS fluctuations.

To further improve the power proportionality factor we tuned the PID-parameters for this third case. The mentioned PID-parameters were selected to match the *spiky* workload obtained in a real-world scenario seen in Figure 4. A simulation using these parameters results in a greater energy reduction than the two previous simulations – this while keeping an acceptable QoS. The result from the last simulation is shown in Figure 9.

To calculate the power proportionality from Figures 7 and 9 we needed to time shift the power curve to accommodate for the delay introduced by the wake-up mechanism. Furthermore, we chose certain points in time where interesting measurement would take place. Table IV views the power proportionality for all three cases, with the power curve shifted one second to the left. This number displays the proportionality factor, meaning that a lower value is obtained when the system is using much energy [J] to serve few requests.

TABLE IV: Measurement of power proportionality obtained from Figure 7 (1) and 9 (1)

Time [s]	617	635	654	680	697	752	765
Req/sec	6	14	10	1	10	50	1
Prop(lin) 7b	1.07	3.43	0.89	0.17	2.45	3.57	0.07
Prop(exp) 7a	0.80	1.87	1.23	0.13	1.26	3.57	0.09
Prop(real) 9	2.54	1.31	1.18	0.66	1.30	3.57	0.61

Prop(lin) and *Prop(exp)* in Table IV represents the power proportionality of the first two simulations on real web server data. As seen in the table the fluctuations are large and close to zero in the last column. A value close to zero means that the power output of the system is much larger than the amount requests made to the system, i.e. the system is wasting much energy. The 7:th column (at time 752 s) shows equal values for all three cases. This happens due to the fact that the system is slightly overloaded, which happens if 50 or more requests are made during one second.

The results in *Prop(real)* show occasionally drops in proportionality such as at times 680 s and 765 s. This drop occurs

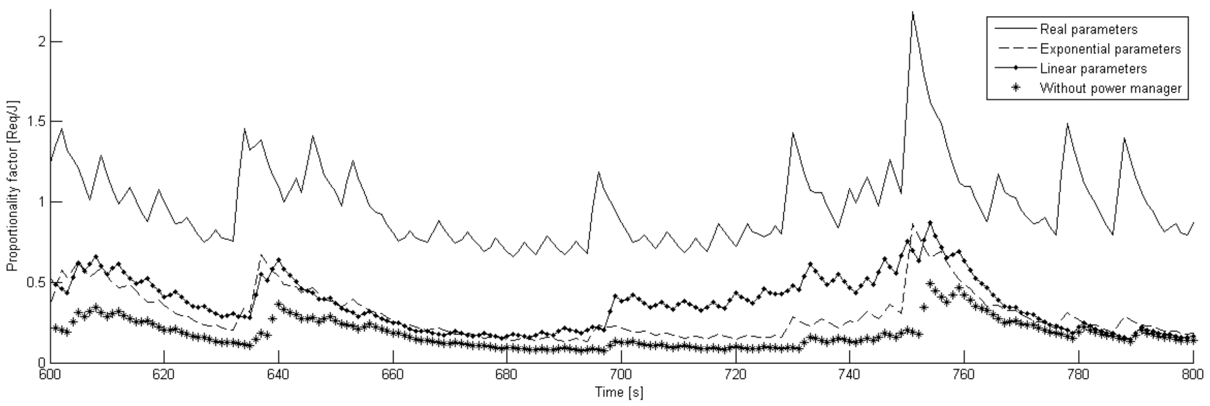


Fig. 8: Measurement of proportionality

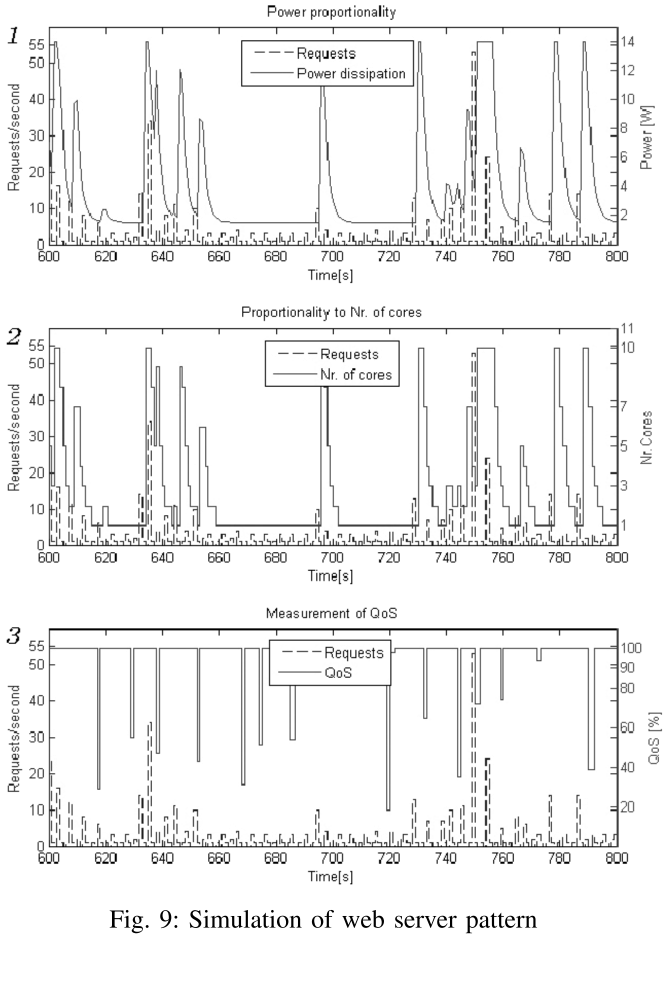


Fig. 9: Simulation of web server pattern

because of the static CPU core that will run even though the system only needs to process one request. Implementation of DVFS would in these cases be useful since the granularity of the power scaling would increase. Furthermore, in *Prop(real)* at times 635 s, 654 s and 697 s the power proportionality decreases even though the request rate is not minimal. This phenomenon is a temporal response from the system during a workload peak. The system compensates for delayed requests by temporary rising the capacity. Over time these workload peaks would not account substantially for the power proportionality of the system.

To illustrate the different proportionality factors in the three different cases, the drawn graph displays the whole time range from 600 s to 800 s (Figure 8). Furthermore, a low pass filter was used to filter the highly fluctuating output signal to better illustrate the average proportionality factor by using different PID parameters. The figure clearly shows that correct PID parameters will result in a higher proportionality factor, and thus less energy waste. The proportionality factor for a system without any power manager (all cores statically on) was also displayed in order to better compare the power proportionality of the manager.

D. Simulation summary

By observing the results from the three different simulations: *Linear cone*, *exponential cone* and *web server data*, we can state that power proportionality could be achieved by setting the appropriate PID control parameters for the workload in question. The outcome from using the power manager is a system with higher power proportionality, which means that most of the CPU power is actually used for real work rather than waiting for work to arrive.

The PID-controller reacts differently depending on the input of the controller, which means that the settings for one environment not necessarily support another environment. A run-time update of the PID variables would mean that the system should automatically accommodate the PID-parameters for not only the workload, but also the workload pattern. Another solution would be a model that reflects an *a priori* workload with sufficient precision. This model could use static PID parameters as long as the workload follows the model.

VI. Conclusions

A energy manager for a many-core web cluster was created in order for the system to show power proportional characteristics when serving alternating amount of work. The power manager matches the system capacity, every time frame, according to the workload by using a PID-controller.

This paper has investigated the power proportionality characteristics of the power manager by simulations performed on determined workload patterns. The results from the simulations were used to determine the relationship between power dissipation and workload for selected sample points.

The simulations were divided up into three different workload patterns. These three patterns were individually simulated and their respective results were compared. The simulations show that power proportionality is achievable, but only with the correct parameters set on the controller. The controller parameters determine how the controller reacts on changes in the input signal.

In the most trivial case we used a linear cone as the workload pattern. The results from the controller showed a non fluctuating and constant relationship between the workload and the total power dissipation of the system. The second simulation used an exponentially increasing workload pattern which, with the same controller parameters, did not reach a sufficient proportionality. The parameters on the controller were adjusted, after which a better result was obtained. Lastly real data from a web server was used as workload pattern. The result showed that the controller parameters from neither of the two previous cases would give a power proportional system. The controller parameters were therefore adjusted to match the *spiky* nature of web server requests, which resulted in an increased power proportionality.

The parameters of the PID-controller need, as a conclusion, to match the workload pattern for the controller to be able to match the capacity of the system to the workload. Incorrect parameters will either result in poor QoS or unnecessary energy waste. The parameters need therefore a model from which the workload pattern is derived, or to dynamically change during run time. Based on these assumptions, our simulations show that the proportionality factor of a many-core system that uses a sleep state based power management is achievable.

VII. Future work

As concluded, future research is needed to determine if the system can reach power proportionality facing a general workload pattern. In order to adapt the system to such a pattern, the PID controller must adjust its control parameters during run-time. The run-time mechanism must therefore both analyze the previous workload pattern and anticipate the future workload pattern in order to make adjustment of the parameters. By recording the history of workload, CPU time, performance etc. the system could create certain models against which the parameter settings are set. After changes in the input variables occur, the models changes and thereby requires different controller settings.

Furthermore the power proportionality and energy reduction need to be compared to a system with both the current power manager and DVFS for each CPU core. The CPUs could thereby scale down their frequencies, and power dissipation (Table I) in accordance with the workload and thus increase the granularity of the system capacity further. Because scaling the frequencies is by orders of magnitude faster than switching on and off cores, the switching delay would not increase substantially.

References

[1] "Efficient servers, a project conducted within the eu-programme intelligent energy europe." [Online]. Available: http://www.efficient-server.eu

[2] B. Schäppi, F. Bellosa, B. Przywara, T. Bogner, S. Weeren, and A. Anglade, "Energy efficient servers in europe. energy consumption, saving potentials, market barriers and measures. part 1: Energy consumption and saving potentials," The Efficient Servers Consortium, Tech. Rep., November 2007.

[3] "Code of conduct on data centres energy efficiency, version 2.0," European Commission. Institute for Energy, Renewable Energies Unit, Tech. Rep., November 2009.

[4] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, pp. 33–37, December 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1339817.1339894

[5] L. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33 –37, 2007.

[6] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis," Vrije Universiteit, Amsterdam, The Netherlands, Tech. Rep. IR-CS-041, Sepember 2007 (revised: June 2008).

[7] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. H. Katz, "Napsac: design and implementation of a power-proportional web cluster," in *Proceedings of the first ACM SIGCOMM workshop on Green networking*, ser. Green Networking '10. New York, NY, USA: ACM, 2010, pp. 15–22. [Online]. Available: http://doi.acm.org/10.1145/1851290.1851294

[8] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," *SIGPLAN Not.*, vol. 44, pp. 205–216, March 2009. [Online]. Available: http://doi.acm.org/10.1145/1508284.1508269

[9] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*. Wiley and sons inc., 2004.

[10] K. Ab, "Kulturhuset," January 2011. [Online]. Available: http://kulturhuset.fi/start/

[11] Kulturhuset, "Request log november 2010 kulturhuset." [Online]. Available: https://research.it.abo.fi/projects/cloud/data/Request_log_kulturhuset_nov2010.zip

[12] J. T. J. Midgley, "Autobench," Xenoclast, May 2004. [Online]. Available: http://www.xenoclast.org/autobench/

[13] T. A. S. Foundation, "Apache," 2010. [Online]. Available: http://www.apache.org/

Paper II

A PID-Controlled Power Manager for Energy Efficient Web Clusters

Simon Holmbacka, Sébastien Lafond, Johan Lilius

Originally published *Proceedings of the International Conference on Cloud and Green Computing, 2011 IEEE International Conference*, pages 712–728. Sydney, Australia.

A PID-Controlled Power Manager for Energy Efficient Web Clusters

Simon Holmbacka, Sébastien Lafond, Johan Lilius
Department of Information Technologies, Åbo Akademi University
Joukahaisenkatu 3-5 FIN-20520 Turku
Email: firstname.lastname@abo.fi

***Abstract*—Large data centers using high-end processors operating continuously around the clock are major energy consumers. Long periods of idling due to low workload will cause a waste in energy because the processors are active but not doing any useful work.**

A cluster of low-end embedded processors could continuously match its computational capacity with the workload at a much finer granularity than a server-grade processor by changing the power states of the CPUs. This paper introduces a framework simulating a system level power manager for many-core clusters targeting server cards used in warehouse-sized data centers. The power management system uses sleep states to switch on or off processing elements in a cluster of low power boards to match the capacity of the whole system with the workload, and thus save energy. A PID-controller is implemented in the system; a component already well known with established methods in the industrial control domain. We intend to use this component to effectively determine the number of active processing elements in the used many-core cluster.

The proposed power manager can save up to 62 percent in energy compared to a system which only uses dynamic voltage and frequency scaling as power management.

***Keywords*-Power Management; Web Clusters; PID-controller; Low Power Processors;**

I. Introduction

Energy efficiency and physical size have become key issues for server cards used in warehouse-sized data centers. These factors do not only affect the operational costs and ecological footprint, but also have an impact on the possibilities to construct or expand data centers. With an average of 10 to 50 percent CPU utilization for servers [1] and the large load fluctuation found in typical web services [2], the use of slower but more energy-efficient cores could match the workload more efficiently with a much finer granularity and higher power proportionality [3] than server-grade cores.

A cluster of mobile processors can provide the same computational power as server-grade processors, but with a lower total energy consumption. Such a cluster can reduce the energy consumption efficiently by switching off elements according to the current need of service. For this purpose the cluster needs a power management on system level i.e. a component controlling the whole cluster as one entity.

This paper proposes a system level power manager for a cluster consisting of low-power nodes. The power manager uses sleep states to dynamically adjust the system capacity according to the workload. The monitored workload is matched so that minimal performance penalty and maximum reduction in energy consumption is obtained.

The PID-controller used in the industrial domain contains well established methods for obtaining stability and equilibrium in a dynamic system. We intend to exploit the theory of the PID-controller, and implement it into our power manager for matching the capacity of the system to the incoming workload.

Simulation parameters and workload data have been obtained by conducting experiments on real hardware, and by collecting statistics from a web space provider. The evaluated cluster is constructed of BeagleBoards [4] equipped with the ARM Cortex-A8 CPU. The chosen platform was selected based on its low price, energy efficiency and performance. By running several simulations on data samples containing 30 minutes of web statistics, we obtained a potential energy reduction of up to 62 percent compared to a similar system that only uses DVFS.

II. Related Work

The authors of [5] suggests a computational environment consisting of high-end Xeon servers combined with low-end mobile processors in order to achieve a fine granularity of system capacity in relation to the workload. All processing elements in the system uses sleep states to match the system capacity with the workload and thus reduce the energy consumption. Once the system recognizes an increase in workload, the system activates the processing elements in accordance with their different capacities and wake-up times. Four different control algorithms for adapting the capacity to the workload were presented and evaluated in the paper.

The authors in [6] also uses per-core sleep states to reduce the energy consumption in high-end server CPUs. The control algorithm used a simple high/low watermark on each CPU to decide which CPU core should be active. The obtained energy reduction for the system was claimed to be 40 % higher than a system with only DVFS available.

We intend to create fast, scalable and efficient capacity controller with control theoretic methods as basis. We argue that the use of the PID-controller could create a near optimal adaption of system capacity to the workload.

To determine the needed capacity of the system Bertini et. al [7] used tardiness for setting the needed performance by altering the CPU frequencies in a multi-tier cluster. Furthermore, the work in [8] presents an energy manager for server clusters based on a power model combined with a closed-loop controller to control the performance with sleep states and DVFS. In this work high-end CPUs were used evaluated with different energy policies and wake-up times were not considered.

Our sleep state-based system in contrast operates with a granularity of seconds and the wake-up time of cores highly influences the system. Our manager and architecture consist only of low-end embedded processors to give a distributed view of the system, and to adapt the manager to future many-core architectures. A combination of using sleep states to reduce energy consumption, the theory of the PID-controller to drive the capacity and the distributed architecture could decrease energy consumption without a substantial performance penalty.

III. Simulation Framework

A. Overview

The dominant consumer of energy on the aforementioned board is the CPU core [9], which our research focuses on. A simulation framework was created in Simulink to simulate and calculate the total energy reduction of the boards induced by using the power manager. The framework minimizes the total energy consumption by deactivating cores while maintaining the required QoS (Section III-B).

The basic structure of the framework is illustrated in Figure 1. The structure consists of a closed-loop system with an input, a PID-controller that controls the system capacity, and an output. The basic processing element in this paper is referred to as a *core*, since embedded systems with multi-core configurations have recently been available. The output of the system is used to determine the amount of cores needed to serve all requests.

Since a sleeping node will not be able to act as the power manager, all state changes will be based on decisions from a monitor node in the cluster i.e. the system level power manager. By running the manager on system level, decisions for power management will benefit the whole cluster instead of only a local node and thus reach closer to a global energy optimum.

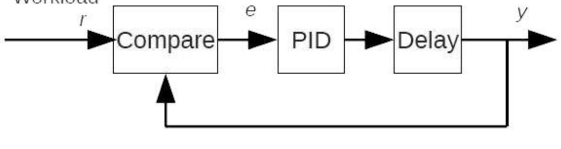


Figure 1. Basic structure of the power management system

B. Performance and quality

The simulation framework compares system capacity, i.e. how many requests the system can handle in a certain time, and the current workload. This comparison is taking place in the *Compare* block (Figure 1), which calculates difference between these two values. Incoming request are being spread out and processed in the web cluster in certain *time frames*. The granularity of the framework is therefore the length of one time frame.

QoS is a metric that is fully implementation dependent. The term describes how well an application is performing compared to its specification. QoS is usually used in soft real-time systems, in which the deadlines are set based on the human usability (or other subjective matters) of the system. Our system uses QoS to give a notion of latency of the request sent to the web service. A QoS drop occurs when a request in a certain time frame is not handled before the end of the time frame. This/these requests are then added to the next time frame and the QoS drops with a certain factor. Our definition of QoS states that as the workload exceeds the system capacity in a time frame, the QoS will drop. Eq. 1 shows the relation between QoS, capacity and workload.

$$QoS = \left(1 - \frac{W - C}{W}\right) \cdot 100 \qquad (1)$$

where W is the current workload and C is system capacity. The magnitude of the QoS drop is simply based on how many of the incoming requests were not handled in one time frame. The QoS is shown as a percentage. The maximum QoS value of 100 % means that the system provides the capacity to handle the whole workload in the measured time frame.

C. Switching delay

Our power manager works within the granularity of seconds. Since switching on cores is not instantaneous, the simulation must contain a delay for changing the CPU states. The algorithms in the PID-controller as well as measurements of the output signal also adds to the overhead of adapting the system capacity to the incoming workload. This overhead is represented in the simulation by inserting a delay block after the output of the PID-controller as shown in Figure 1. The delay can be adjusted in the simulation framework to represent different system configurations.

D. PID-controller

The PID-controller (Figure 1) is a common module in many control systems. It controls an output signal y depending on the input signal r and the controller settings. The difference between r and y is called the error value e, and is measured by using a feedback loop. The goal of the PID-controller is to minimize the error value and achieve equilibrium in the system.

The behavior of the PID-controller is determined by setting P, I and D values in the controller. These values

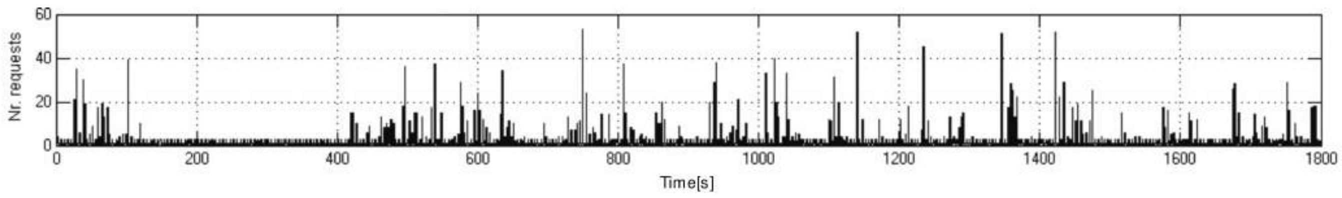


Figure 2. Workload sample from [10] 1. November 2010

choose how the output signal should react to changes in the input signal. The proportional part of the controller is set by the P parameter, which determines how fast or aggressive the controller reacts to changes in the input signal. I is the integral part of the controller. The main function of the integral value is to ensure that the process output agrees with the set point value in steady state. The derivative value D determines how the system reacts to changes in the reference value. By using a derivative term, the future of the reference curve is predicted based on previous values. The derivative term also enhances stability in the system [11].

The PID-controller is used in our power manager to select the amount of active CPU cores needed to process all requests in a time frame – this means that the PID-controller strives to activate only the minimum amount of cores and therefore minimize the energy consumption.

E. System capacity

The output of the system shown in Figure 1 is the current number of active CPU cores per time frame, which is determined as the output from the PID-controller.

Furthermore, the simulation framework supports the usage of *statically* active cores. These cores will be active and run on highest frequency completely independent of the control system. A high number of statically active cores allows the system to instantaneously being able to process the work between workload peaks. Workload peaks will decrease the QoS because of the delay the power management system introduces before it accommodates to the work peak. A high number of static cores will therefore slow down the QoS decrease during such a period, but will increase the average power dissipation of the cluster.

F. Final energy consumption

The simulation framework calculates the energy consumption for each time frame. The energy consumption is derived from the amount of active cores multiplied by the power dissipation of a core. We make the assumption that each core has two different states: *running* or *sleeping*. Since the board itself (with the CPU excluded) dissipates a small amount of energy, the power dissipation of the whole board is included in the output of one core for simplicity. The obtained power dissipation values were measured on the BeagleBoard with the DSP and the display subsystems disabled.

IV. Simulation Data

In order to simulate a realistic situation we conducted experiments to determine the parameters and settings for the simulation framework.

A. Web server requests

The web server requests used in the simulations were derived from [10] which is a Finnish web space provider. These http requests were addressed to over 750 websites and 510 domain names. By using data from an existing web space provider, we created a realistic situation for simulation. The workload curve pictured in Figure 2, relates to the number of http requests in a daytime sample from 1. November 2010. The curve shows, on average, a low workload with high peaks concentrated into certain time intervals. 30 minute samples were collected on the same date from the aforementioned server, and used as workload in the simulations. The data is freely available from [12].

B. PID parameters

As mentioned, the PID parameters determine how the controller reacts to changes in the input signal. This means that finding the appropriate parameters for the PID-controller is essential for having good regulation.

Several control methods for tuning PID parameters exist, and we will here focus on two common methods based on the frequency response of the closed loop system. Frequency response-based methods define the PID-parameters by determining the critical gain k_c in the closed-loop system. k_c is determined by increasing the controller gain until the output is on the border to instability, after which the period of the output signal t_c can be estimated. When these two parameters are determined, design recommendations are used to calculate the PID-parameters.

C. Ziegler-Nichols' frequency response-based recommendation

Ziegler-Nichols methods [11] were designed to give a good output response to load disturbances. This design recommendation is considered to give an aggressive controller with the risk of heavy overshoots, which means that our power manager will strive to quickly adjust the output resulting in fast reaction time and high overall QoS. Overshoots are a result of the control signal reaching over the desired set value to a certain amount before the controller stabilizes to the set value. This effect can cause slight energy waste because of unnecessary resource allocation.

Values for the PID-parameters, based on k_c and t_c can be obtain from Equation 2.

$$P = 0.6 \cdot k_c$$

$$I = \frac{1}{0.5 \cdot t_c} \tag{2}$$

$$D = 0.12 \cdot t_c$$

D. Åström-Hägglund's frequency response-based recommendation

The Åström-Hägglund method [13] also uses the parameters k_c and t_c obtained from the critical gain experiments to define the controller parameters. Furthermore, a constant κ has been defined through experiments and optimizations and is considered to give the system more robustness. κ is defined as :

$$\kappa = \frac{1}{K_p \cdot k_c} \tag{3}$$

where K_p is the process gain and k_c is the critical gain. This design suggests PID-parameters defined as:

$$P = (0.3 - 0.1 \cdot \kappa^4) \cdot k_c$$

$$I = (\frac{0.6}{1 + 2 \cdot \kappa})^{-1} \cdot t_c \tag{4}$$

$$D = \left(\frac{0.15(1-\kappa)}{1 - 0.95 \cdot \kappa}\right) \cdot t_c$$

The integral part in a PID-controller can cause problems when the input signal has great disturbances as the case shows in Figure 2. *Integral windup* is a phenomenon where the integral term accumulates a significant error during an overshoot. We have chosen to neglect the I-term completely to solve this problem. The nature of the power manager makes it possible to ignore the static control error that would otherwise have been eliminated with the I-term. This is due to the fact that the web cluster uses a discrete amount of cores and is not disturbed by a steady state value that is slightly off the set value. The implementation of the controller without an I-term will also be simpler with less calculation overhead. The result is actually a controller of PD-type, which is equal to a PID-controller with the I-term set to zero.

E. Static cores

The simulations were run with different configurations of static cores in order to measure the impact on the result. We used one to four static cores in different simulations. All four combinations were also simulated together with the different PID tuning methods to give a result on the energy and QoS relation between methods and static cores.

F. BeagleBoard power dissipation

To obtain values for the simulation framework and be able to run a proof-of-concept simulation, the power dissipation of one BeagleBoard revision C3 low-power platform was measured. The BeagleBoard is equipped with one ARM Cortex-A8 processor-based TI-OMAP3530 chip [4]. The system ran Ångström Linux kernel version 2.6.32 and was controlled through a remote serial console. The operating performance points (OPPs) of the TI-OMAP3530 chip were used to dynamically scale the clock frequency and voltage of the ARM subsystem. The values from this experiment will be used to simulate the energy reduction using DVFS as power manager compared to the proposed PID-controlled power manager and a system without power management. The OPPs were accessed through the Linux ACPI. To avoid unwanted energy consumption, the display and DSP subsystems of the TI-OMAP3530 were disabled. The BeagleBoard includes a resistor, which provides a way to measure the current consumption used by the board. The voltage drop across the resistor was measured for each OPP and the corresponding power was calculated. The obtained power values of the system running at respective voltage and clock frequency are displayed in Table I. To ensure that the load would remain constant during the measurements, the processor was stressed to 100 % utilization using a simple program that recursively counts Fibonacci numbers. Furthermore, the power dissipation of a board with a sleeping core was

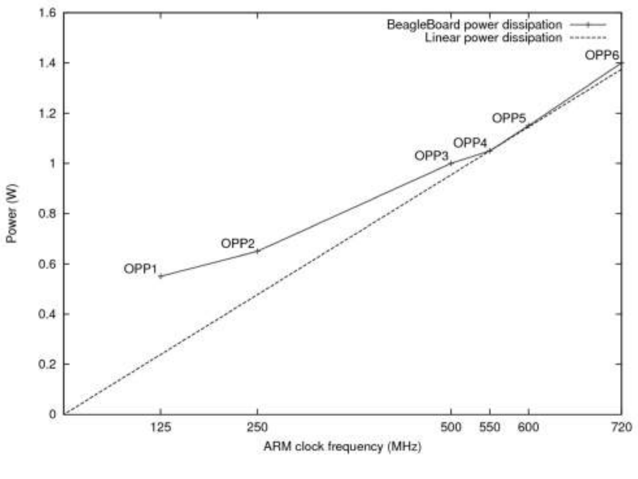


Figure 3. Non-linear power scaling by using DVFS

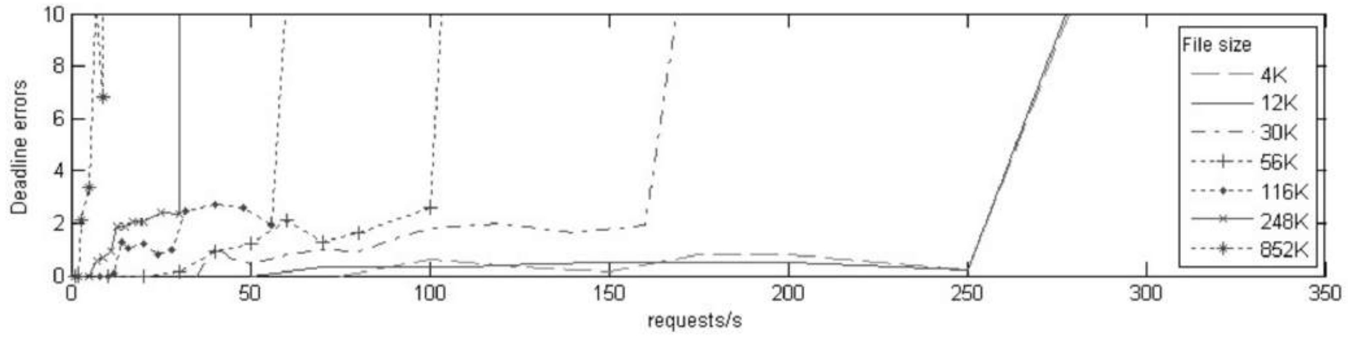


Figure 4. Capacity test for the BeagleBoard using Autobench

Table I
POWER DISSIPATION OF THE BEAGLEBOARD

Freq. [MHz]	720	600	550	500	250	125
Voltage [V]	1.35	1.35	1.27	1.20	1.06	0.985
Power fully [W] loaded	1.40	1.15	1.05	1.00	0.65	0.55

measured to dissipate 0.2 W. Detailed information of this experiment can be found in [14].

The Table I and Figure 3 clearly show that the power dissipation does not drop linearly according to the clock frequency. Therefore, we intend to explore the possibility of using sleep states instead of DVFS as power management.

G. BeagleBoard wake-up time

To measure the wake-up latency we configured the system as illustrated in Figure 5. The expansion pin 23 of the BeagleBoard was set to alternate between logic '1' and logic '0'. To initiate the wake up the system, the voltage on expansion pin 8 was set high. This will cause an interrupt that wakes up the system. The oscilloscope was connected to expansion pins 23 and 8. A transition from '0' to '1', i.e. the wake-up signal, on pin 8 was set to trigger the oscilloscope. The wake-up time for the BeagleBoard was on

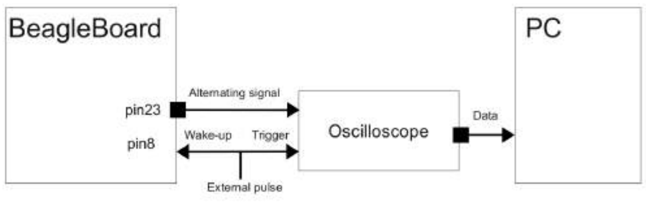


Figure 5. Schematic of wake-up tests

average measured to be 650 ms – with a standard deviation of 50 ms. Based on this measured wake-up time we set the transition delay in the simulation framework to 1000 ms to accommodate for overhead related to other eventual factors.

H. BeagleBoard load capacity

The system capacity is dependent on both the number of CPU cores in use and their capacity. We needed to determine the capacity of a BeagleBoard in order to run a realistic simulation.

Experiments were conducted on a BeagleBoard to give the number of requests per second a BeagleBoard could handle. The test tool in use was Autobench [15] which generates requests to an Apache server running on the BeagleBoard. The number of requests per second generated by Autobench started from a selected number and increased by specified increments. When the number of requests per second start to exceed the capacity of the host, deadline *errors* will start to occur as the selected deadlines for the requests are not met. The selected deadline in our experiments was one second, in order to match the time frame of the workload described in section III. Moreover a range of files each request needed to process was chosen and shown in Table II. The table also shows at what point deadline errors start to occur for the different file sizes. The file sizes were selected based on typical file sizes used in a web server.

The result of the experiment showed a certain error rate produced when the requests were not processed within the given time interval of one second.

Table II shows that a BeagleBoard in general can handle between 75 and 2 requests per second without errors, when using file sizes between 4 KB and 852 KB. A large file size such as 2.4 MB will produce large errors already after one request per second; this implies that experiments with larger file sizes are not needed. Figure 4 illustrates how the errors increase according to the increasing request rate. The curves represent the outcome of different file sizes.

Table II
MEASURED CAPACITY OF THE BEAGLEBOARD

File size [KB]	4	12	30	56	116	248	852	2400
Max [req/s]	75	50	35	20	10	5	2	0

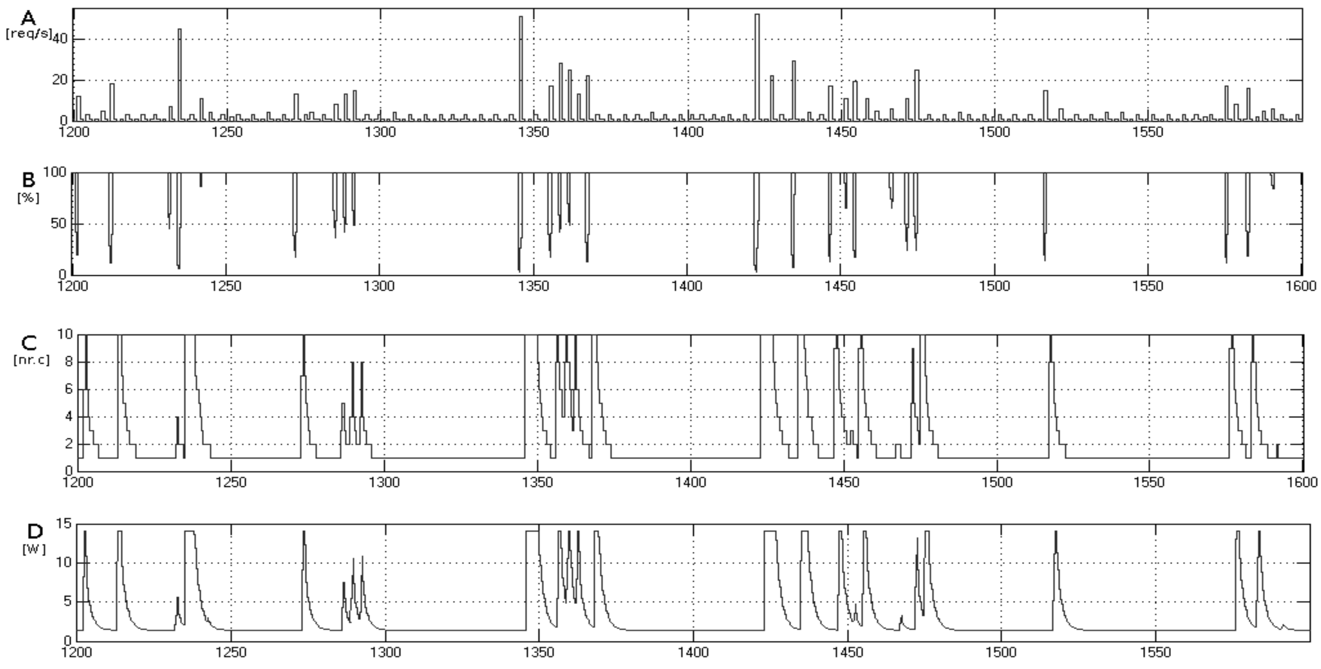


Figure 6. Results from simulation with Åström-Hägglund recommendations and 1 static core. A: Incoming requests, B: QoS value, C: Number of cores in use, D: Energy consumption

Related experiments in [5] result in a capacity of 5 requests per second for the BeagleBoard, which in our test setup would compare to a file size of 248 KB. The selected file size for our simulations was therefore chosen to be 248 KB. The file size is constant for each simulation with a maximum amount of 10 cores available.

Altering file sizes is a typical real-world scenario for web servers – this case is a general load balancing problem [16] and it has not been focused on in our simulations. Future research is needed to determine the impact of altering file sizes on the system.

V. Simulation Results

A. Comparisons

In order to draw a conclusion about the efficiency of our power management, our simulations should be compared to other power management systems.

Since existing systems implement power management such as DVFS, we also need to compare the final results with a 10 core system which is able to dynamically scale down its voltage and frequency. We created a framework for this purpose. Our simulations used the OPPs and the corresponding power dissipations presented in Table I. The simulation results show a typical 45 % energy reduction with DVFS enabled, compared to a system without power management. During a 30 minute run using a 10-core cluster, a system without power management would consume:

$$E_{full} = 10 \cdot 1.4W \cdot 30 \cdot 60s = 25200J \tag{5}$$

When enabling DVFS the energy consumption was reduced to $13558J$.

B. Our simulations

Given the values from the measurements we set-up the simulation framework as a 10-core system. The simulations were run for both tuning methods: *Ziegler-Nichols* and *Åström-Hägglund* – using the file size of 248 KB. This simulation was run four times, to use all combinations of static cores (1,2,3 and 4). The average QoS and total energy consumption was calculated and stored for all combinations of settings.

Figure 6 shows results from a simulation, which used the *Åström-Hägglund* method and one static core. The graphs in the figure are only showing the time interval [1200 1600]s for illustrative reasons. The graph in Figure 6(A), shows the incoming requests to the service. The corresponding QoS is presented as percentage in Figure 6(B), the number of currently active cores is shown in Figure 6(C) and the final power dissipation in Figure 6(D).

Table III shows the result for all simulations. The result consists of two important values: QoS and energy consumption. The values change depending on the used tuning method and amount of static cores. As seen in the table, both the QoS and energy consumption will steadily rise when switching on more static cores or using the more aggressive Ziegler-Nichol method.

Table IV compares the energy reduction between Ziegler-Nichols' and Åström-Hägglund's frequency response rec-

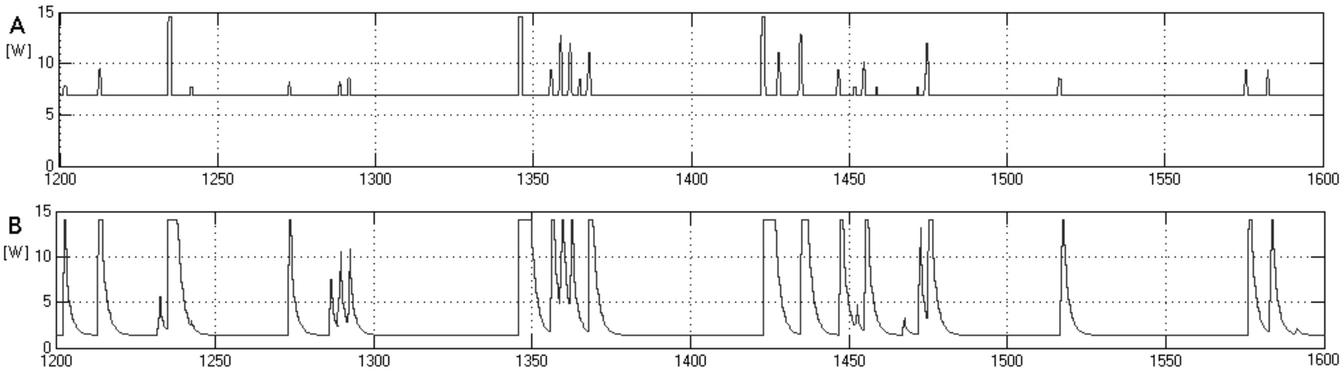


Figure 7. Energy graph comparing A: DVFS and B: sleep states

Table III
RESULTS FROM SIMULATIONS USING ZIEGLER-NICHOLS' AND ÅSTRÖM-HÄGGLUND'S FREQUENCY RESPONSE RECOMMENDATIONS. QOS [%] / ENERGY [J]

Static cores	Z-N	Å-H
1	96.8 / 6304	96.3 / 5190
2	98.1 / 7508	97.8 / 6746
3	98.9 / 9075	98.8 / 8612
4	99.2 / 11105	99.2 / 10787

ommendations for PID tuning. As expected, the more aggressive tuning method will result in less energy reduction, but as was seen in Table III the QoS was overall higher. By comparing the results with the reference value $13558J$ (obtained by only using DVFS), one can clearly state that a power management system that uses sleep state could reduce the energy consumption significantly compared to a system without a power management or to a system using DVFS as power management.

Table IV
TOTAL ENERGY REDUCTION FOR DIFFERENT CONTROL METHODS COMPARED TO A SYSTEM USING DVFS. THE TABLE SHOWS THE ENERGY SAVINGS IN %

Static cores	Z-N	Å-H
1	53.5%	61.7%
2	44.6%	50.2%
3	33.1%	36.5%
4	18.1%	20.4%

Figure 7 displays clearly the reason why sleep states will reduce the energy consumption more than DVFS. Part (A) in figure 7 shows the power dissipation for system in time range [1200 1600]s using DVFS as power management. In comparison, part (B) shows how sleep state-base power management allows the system to drop the power dissipation much more than DVFS, and therefore resulting in lower energy consumption.

Figure 8 shows a chart comparing the tuning methods to DVFS in terms of energy consumption and QoS. Naturally

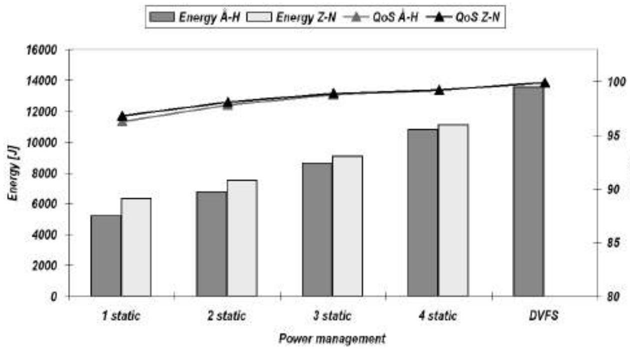


Figure 8. Energy chart comparing DVFS and sleep states

the sleep state-based power manager behaves more like DVFS when adding more static cores since more cores will then be constantly active. The most energy is saved when using the Åström-Hägglund method with one static core.

The lowest QoS will arise using a more conservative tuning method such as *Åström-Hägglund* and few static cores. Despite these constraints the simulations show that the QoS will only drop by approximately 4 % as seen in Table III. We assumed here that DVFS has 0 % drop in QoS. The table also shows that switching to a more aggressive tuning method (such as Ziegler-Nichols) and increasing the number of static cores will increase the QoS to over 99 % if requested.

VI. CONCLUSIONS

We have presented a power management system for many-core clusters. The power manager uses sleep states to match the capacity of the system with the workload to minimize the energy consumption while keeping the system performance on an acceptable level.

We have developed a simulation framework to evaluate the efficiency of the power management system. The framework reads a workload as input and shows, for each time frame, the results namely: number of cores in use, the QoS, and the energy consumption. Finally the framework shows the total energy consumption and the average QoS over the whole simulation.

The amount of active cores in the cluster is determined by a PID-controller that based on the well defined recommendations from Ziegler-Nichols and Åström-Hägglund methods drives the capacity of the system to just the necessary minimum.

Our simulation framework is based on parameters from the BeagleBoard equipped with an ARM Cortex-A8 processor. The capacity, power dissipation and wake-up time of the BeagleBoard was measured by experiments – this gives a realistic simulation and comparison of efficiency.

The results of our simulations show that our power management has the potential to reduce the energy consumption by 18 to 62 percent depending on the desired QoS, compared to a system that uses DVFS as power management. The reduction in energy arises from the fact that typical web servers have long idle times during which the full capacity of the system is not needed. Energy consumption can be reduced by replacing high-end server CPUs with clusters of low-end boards. The achieved granularity of low-end boards gives arise to extensive power management on system level which scales in a large data center.

VII. Future work

The power manager is currently being implemented on real hardware that uses the aforementioned CPU configuration. The real implementation will show the relation in energy consumption and an exact value of the introduced overhead of communication, algorithms etc.

We intend to further improve the power management for more intelligent control. Our simulations presented in [3] shows that static PID parameters is not sufficient to control all environments. Self-tuning regulators is described in [17] and could provide the necessary properties to adapt the PID-controller to heavily changing workload patterns.

Currently the framework does not support any tools to ensure a maximum latency for a request. By ensuring the maximum latency, a user can be guaranteed to receive a reply from the server in a certain time interval. PID-parameters can be influenced of this additional requirement and eventually self-tune to provide sufficient CPU power.

The file sizes used in the simulations have so far been constant. To compare the simulation to a further realistic case, the file sizes should change dynamically during the simulation according to an appropriate distribution function. The simulation framework can also be extended to explore situations of stochastic file sizes.

References

[1] L. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33 –37, dec 2007.

[2] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis," Vrije Universiteit, Amsterdam, The Netherlands, Tech. Rep. IR-CS-041, Sepember 2007 (revised: June 2008).

[3] S. Holmbacka, S. Lafond, and J. Lilius, "Power proportional characteristics of an energy manager for web clusters," in *Proceedings of the 11th International Conference on Embedded Computer Systems: Architectures Modeling and Simulation*. IEEE Press, July 2011.

[4] *OMAP35x Technical Reference Manual*, Texas Instruments Incorporated, July 2010.

[5] A. Krioukov and P. Mohan, "Napsac: Design and implementation of a power-proportional web cluster," in *Proceedings of the first ACM SIGCOMM workshop on Green networking*, ser. Green Networking '10. New York, NY, USA: ACM, 2010, pp. 15–22.

[6] J. Leverich and M. Monchiero, "Power management of datacenter workloads using per-core power gating," *IEEE Computer Architecture Letters*, vol. 8, pp. 48–51, 2009.

[7] L. Bertini, J. Leite, and D. Mosse, "Siso pidf controller in an energy-efficient multi-tier web server cluster for e-commerce," in *Second IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, Munich, Germany, June 2007.

[8] T. Horvath and K. Skadron, "Multi-mode energy management for multi-tier server clusters," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. New York, NY, USA: ACM, 2008, pp. 270–279.

[9] S. Madhavapeddy and B. Carlson, *OMAP 3 Architecture from Texas Instruments Opens new Horizons for Mobile Internet Devices*, Texas Instruments Incorporated, 2008.

[10] "Kulturhuset," http://kulturhuset.fi/start/, January 2011, [Online; accessed 31-May-2011].

[11] K. J. Åström and T. Hägglund, *Automatic tuning of PID controllers*. Instrument Society of America, 1988.

[12] Kulturhuset.fi, "Request log november 2010 kulturhuset," https://research.it.abo.fi/projects/cloud/data/Request_log_kulturhuset_nov2010.zip.

[13] K. J. Åström and T. Hägglund, *Advanced PID control*. Research Triangle Park, 2006.

[14] J. Smeds, "Evaluating power management capabilities of low-power cloud platforms," Master's thesis, Åbo Akademi University, Finland, 2010.

[15] J. Midgley, "Autobench," Xenoclast, May 2004. [Online]. Available: http://www.xenoclast.org/autobench/

[16] E. Musoll, "Hardware-based load balancing for massive multicore architectures implementing power gating," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 29, pp. 493–497, March 2010.

[17] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*. IEEE Press, 2004.

Paper III

A Task Migration Mechanism for Distributed Many-Core Operating Systems

Simon Holmbacka, Mohammad Fattah, Wictor Lund, Amir-Mohammad Rahmani, Sébastien Lafond, Johan Lilius

Originally published *The Journal of Supercomputing, 2014 Springer*, volume 68, Issue 3, pages 1141–1162

J Supercomput (2014) 68:1141–1162
DOI 10.1007/s11227-014-1144-7

A task migration mechanism for distributed many-core operating systems

Simon Holmbacka · Mohammad Fattah · Wictor Lund · Amir-Mohammad Rahmani · Sébastien Lafond · Johan Lilius

Published online: 8 March 2014

Abstract Spatial locality of task execution is becoming important in future hardware platforms since the number of cores is steadily increasing. The large amount of cores requires an intelligent power manager and the high chip and core density requires increased thermal awareness to avoid thermal hotspots on the chip. This paper presents a lightweight task migration mechanism explicitly for distributed operating systems running on many-core platforms. As the distributed OS runs one scheduler on each core, the tasks are migrated between OS kernels within the same shared memory platform. The benefits, such as performance and energy efficiency, of task migration are achieved by re-locating running tasks on the most appropriate cores and keeping the overhead of executing such a migration sufficiently low. We investigate the overhead of migrating tasks on a distributed OS running both on a bus-based platform and a many-core NoC—with these means of measures, we can predict the task migration overhead

S. Holmbacka (✉)
Turku Centre for Computer Science, Turku, Finland
e-mail: sholmbac@abo.fi

M. Fattah · A.-M. Rahmani
Department of Information Technologies, University of Turku, Turku, Finland
e-mail: mofana@utu.fi

A.-M. Rahmani
e-mail: amirah@utu.fi

W. Lund · S. Lafond · J. Lilius
Institution of Information Technologies, Åbo Akademi University, Turku, Finland
e-mail: wlund@abo.fi

S. Lafond
e-mail: slafond@abo.fi

J. Lilius
e-mail: jlilius@abo.fi

and pinpoint the emerging bottlenecks. With the presented task migration mechanism, we intend to improve the dynamism of power and performance characteristics in distributed many-core operating systems.

Keywords Task migration · Distributed operating systems · Many-core systems · Network-on-chip · Shared memory systems

1 Introduction

The notion of spatial resource locality is a measurement of the distance between executing tasks and their resources. This value reflects the communication delay introduced between the communicating tasks due to spatial separation. In a many-core network-on-chip (NoC) processor, this overhead is noticeable as messages need to propagate along the routing network of the chip. To minimize the communications overhead when using inter-core communication, the communicating tasks should be placed as close as possible to each other and possibly share caches. An optimal mapping of tasks can in a static system be done at compile time, but in a more general purpose PC with dynamic task creation, execution times, suspension, etc. the tasks should migrate on the chip during runtime to obtain the optimal locality.

System performance is usually improved by mapping tasks in parallel applications on multiple cores to improve the hardware utilization, since multiple processing elements are then capable of executing separate parts of the application in parallel. On the other hand, performance improvements are usually achieved with the sacrifice of energy. In contrast to parallelizing tasks, collecting them to only a few cores allows for sleep state based power management to shut down idle cores and enable a more energy efficient system.

Another important issue caused by the locality of task execution is the thermal balance inside the chip [1,2]. By changing the location of task execution on the chip, it is possible to avoid thermal hotspots which can gradually wear out the chip [3]. Work has previously been done in terms of task scheduling and heat distribution on the chip. An example is shown in [4] in which the mapping of tasks affects the thermal gradient of the CPU. The authors show a highly parallelized mapping in which the temperature is more evenly balanced, while concentrating tasks to only a few CPU cores forms a hotspot. Task mapping on many-core systems affects, therefore, the temperature and hotspots on the chip based on the spatial locations of the tasks. This effect will show even more clearly in 3D chips [5], since heat producing elements will spread out in three dimensions. Recently, there has been several work showing that the maximum thermal conduction usually takes place from the die which is closer to the heat sink [6–8]. This highlights the key roles of task migration (to move the computation) and packet routing (to move the communication) in mitigating hotspots and enhancing the thermal herding process.

In this paper, we present the implementation of a task migration mechanism using checkpoints for homogeneous many-core systems with shared memory. We contribute with an implemented task migration mechanism with a focus on:

- Shared memory task migration on distributed OSs and its memory mapping
- Context transfer and checkpointing on distributed OSs
- A comparison between utilizing task migration on bus-based and NoC-based systems
- A FreeRTOS Many-Core port for the ARM Cortex-A9 MPCore

2 Related work

Load balancing in conventional Linux [9] kernels on symmetric multi-processing (SMP) systems has existed for decades. The Linux completely fair scheduler (CFS) aims towards balancing the work as evenly as possible over all processing elements in the system [9]. The Linux load balancer inserts tasks into the run queue on a selected core while keeping all references to kernel resources unmodified. We have created a task migration mechanism for, in contrast to the monolithic single kernel Linux environment, a distributed OS [10–12] consisting of multiple kernels. The difference between our notion of task migration and conventional SMP load balancing is mainly the transfer of task context. Migrating a task between OS kernels requires more explicit context transfer, since the kernels are working independently of each other. Our task migration mechanism is implemented for many-core asymmetric multi-processing (AMP) OSs, which use time sharing scheduling only on local core level and explicit space sharing between OS kernels [10].

Many task migration techniques have been investigated [2,13–15], and the choice is usually dependent on what hardware configuration and what kind of OS is in use. Task migration between physically separate memories requires a transfer of both data and program memory area to the new memory location [16]. Heterogeneous task migration techniques have also been considered in previous work, in which the program code is modified to support the destination architecture [17]. This introduces several other challenges such as memory alignment, endianness and different instructions.

In contrast to previous works, our task migration mechanism is intended for a distributed OS with multiple kernels running on a homogeneous shared memory architecture with a MMU. Because of this architecture, only a pointer to the task handle needs to be physically moved. This means that in contrast to [13] and [2], the task size does not influence the migration overhead and no ISA translation is needed.

Different notions of task migration and strategies to initiate task migration have been previously presented [15,18–22]. Notably in [20], a replication mechanism is used to migrate tasks between cores in a multi-core system. As a task is created on one core, there is also a replica of the same task created on the other cores on which the task is migratable to. The replica tasks are suspended while the original is put in the running state. When migrating the task, the replica task receives its starting point and state and starts running exactly from the point at which the original task was suspended. A task migration mechanism based on re-creation was presented in [15]. The re-creation strategy involves creating a task only on the native core initially. During the execution of task migration, the task is completely copied to the other core's memory and started from the point at which it was suspended. After the migration is completed, the original task is suspended and deleted on the first core.

The migration of a task, in the first contribution of our work (bus-based platform), is based on only migrating the memory references a task is using. This migration strategy is possible since our target platforms include a shared memory between all cores. No replica of the task is needed, nor any transfer of program memory or the stack.

As the second contribution of this work, we also investigate the impact of our task migration mechanism on networks-on-chip-based platform. In this platform, accesses to the shared memory are handled through the network resources, while the memory is utilized by all nodes in the network. This will make a different traffic and memory access patterns compared to the bus-based platform. There have been several migration mechanisms for mesh-based and NoC-based multiprocessor System-on-Chips (MPSoCs). In [23], the authors proposed two migration schemes called Diagonal and Gathering–Routing–Scattering. The Diagonal scheme explores all disjoint paths to migrate a task in a single step. It operates based on XY dimension-ordered routing. They further expand their work by the Gathering–Routing–Scattering scheme.

Goh et al. [24] present two task migration techniques to rearrange active tasks in the mesh to form a larger contiguous area of free nodes for future mappings. Goodarzi et al. [25] propose a task migration mechanism using virtual point-to-point (VIP) connections. They tried to minimize the migration overhead using the Gathering–Routing–Scattering technique to migrate sub-meshes over VIP connections. The authors in [26] present a NoC-based MPSoC, using task migration to balance traffic at runtime. In this technique each processor may have one or a set of running tasks. They benefit from a real-time operating system on each processor for monitoring the deadlines as well as deciding on task migration. Moraes et al. propose a migration protocol to guarantee a correct task and context migration. Their technique provides in-order message delivery after the migration.

As it can be observed from the discussed NoC-based studies, the main focus is on the distributed memory based or non-uniform memory access (NUMA) architectures. In this paper, we concentrate on uniform shared memory access architectures. Task migration in this context has been rarely addressed in recent years. In this architecture, while migrating a task from a source node to a destination node, updating dirty cache lines in the main memory should be also taken into account. This is a situation which also happens for our bus-based platform and provides a fair comparison between the both platforms.

3 Evaluated platforms

For our work, we are considering a many-core platforms consisting of homogeneous CPU cores, shared memory and a MMU. These characteristics have been obtained in recent NoC-based many-core platforms [27–30], and is therefore a relevant choice according to the current hardware trend.

When using these kinds of platforms, the monolithic kernel architecture used in for example Linux starts to suffer from scalability problems [31]. The reason is mainly because inter-core locking of data structures in the kernel is required [32]. These locks are used to protect data structures from being accessed simultaneously by several cores, but become a bottleneck as the number of CPU cores increases. Linux uses, for example, a per-process kernel mutex which serializes calls to `mmap` and `munmap` [31]. This is because Linux was originally not intended to run on many-core platforms, and is thus not built with scalability in mind.

Instead, we focus on using a distributed operating system as our target platform. This OS structure has been adopted by several research operating systems such as the distributed/multikernel structure in Barrelfish [10], Corey [33] and fos [34] or the satellite kernels in Helios [11]. When using a multikernel OS, tasks can use core-local kernel calls instead of sharing one big kernel, which means that tasks issue kernel calls only (or mainly) to the core they are currently running on. No core-to-core communication or inter-core spinlocks are therefore required for kernel calls. We will map one small kernel with one independent scheduler on each core. The scheduler is time sharing tasks only on the local core with a real-time scheduler, and task migration is achieved by explicitly requesting a transfer of a task from one kernel to another.

Our task migration mechanism has been created explicitly for shared memory distributed operating systems. The platform is assumed to use shared memory for task stack allocation, dynamic memory allocation, program code and for inter-core message passing between tasks. The kernels can run either in shared memory or in private core memory.

We will use two different types of platforms for our experiments: one bus-based machine with a relatively predictable core-to-core overhead and one scalable NoC-based many-core platform.

3.1 Bus-based platform

The first used platform was the Versatile Express [35] board equipped with an ARM Cortex-A9 based CA9 NEC [36] CoreTile 9 × 4 quad-core chip running at 400 MHz with 1 GB of DDR2 main memory.

A multi-core port of FreeRTOS [37] for the ARM Cortex-A9 MPCore was created as a demonstration platform for the task migration mechanism and is freely available [38]. FreeRTOS is a small real-time kernel ported to many popular architectures. The kernel supports a real-time scheduler on top of which applications can be scheduled with hard real-time requirements. This RTOS was chosen due to its simplicity, small overhead and portability.

Figure 1 shows the structure of our system using one OS per core, a certain number of tasks and one task migration mechanism (TM) per core. The kernel of each OS is located, in this case, in separate parts of the shared memory and each instance schedules tasks only on the local core. This is an AMP OS view, which means that each OS instance (with scheduler) is running independent of the others and tasks on different cores do not share the same OS view as in the SMP case.

3.2 NoC-based platform

The Intel SCC [29] platform is our targeted network-based many-core platform. The SCC platform consists of 48 identical cores connecting together through a 6×4 2D

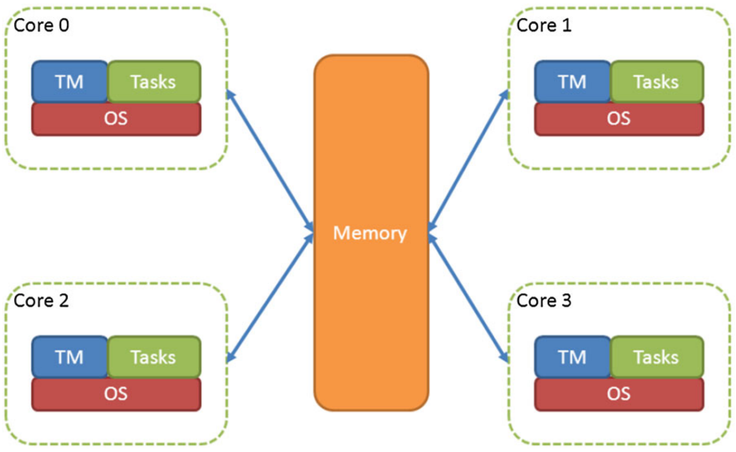


Fig. 1 Structure of the platform with shared memory for both OS kernels and core-to-core communication

fully synchronous NoC infrastructure with rigorous performance and power requirements. The 48 cores are placed in a tile formation with two cores per tile. Each core is clocked at 533 MHz and is connected to a 800 MHz network. The cores have two 16 kB of private L1 caches for data and instruction, and share a 256 kB L2 cache. The whole platform is connected to the main memory through 4 memory controllers (MC). The DDR3 main memory is clocked at 400 MHz.

Accordingly, we took the parameters from the SCC and modeled it with our in-house cycle-accurate SystemC many-core platform to have a close-to-realistic experiment environment. The simulation platform utilizes a pruned version of Noxim [39] as its communication architecture. With one core per tile, the constructed system is a 6 $\times$ 8 NoC-based platform. This configuration enhances the clarity and transparency of our analysis. The block diagram of the instantiated platform is shown in Fig. 2. Using the platform, each L1 cache miss results in replacing a cache line with the required line. If the required line is available in the shared L2 cache, the replacing line is sent to the target L2 part to get written out, after which the target L2 sends back the requested line to the requesting core. The L2 stalls for 18 clock cycles, which is equal to the SCC L2 reply time. In case loading the missed line from the main memory is required, the same scenario occurs by sending a cache line to the closest MC and fetching the requested line. The main memory stalls for 46 clock cycles according to its access time in the SCC platform.

According to the obtained parameters, the task migration methodology mentioned in the next section is modeled using the developed networked many-core platform. The migration occurs from a source core to a destination core while other cores are accessing the cache lines with respect to the caches miss rates.

4 Task migration methodology

The procedure to perform task migration on distributed-kernel operating systems is to safely suspend a selected task and transfer its state and references to another kernel on another CPU core. After the transfer, the task should be able to continue executing from the same point and with the same state it had before it was suspended. Moreover,

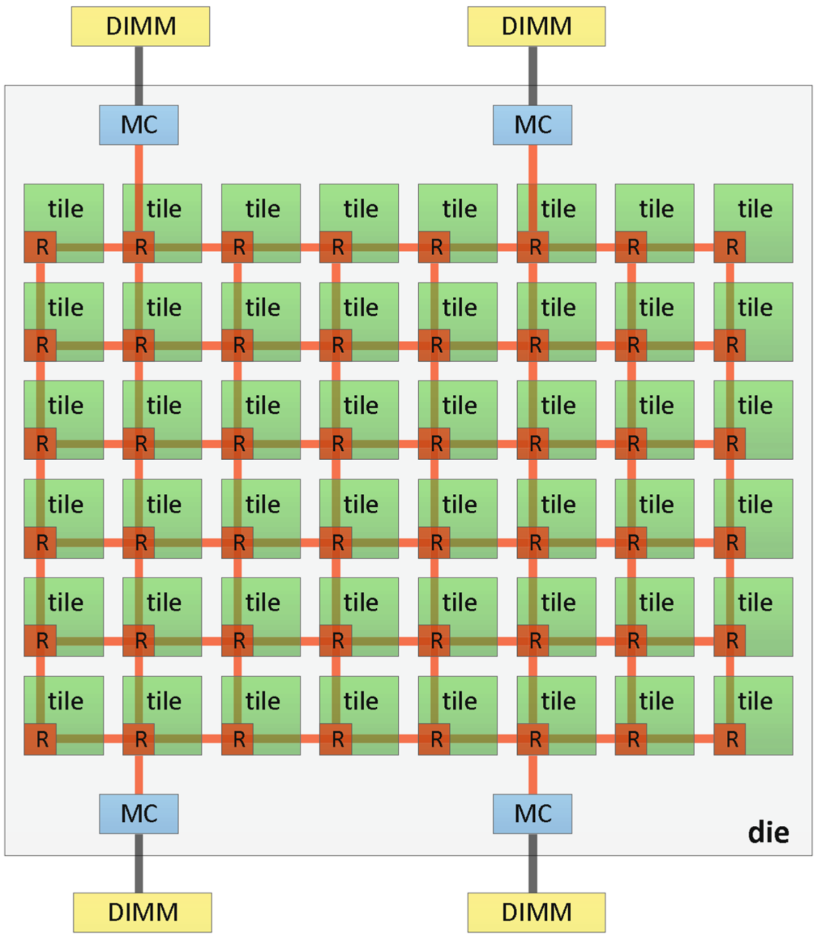


Fig. 2 NoC-based platform top-level architecture

the task should be attached to the task list of the target kernel and detached from the task list of source kernel. The task should also keep the same priority, name, stack pointer and stack size as it is transferred to the target kernel.

4.1 Notion of task migration

Because our notion of task migration covers migration between completely independent OS kernels, the task must be ensured a safe state to keep the notion of consistency [40] in case I/O, communication or kernel functionality is used. A safe state is a defined state in which the task is guaranteed to not be influenced by any external factors disturbing the transfer of the task state. Arbitrary transfer of a task might issue abrupt terminations of core-dependent resources such as I/O communication, which could lead to lost data or unwanted timeout errors. Any usage of resources, kernel functionality, intra-core communication or other non-preemptive functionality must, therefore, complete or safely be aborted before a migration can occur. Because of this uncertainty in computer programs, a checkpointing mechanism is used to depict points in the program at which is it safe to migrate the task. Checkpointing also decreases the complexity of the task migration mechanism, since all migrations are done at completely predictable points.

To make a program migratable, the programmer sets the checkpoints as the program is created. In our model, a checkpoint is set by a simple function call `TASK_IN_SAFE_STATE()`. This point is the dedicated place at which a task can migrate to another core.

The initiation of task migration is up to the system or another task, e.g., a power manager. Our task migration mechanism uses an observer task which recognizes scenarios for task migration. The observer can investigate the setup of other cores and make decisions where to move a task from which source core. Migration requests are signaled by a *request hook* in each migratable task, which sets a migration request flag initiated by the observer. A request hook is a special callback function inside the task (shown in Listing 1), which is callable from outside the task, e.g., the observer task.

Listing 1 Callback example

```
void TaskCallBack(){
    migration_request = 1;
}
```

The migration request flag is regularly checked by the task itself to reach the safe state if a request is issued. This procedure should be followed to achieve task migration in our model:

1. An observer actor in the system requests the migration of *Task 1* to *Core n*
2. Request hook is called in *Task 1* and the `migration_request` flag is set
3. *Task 1* checks the `migration_request` flag in the application, which now is set, and enters the safe state though the function `TASK_IN_SAFE_STATE()`
4. The task migrator is called and *Task 1* is migrated to *Core n*

Figure 3 illustrates the request flow with the same respective steps, in which the left part is the source core and the right part is the target core.

4.2 Use-case example

Since the checkpoints are placed by the programmer, the system should be analyzed beforehand to determine an eventual request lag. The request lag is the time between a migration request has been issued by the observer task until the task reaches the safe state. This time is minimized by placing checkpoints more frequently in the program. Polling the request flag uses only three instructions with the `-O3` flag on the Cortex-A9 CPU, but since a more frequent occurrence of checkpoints slightly increases the task overhead, the frequency of the checkpoint placement should be taken into account. Listing 2 shows a simple loop incrementing numbers and calling a function `foo()`. A checkpoint is set after each loop iteration, meaning that the task containing the loop can be migrated after each loop iteration. The task checks the `migration_request` flag at each loop iteration, which means that in worst case the migration lag is the time of one loop iteration. In case one loop iteration is very long, the programmer could consider adding checkpoints inside the loop itself to reduce the request lag. Still a

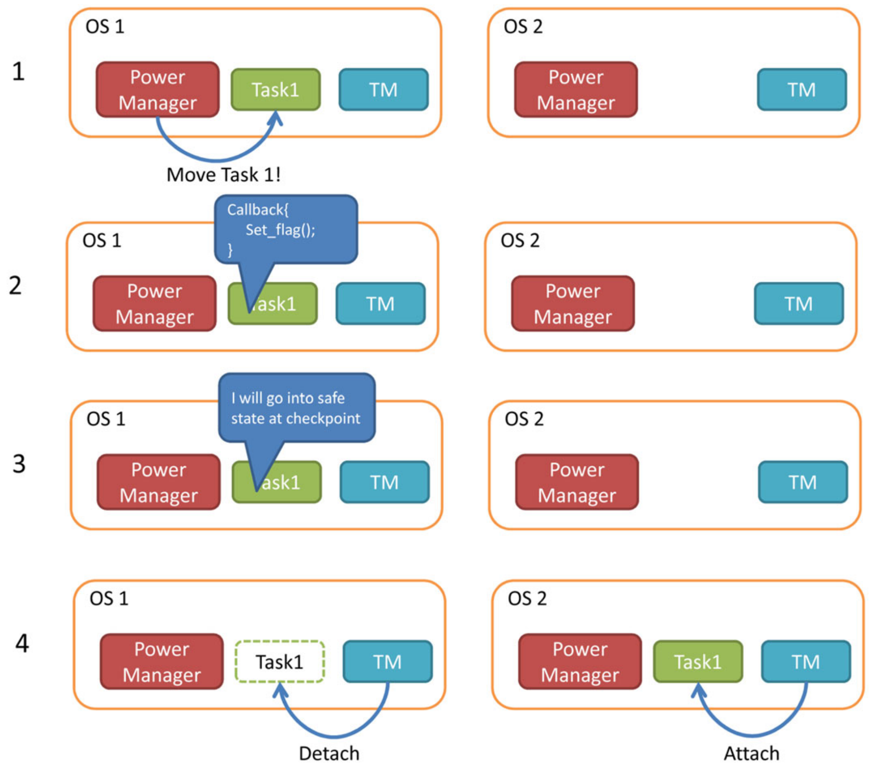


Fig. 3 Request flow in task migration. *Left part* source core, *right part* target core

more frequent use of checkpoints will affect the execution time and power efficiency of the system.

To find the optimal compromise between the request lag and the checkpoint overhead, measurements should be conducted. However, this question is very use-case dependent and is not part of this paper because of: (1) the introduced overhead is architecture dependent, (2) the introduced overhead is compiler dependent, (3) the trade-off between request lag and overhead is a subjective question and depends on which property is most desired from the application.

Listing 2 Checkpoint example

```
void looptask(){
   while (1) {
      for(i=0 ; i<MAX ; i++){
         foo();                      /*Call to function*/
         if (migration_requested)
            TASK_IN_SAFE_STATE();    /* Go into safe state and suspend*/
}   }   }
```

5 Implementation

The task migration mechanism has been implemented in the C-language specifically for FreeRTOS on 1300 lines of code in total. It consists of a migrator task mapped

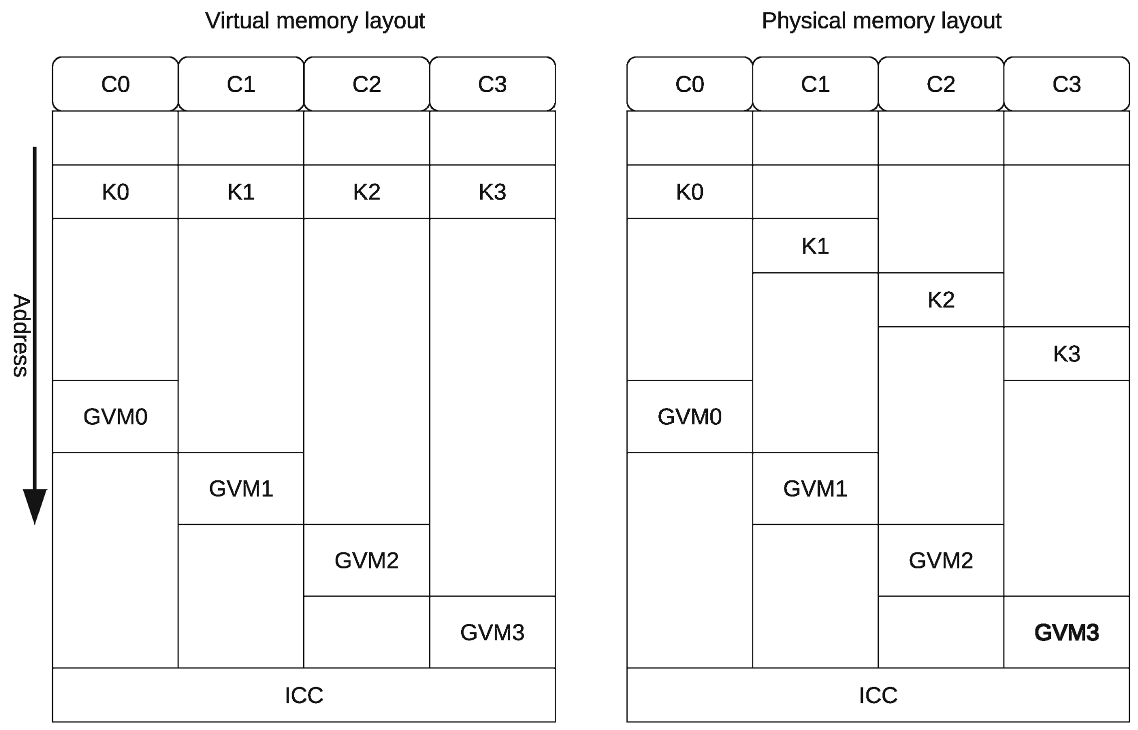


Fig. 4 Memory layout for a Quad-Core CPU

on each core, which handles the physical transfer and inter-core communication. The FreeRTOS kernel was modified to support the dynamic attachment and detachment of tasks from the task list while keeping the tasks' state consistent. The complete modification to FreeRTOS was implemented using 110 lines of C-code. This section describes the most important part of the implementation, namely how the memory is used between the kernels and how the task state is transferred across cores.

5.1 Virtual memory mapping

Virtual memory is used to replace the physical memory layout from the system, and replace it with a virtual representation which is easier to operate against.

In our model, each core C contains one kernel K. Each kernel uses the same virtual memory space, which means that each kernel has the same memory view; this is seen in the left part of Fig. 4. This mirrored view abstracts away the fact that the kernels actually execute in separate memory locations in the physical memory (right part of Fig. 4).

With this setup, all tasks can issue kernel calls with the same address of reference. This increases the OS scalability since no inter-core kernel locks are required, but tasks always call the local kernel. An example is later shown in Sect. 5.2.

The tasks running on a kernel are given a specific memory location: the Globally Visible Memory (GVM). This location depends on the location (in which kernel) the task is created. For example a task created on core $C1$ will allocate its stack space in $GVM1$. The GVMs are not, on the other hand, provided with the same virtual memory view. This is because tasks should be able to switch kernel to be scheduled on. The state of the task must always refer to the absolute memory address space to be kept consistent independent on what core the task is running on.

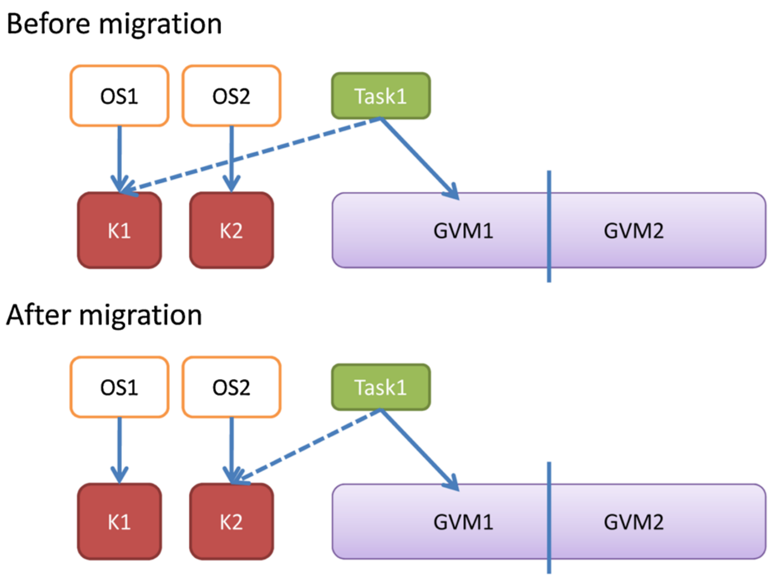


Fig. 5 Updating kernel references during task migration

For example, consider the case in Fig. 5: a task $T1$ with stack memory $GVM1$ migrating from $C1$ to $C2$. For the state to be kept consistent, $T1$ must still keep the references to $GVM1$ even though it is moved to $C2$.

If the stack pointer of $T1$ should now point to $GVM2$ instead, the content of the stack would not be kept consistent without physically moving the whole stack to $GVM2$. Since we assume a shared memory architecture, it is possible to only pass references to the corresponding GVM instead of a complete transfer. The migration overhead will also be much smaller since less information is moved.

After $T1$ has migrated from $C1$ to $C2$ the pointer to $GVM1$ is passed to $K2$, which updates its local task list. If $T1$ is deleted on $K2$, $K2$ sends a message to $K1$ to free the allocated memory $T1$ was using in $GVM1$. Generally, if a core issues a delete command on a task created on a non-local core, the delete request propagates back to the origin of the task to free the memory allocated by the task.

The memory reserved for core-to-core communication (ICC) is a statically allocated area in the highest part of the memory and is used to pass messages between cores. This part of the memory is set non-cacheable for the data to not rely on any write back from the local cache to the main memory. The core-to-core communication overhead will therefore have higher timing predictability.

Communication with shared variables is feasible and will not be affected by the task migration, since the address of the shared variable is located in the globally visible memory part and can thus be accessed by any task independently of what core the task is scheduled on.

5.2 Context transfer

The state transfer in a task migration is more complex than on Linux SMP systems since the task is moved to a different OS instance while keeping its state consistent. The state of a task is any entity stored as meta-data in the task which determines the execution of the task, and is during runtime modifiable. Besides the name and function pointer to the task itself, the following context is transferred during a task migration:

5.2.1 Stack state

The stack is initially created in a certain GVM depending on which core the task is created on. Upon task migration the location pointer to the stack is transferred to the target core. The stack itself is not physically moved since we assume that task stacks are located in the GVM.

5.2.2 Heap state

All dynamically allocated variables are stored in the heap. Similarly to the stack, the heap variables are stored in the GVM on the core the task using the variables was created on. As the task is migrated, all dynamically allocated variables pass their reference pointers to the new core, which means that no data are physically moved similarly to the stack state.

5.2.3 Function references

The motivation behind using a distributed kernel is to create a scalable OS for many-core architectures. An important functionality in this type of architecture is to enable core-local kernel calls. All tasks should therefore only use the local kernel for kernel calls.

Consider the system shown in Fig. 6: a task T1 is created on core C1 and uses kernel K1 for kernel calls. After the task migration to C2, T1 should update its kernel reference to K2 to use the core-local kernel calls.

To obtain this functionality, we have implemented re-linkable elf binaries for FreeRTOS. All tasks are compiled to distinct elf binaries and are linked together with the kernel on a core. During the task migration, the link between the task and the kernel is broken and re-linked with the kernel on the target core. The memory references to the kernel do not change, since the virtual memory ensures the same memory outlook of all kernels (as shown in Fig. 4). In this way, the tasks do not need to keep track on

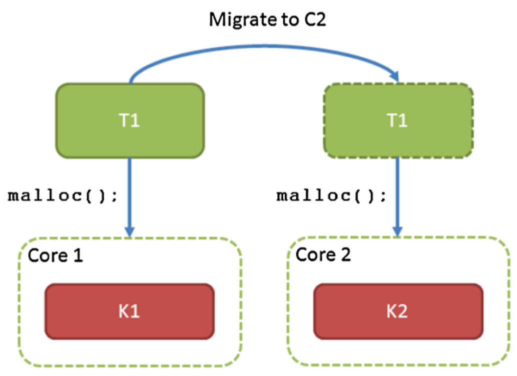


Fig. 6 Update of kernel reference after task migration

what core they are mapped on, which makes the programming completely location transparent for the programmer.

5.2.4 Inter-task communication

Tasks communicating with shared memory will retain the memory location used for the communication without any modifications. This is possible because all tasks can access any GVM at any time. The migrated task will, after the migration, keep the address to the shared memory at which the communication was taking place.

Communication with message passing between tasks is a part of future work. This functionality is non-trivial since the message passing mechanism is dependent on local or non-local communication. Non-local communication requires explicit core-to-core communication because the communicating tasks are located on different cores, while local communication should only use the message queue mechanism to not introduce unnecessary overhead. The solution to this problem is to create a multi-core extension to the FreeRTOS message passing mechanism that would be able to pass messages between tasks over a core-to-core channel.

6 Use-case evaluation

Our first evaluations are based on the ARM platform since the NoC platform is able to simulate the intended behavior but not execute all necessary functionalities. Later in Sect. 7, the overhead evaluation of task migration is measured for both platforms.

The evaluation setup consists of four identical video playing tasks mapped on the ARM quad-core platform described in Sect. 3. Each task plays a video with a certain resolution. The frame rate (fps) is measured with a regular interval to evaluate the performance of the video. We evaluated the system for both performance and power efficiency to demonstrate the improved dynamism of the system.

6.1 Performance evaluation

The first test was run to show how the performance of the video tasks is boosted by parallelization; namely by migrating tasks to all available cores. Our goal for this test is to obtain a stable video playback (25 fps) for all four videos. Initially, four large-resolution videos were mapped on Core0 on the ARM platform. After measuring a low frame rate, three of the video tasks were migrated to other cores, resulting in a system with one video task per core. The video task included one safe state point per frame, which resulted in 11 additional lines of source code in the application.

Figure 7 shows the execution of the test. At the beginning of the test ($t = [1, 3]$), all videos play with a frame rate between 6 and 14 fps, which is too low for user satisfaction. At $t = 3$, the first video task (Video1) migrated to another core, which results in an increased frame rate for Video1 at $t = 5$, and also a higher frame rate for the remaining video tasks on Core0. Similarly at $t = 6$, Video2 is migrated and thus achieves satisfactory frame rate at $t = 9$. Finally, Video3 migrates at $t = 13$ and is fully stable at $t = 14$. The high peak of the non-migrated task (Video4) is due to the

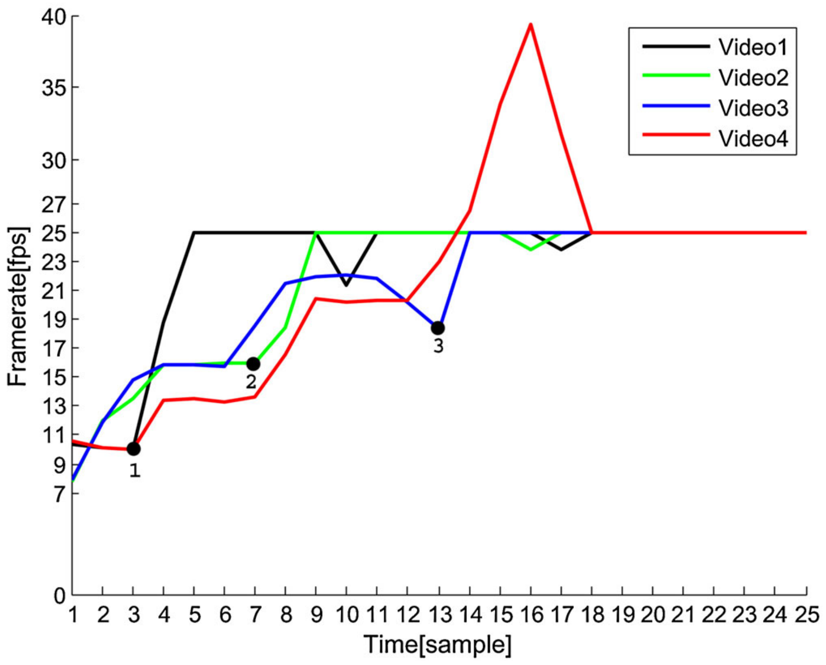


Fig. 7 Frame rates for large-resolution videos migrated to four cores. Task migration initiated at $t = 3$. Points 1, 2 and 3 show migration points in time

frame dropping mechanism used to compensate for low frame rate in the beginning of the test.

All videos are mapped on their own dedicated core at $t = 14$, and all videos tasks are executed completely in parallel. By parallelizing tasks and better utilizing the hardware, all videos increase the frame rates to 25 fps and keep a stable playback after $t = 18$.

6.2 Power evaluation

The second test was used to show the power efficient potentials of task migration. In this setup, we started by having four small- resolution videos mapped one on each core on the same quad-core ARM platform. The power output was measured directly from an internal register in the Cortex-A9 MPCore chip. Our goal with this test is to minimize the power dissipation as much as possible while keeping the frame rate stable. From the starting point of having four parallel videos, all video tasks were migrated to one core (Core0). The remaining idle cores (Core1, Core2, Core3) were shut down since no tasks were mapped on them after the migration.

Figure 8 shows the frame rate and related power output for the test. The curves related to videos are plotted against the frame rate axis and the power dissipation curve against the power axis. The test starts with having video playbacks with frame rates of 25.0 fps which is sufficient to the user, however the small resolution of the video format would allow collecting all videos to a single core. At $t = 7$ in Fig. 8, the migration mechanism starts to collect one video task at the time to Core0, and completes this operation at $t = 12$.

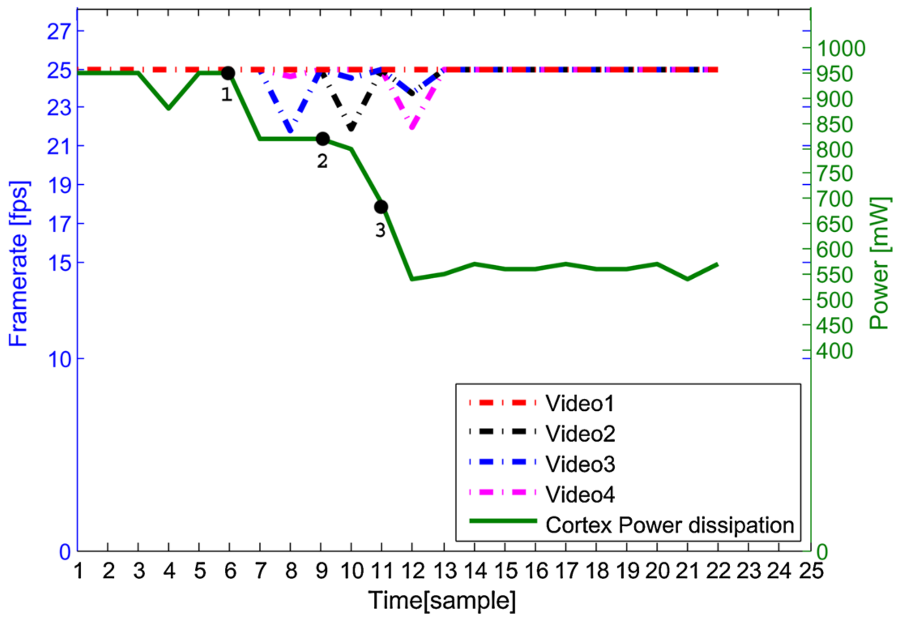


Fig. 8 Frame rates for small-resolution videos migrated to one core and Cortex-A9 power output. Task migration initiated at $t = 7$. Points 1, 2 and 3 shows migration points in time

The figure shows how the power dissipation initially starts at about 900 mW and decreases to roughly 550 mW after the tasks have migrated to Core0. This clearly affects the energy consumption of the platform since the power output is reduced by 40 %. Figure 8 also shows that Core0 alone is able to keep a sufficient frame rate of 25 fps for all four videos during normal playback. At the time of migration, the migrated video tasks occasionally measure a slight frame rate drop, but the overall quality is still kept sufficient.

7 Migration overhead evaluation

The migration of tasks naturally introduces an overhead due to the moving of data and detachment and attachment of tasks to/from the OS scheduler. A simple evaluation to measure the total overhead was run to demonstrate the feasibility of migrating a streaming task such as a video player without noticeable interruption. Our defined overhead of a task migration is measured as the time between *the suspension of the task on the sender core and the resume of the task on the target core*. This overhead was measured in three parts:

1. Time of task detachment and the activation of inter-core communication
2. Time for moving data over the inter-core channel
3. Time between the arrival of inter-core data and the attachment of the task

To have an as smoothly running application as possible, this overhead should be kept sufficiently small. We evaluated the overhead on both the bus-based ARM platform and on the NoC platform since the results should be very different: the completely symmetric ARM platform should introduce more or less a constant migration overhead while the communication delay on NoC platform is highly dependent on, e.g., the migration distance.

Table 1 Task migration overhead measurements on bus-based platform

	Part 1	Part 2	Part 3	Total
Time	17 ms	38 ms	45 ms	100 ms

7.1 Bus-based platform

The bus-based platform was the quad-core ARM Cortex-A9 interconnected with a shared memory for core-to-core communication. Part 1 and 3 were measured with a simple tic-toc timer which counts the elapsed time between two time instances using OS ticks as time unit, which is easily converted into Milli seconds. Part 2 (inter-core communication delay) was measured with a provided inter-core communications library for FreeRTOS. This library uses inter-core software interrupts to synchronize the time between two communicating inter-core channels, since the clocks of different cores are not identical. Measurements were run several times and the deviation of the results was zero for all three parts for a system with only one migrating task. The total overhead is presented in Table 1.

Table shows that Part 3, which consisted of attaching the task to the new OS scheduler, introduced the largest overhead when migrating one of our video tasks. A reason for this is due to L1 cache misses as the task is moved to a new core with a cold L1 cache. To reduce this overhead, the system could be set to warm up the cache lines before the task continues the execution or possibly enabling a shared L2 cache. Detaching the task (Part 1) had the least overhead with only 17 ms, and the physical data transfer 38 ms. The data transfer included a provided inter-core communications protocol binary, which enabled us to analyze clock synchronization, but overhead analysis of the internal mechanism was not possible due to closed source.

A fair comparison with related mechanisms is not directly straight forward since methodologies, platforms and the notion of task migration usually differ. For example, a NUMA architecture with non-uniform access time to memory would show a larger delay when migrating the task to a destination located further away. Moreover, with the introduced overhead, the video playback was—to the user—very smooth and the slight freeze during the task migration of a 25 fps video was hardly noticed. The task migration mechanism introduced in total 160 additional bytes to the application, which corresponded to 40 additional instructions to run (with gcc -O3 flag).

Single bus-based systems have, however, showed poor scalability because the single path of communication becomes a bottleneck when the core number increases. Not only core-to-core task migration will become slower because of the bus congestion, but also other data intense applications accessing the memory frequently. In case of distributed OS task migration, the likelihood of having an available bus decreases as the number of core increases. This would affect the overhead Part 2 in Table 1, since the physical data movement would be slower. Nevertheless, Part 1 and 3 are completely independent of the core-to-core communication and would not be affected by upscaling the system.

7.2 NoC-based platform

As mentioned before, we assume the SCC [29] platform with 48 cores to explore the migration overhead in a NoC-based shared memory system. Each core has its own private L1 data and instruction caches, while L2 cache is shared among the cores. In this subsection, we measure the networking time required in the migration process across different parameters. Since no physical OS is run on the NoC simulator, the additional overhead of OS functionalities are not included.

It is observed in [41] that 0.025 bytes of data are demanded in average by each instruction in the PARSEC benchmark (data access rate). Thus, a cache line is accessed every 1,280 instructions when assuming each cache line is 32 bytes on the SCC platform. Considering two threads running on each core, each core can issue around 1.5 instructions per clock cycle [41]. As a result, a cache line, in the system with the specified characteristics, will be accessed almost every 800 clock cycles by the cores. Note that the network clock frequency is 800 MHz and the cores are clocked to 533 MHz.

The scenario for a task migration is simulated as follows: (1) the L1 data cache of the source processor is cleaned. This means to write the contents of the cache out to the main memory and clear the dirty bit(s) in the cache. This makes the content of the cache and main memory coherent with each other. (2) The migration command is sent to the destination core by passing the memory reference of the migrating task to it. (3) The destination core will fetch the code and data of the migrating task and load them into its private L1 cache. The data volume to be written out and fetched is set equal to the SCC processors data cache sizes (16 kB) and we assume instruction codes to be 6 kB for each task. Note that the evaluated times are relevant to networking costs, memory and cache access latencies according to SCC characteristics. However, the overhead of the OS and local processors will be applied as an offset in real experiments as they are almost independent of the network performance.

In our first study, we explore the L1 miss rate effect on the migration time when no L2 cache is enabled in the system. In this case, all the L1 misses are handled through the main memory. Accordingly, upon a cache line miss, one of the lines is written out to the main memory, and the requested line is fetched back. The migration occurs while all the cores are issuing memory accesses constantly according to the defined miss rate. Figure 9 shows the elapsed time during the migration process with different L1 miss rates. Note that the source and destination are selected based on to the farthest physically separated nodes to the memory controllers.

As seen in Fig. 9, the migration time depends on the L1 cache miss rates. When the L2 cache is disabled, the memory controllers (the SCC contains four memory controllers) become the networking bottlenecks; i.e., all the missed cache lines are fetched from the main memory through the memory controllers. Note that in real application cases, the cache misses and data access rates are highly dependent on the tasks characteristics and is not identical for all processors.

In the second experiment, we enabled the L2 cache. The L2 cache is shared and distributed among all the processors. The address space is evenly distributed over L2 caches so the content related to a cache line miss in L1 can exist in any of the L2 cache parts of the system. Figure 9 also shows the migration time when L2 cache is enabled

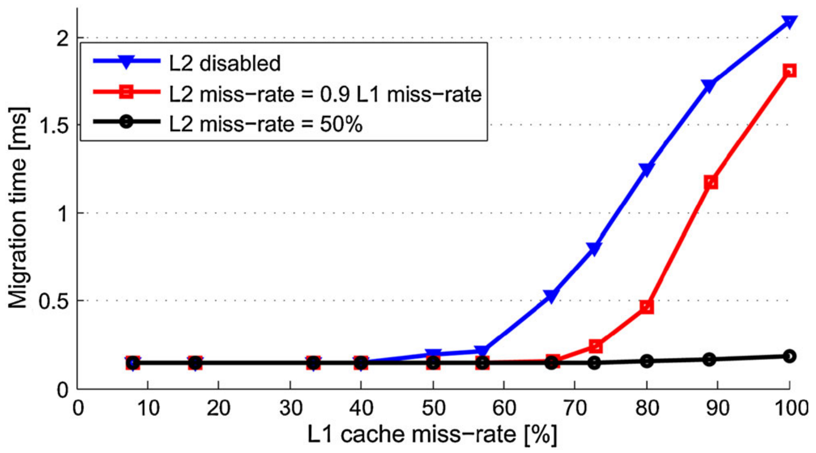


Fig. 9 Networking time in migration process without and with L2 cache enabled; with L2 cache miss rate set to 50 %

and 50 % of L1 misses can be fetched from it. By enabling the L2 cache, the migration time will significantly decrease since the L2 has a much shorter access time compared to main memory, and also releases the hot-spot pressure on the memory controllers.

Finally, Data intensive applications will affect both the L1 and L2 miss rates. We, therefore, assumed the L2 cache miss rate to follow L1 miss rate as: $\text{L2}_{miss-rate} = 0.9 \times \text{L1}_{miss-rate}$. Figure 9 also shows the migration latency in this case. As expected, the migration time will increase in data intensive applications. In case of extremely data intensive applications, in which L1 miss rate will go beyond 70 %, the system main memory will become the system bottleneck and increases the migration time dramatically. Note that the case will vary according to the number of cores accessing the main memory. This highlights the memory wall in the future shared memory many-core systems.

As shown above, the main memory will be the performance bottleneck in case of data intensive applications. Hence, the migration time will be high despite the source and destination distance to the memory controller. On the other hand, computation intensive applications make the network calm and keep the network latency small despite the communication distance. In other words, the distance between either cores or memory controllers-to-cores does not in practice influence the migration time.

This is observed in Fig. 10 when placing the source and destination in different distances from the memory controllers and running the migration process under various L1 miss rates. The extracted results in Fig. 10 show the source and destination settled in processors by Manhattan Distances (MD) of 1 to 5 from the memory controllers. The overhead is practically not affected by the migration distance.

8 Heterogeneous approach

The above experiments are done assuming a homogeneous system. However, differences in the obtained results are expected when considering a heterogeneous system, in which two types of heterogeneity impact on migration time:

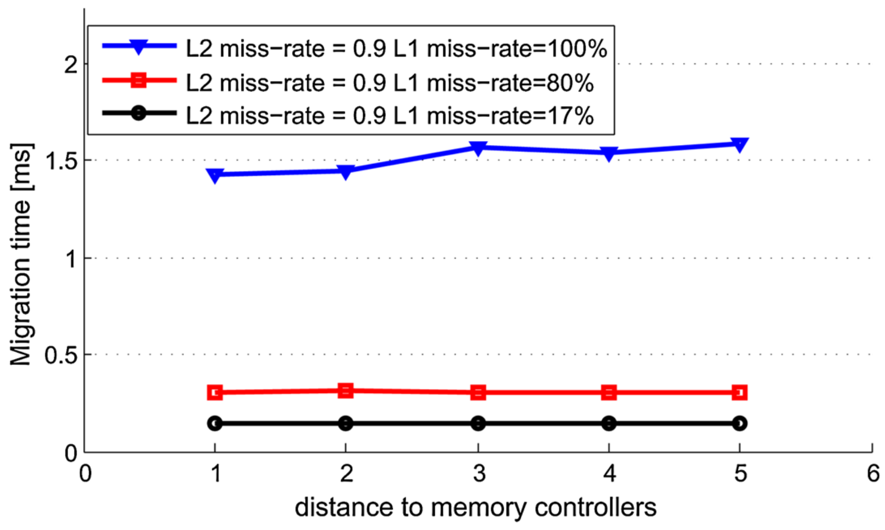


Fig. 10 The migration time is independent of source and destination distance from the memory controllers

First is the heterogeneous workload. As mentioned in the previous sections, all nodes are assumed to have identical traffic behavior; i.e., all nodes inject/consume the same amount of data into/from the network. Accordingly, a balanced load is offered to all network resources; such that the migration time is not influenced by source and destination placement as seen in Fig. 10. However, an asymmetric workload will offer an unbalanced load to each of the network resources as well as the MCs and caches [42]. Accordingly, a different migration time will be delivered by changing the source and destination node of migration process. Note that in this case, the difference is not derived from the distances to MCs, but from the heterogeneous traffic distribution over the network, which is out of the paper scope.

Secondly, the heterogeneity can be expressed by different types of core architectures in the nodes. The source and destination selection is therefore influenced by the node type, and the traffic distribution in the system is most likely unbalanced according to individual core performance. As a consequence, the source and destination selection will have an impact on the migration overhead and requires an independent study on for example the *fitness* of a workload on a certain core. Nevertheless, the same migration methodology can be used in heterogeneous systems, while a suitable mechanism for source and destination selection will be required.

9 Conclusions

This paper has described the implementation of a task migration mechanism for distributed many-core operating systems. We have presented the main points in building a task migration mechanism for shared memory platforms including memory mapping, state transfer and checkpointing. The task migration methodology was evaluated against both a traditional bus-based platform and a NoC-based platform to stress the diversity of future many-core platforms.

Several different characteristics can be obtained by managing the location of task execution such as communication latency, power and application throughput. We studied the overhead effects of task migration as the time needed to logically and physically move task references. The bus-based system demonstrated a constant overhead because of the light OS interference and the predictable latency to the main memory. The less predictable NoC platform showed, on the other hand, a varying migration time depending on the L1 cache misses as a result of interfering tasks running simultaneously. The task migration overhead is in this case largely dependent on (a) the L1 miss rate and (b) the presence of L2 cache and its availability. Migrating a task on shared memory platforms by simply transferring the task handle reference imposes, however, no significant overhead variations with migration distance. The chosen NoC layout and its characteristics limits the variations in task migration to memory accesses because of the network speed compared to the memory itself.

With more and more diverse and complex hardware, more intelligent software solutions are required to fully utilize the potential of the hardware. The distributed OS design together with task migration gives us the opportunity to tune the spatial execution location on large NoC platforms, and is a step towards this goal.

10 Future work

Tasks using inter-task communication need to update the references for passing messages if one of the communicating task changes its spatial location. Core-to-core communication must be explicitly pointed to tasks located on different cores, while tasks on the same core can use simple message queues.To hide these details from the programmer, we have developed a lightweight component framework for real-time systems [43]. With the framework, the developer is able to setup specific communication interfaces used for inter-task communication and could be used to rise the level of abstraction for the inter-task communication. We intend to integrate the possibility of task migration into the framework, which simplifies the updating of communication references.

To determine the physically timed overhead for the NoC platform, FreeRTOS should be ported to the architecture and be run on the real hardware with given use-case applications. While simulation gives results for the communication-based part of task migration, a real hardware experiment would add the inherited OS scheduling offset for task migration.

Acknowledgments This work has been supported by the Artemis JU project RECOMP: Reduced Certification Costs Using Trusted Multi-core Platforms (Grant Agreement Number 100202). The present work benefited from the input of William Davy, Wittenstein ltd., who provided valuable software and integration efforts to the research summarized here.

References

1. Cuesta D, Ayala J, Hidalgo J, Atienza D, Acquaviva A, Macii E (2010) Adaptive task migration policies for thermal control in mpsocs. In: Proceedings of the IEEE 2010 Annual Symposium on VLSI, vol 1. Ecole Polytechnique Fédérale de Lausanne and Politecnico di Torino
2. Mulas F, Atienza D (2009) Thermal balancing policy for multiprocessor stream computing platforms. IEEE Trans Comput Aided Des Integr Circuits Syst 28:1870–1882

3. Vaddina K, Rahmani A-M, Latif K, Liljeberg P, Plosila J (2011) Thermal analysis of job allocation and scheduling schemes for 3D stacked NoC's. In: Proceedings of the Euromicro conference on digital system design, pp 643–648
4. Musoll E (2010) Hardware-based load balancing for massive multicore architectures implementing power gating. IEEE Trans Comput Aided Des Integr Circuits Syst 29(3):493–497. doi:10.1109/TCAD.2009.2018863
5. Matsumoto K, Ibaraki S, Sato M, Sakuma K, Orii Y, Yamada F (2010) Investigations of cooling solutions for three-dimensional (3d) chip stacks. In: 26th Annual IEEE semiconductor thermal measurement and management symposium, SEMI-THERM 2010, pp 25–32. doi:10.1109/STHERM.2010.5444319
6. Rahmani A-M, Vaddina K, Latif K, Liljeberg P, Plosila J, Tenhunen H (2012) Design and management of high-performance, reliable and thermal-aware 3D networks-on-chip. IET Circuits Devices Syst 6(5):308–321
7. Rahmani A-M, Vaddina K, Latif K, Liljeberg P, Plosila J, Tenhunen H (2012) Generic monitoring and management infrastructure for 3D NoC-Bus hybrid architectures. In: Proceedings of the IEEE/ACM international symposium on networks on chip, pp 177–184
8. Vaddina K, Rahmani A-M, Latif K, Liljeberg P, Plosila J (2012) Thermal modeling and analysis of advanced 3D stacked structures. Procedia Eng 30:248–257
9. Jones MT Inside the linux scheduler, developerWorks. URL http://www.ibm.com/developerworks/library/l-completely-fair-scheduler/
10. Baumann A, Barham P (2009) The multikernel: a new os architecture for scalable multicore systems. In: Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles, SOSP '09. ACM, New York, pp 29–44
11. Nightingale EB, Hodson O, McIlroy R, Hawblitzel C, Hunt G (2009) Helios: heterogeneous multiprocessing with satellite kernels. In: Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles, SOSP '09, ACM, New York, NY, USA, pp 221–234. doi:10.1145/1629575.1629597
12. Boyd-Wickizer S, Chen H, Chen R, Mao Y, Kaashoek F, Morris R, Pesterev A, Stein L, Wu M, Dai Y, Zhang Y, Zhang Z (2008) Corey: an operating system for many cores. In: Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI'08, USENIX Association, Berkeley, CA, USA, pp 43–57
13. Engin TJE Bag distributed real-time operating system and task migration. Turkish J Elect Eng Comput Sci 9 (2)
14. Saraswat PK, Pop P, Madsen J (2009) Task migration for fault-tolerance in mixed-criticality embedded systems. SIGBED Rev 6(3):6:1–6:5. doi:10.1145/1851340.1851348
15. Bertozzi S, Acquaviva A, Bertozzi D, Poggiali A (2006) Supporting task migration in multi-processor systems-on-chip: a feasibility study. In: Proceedings of the conference on design, automation and test in Europe: Proceedings, 3001 Leuven, Belgium, pp 15–20
16. Armstrong JB (1995) Dynamic task migration from simd to spmd virtual machines. In: Proceedings of the 1st international conference on engineering of complex computer systems, ICECCS '95. IEEE Computer Society, Washington, DC, p 326
17. DeVuyst M, Venkat A, Tullsen DM (2012) Execution migration in a heterogeneous-isa chip multiprocessor. In: 17th International conference on architectural support for programming languages and operating systems (ASPLOS 2012). IEEE Computer Society, New York
18. Aguiar A, Filho SJ, dos Santos TG, Marcon C, Hessel F (2008) Architectural support for task migration conserning mpsoc. SBC
19. Acquaviva A, Alimonda A, Carta S, Pittau M Assessing task migration impact on embedded soft real-time streaming multimedia applications, EURASIP J Embed Syst (9)
20. Layouni LGS, Benkhelifa M, Verdier F, Chauvet S (2009) Multiprocessor task migration implementation in a reconfigurable platform. In: International conference on reconfigurable computing and FPGAs, 2009. doi:10.1109/ReConFig.37
21. Brio E, Barcelos D, Wagner F (2008) Dynamic task allocation strategies in mpsoc for soft real-time applications. In: Proceedings of the conference on design, automation and test in Europe. IEEE Council on Electronic Design Automation and EDAA : European Design Automation Association, ACM, New York, pp 1386–1389
22. Smith P, Hutchinson NC (1998) Heterogeneous process migration: the tui system. Softw Pract Exp 28(6):611–639

23. Chen T-S (2000) Task migration in 2D wormhole-routed mesh multicomputers. Inf Process Lett 73(3–4):103–110
24. Goh L, Veeravalli B (2008) Design and performance evaluation of combined first-flit task allocation and migration strategies in mesh multicomputer systems. Parallel Comput, pp 508–520
25. Goodarzi B, Sarbazi-Azad H (2011) Task migration in mesh NoCs over virtual point-to-point connections. In: Proceedings of the Euromicro international conference on parallel, distributed and network-based processing, pp 463–469
26. Almeida G, Varyani S, Busseuil R, Sassatelli G, Benoit P, Torres L, Carara E, Moraes F (2010) Evaluating the impact of task migration in multi-processor systems-on-chip. In: Proceedings of the symposium on Integrated circuits and system design, pp 73–78
27. Shao YS, Brooks D (2013) Energy characterization and instruction-level energy model of intel's xeon phi processor. In: 2013 IEEE international symposium on low power electronics and design (ISLPED), pp 389–394. doi:10.1109/ISLPED.2013.6629328
28. Potluri S, Tomko K, Bureddy D, Panda DK Intra-mic mpi communication using mvapich2: Early experience. Texas Advanced Computing Center (TACC)-Intel Highly Parallel Computing Symposium
29. Howard J, Dighe S, Hoskote Y, Vangal S (2010) A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In: Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp 108–109. doi:10.1109/ISSCC.2010.5434077
30. Wentzlaff D, Griffin P, Hoffmann H, Bao L, Edwards B, Ramey C, Mattina M, Miao C-C, JFB III, Agarwal A (2007) On-chip interconnection architecture of the tile processor. IEEE Micro 27:15–31. doi:10.1109/MM.2007.89
31. Boyd-Wickizer S, Clements AT, Mao Y, Pesterev A, Kaashoek MF, Morris R, Zeldovich N (2010) An analysis of linux scalability to many cores, in: Proceedings of the 9th USENIX conference on Operating systems design and implementation, OSDI'10, USENIX Association, Berkeley, CA, USA, pp 1–8
32. Kleen A (2009) Linux multi-core scaleability, in: Linux Kongress 2009, Dresden
33. Boyd-Wickizer S, Chen H, Chen R, Mao Y, Kaashoek F, Morris R, Pesterev A, Stein L, Wu M, Dai Y, Zhang Y, Zhang Z (2008) Corey: an operating system for many cores. In: Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI'08, USENIX Association, Berkeley, CA, USA, 2008, pp 43–57. http://portal.acm.org/citation.cfm?id=1855741.1855745
34. Wentzlaff D, Agarwal A (2009) Factored operating systems (fos): the case for a scalable operating system for multicores. SIGOPS Oper Syst Rev 43:76–85
35. ARM, Coretile express a9x4 technical reference manual, http://infocenter.arm.com/help/topic/com.arm.doc.dui0448e/DUI0448E_coretile_express_a9x4_trm.pdf (2011)
36. ARM, Cortex a9 technical reference manual, http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388e/DDI0388E_cortex_a9_r2p0_trm.pdf (2009)
37. Barry R (2009) FreeRTOS Reference Manual: API functions and Configuration Options. Real Time Engineers Ltd
38. Ågren D (2012) Freertos cortex-a9 mpcore port. https://github.com/ESLab/FreeRTOS---ARM-Cortex-A9-VersatileExpress-Quad-Core-port
39. Fazzino F, Palesi M, Patti D Noxim: Network-on-chip simulator, URL: http://sourceforge.net/projects/noxim
40. Banno F, Marletta D, Pappalardo G, Tramontana E (2010) Tackling consistency issues for runtime updating distributed systems. In: 2010 IEEE international symposium on parallel distributed processing, workshops and Phd forum (IPDPSW), pp 1–8. doi:10.1109/IPDPSW.2010.5470863
41. Bhadauria M, Weaver VM, McKee SA (2009) Understanding PARSEC performance on contemporary CMPs. In: Proceedings of the 2009 IEEE international symposium on workload characterization (IISWC), Washington, DC, USA, pp 98–107
42. Das R, Ausavarungnirun R, Mutlu O, Kumar A, Azimi M (2013) Application-to-core mapping policies to reduce memory system interference in multi-core systems. In: 2013 IEEE 19th international symposium on high performance computer architecture (HPCA2013), pp 107–118. doi:10.1109/HPCA.2013.6522311
43. Slotte R (2012) A lightweight rich-component framework for real-time embedded systems, Master's thesis, Åbo Akademi University

Paper IV

Thermal Influence on the Energy Efficiency of Workload Consolidation in Many-Core Architectures

Fredric Hällis, Simon Holmbacka, Wictor Lund, Robert Slotte, Sébastien Lafond, Johan Lilius

Originally published *Proceedings of the 24th Tyrrhenian International Workshop on Digital Communications, 2013 IEEE International Conference*, Genoa, Italy.

Thermal Influence on the Energy Efficiency of Workload Consolidation in Many-Core Architectures

Fredric Hällis, Simon Holmbacka, Wictor Lund, Robert Slotte, Sébastien Lafond, Johan Lilius
Department of Information Technologies, Åbo Akademi University
Joukahaisenkatu 3-5 FIN-20520 Turku
Email: firstname.lastname@abo.fi

***Abstract*—Webserver farms and datacenters currently use workload consolidation to match the dynamic workload with the available resources since switching off unused machines has been shown to save energy. The workload is placed on the active servers until the servers are saturated. The idea of workload consolidation can be brought also to chip level by the OS scheduler to pack as much workload to as few cores as possible in a many-core system. In this case all idle cores in the system are placed in a sleep state, and are woken up on-demand. Due to the relationship between static power dissipation and temperature, this paper investigates the thermal influence on the energy efficiency of chip level workload consolidation and its potential impact on the scheduling decisions. This work lay down the foundation for the development of a model for energy efficient OS scheduling for many-core processors taking into account external factors such as ambient and core level temperatures.**

I. Introduction

Energy efficiency is becoming a key issue in all types of computing systems, from hand-held devices to large scale distributed systems. The energy efficiency and proportionality characteristics can be studied on different levels, from the level of enterprise server farms and datacenters to the level of cores in a many-core processor. On the level of datacenters, a mismatch between the energy-efficiency region [1] and the typical processed workload is usually observed. This leads to a non-proportional relationship between the produced work and the corresponding energy consumption i.e. all energy is not used for useful work. Several approaches based on load consolidation [2], [3] were proposed to solve this issue and to accomplish better energy efficiency.

On the level of cores in a many-core processor, the current Advanced Configuration and Power Interface (ACPI) standard defines processor power states, called C-states or sleep states, and performance states called P-states. Using only P-states by exploiting dynamic voltage and frequency scaling (DVFS) mechanisms does not fully solve the power proportionality problem as an idling core still dissipates a non-negligible static power. The overall static power of a chip can be reduced by removing cores from the set of active cores and by using their C-states. Therefore, mapping portions of the workload to appropriate processing elements at any time and by using the processor and performance states influences the power dissipation of the processor. This mapping decision is usually done within the OS scheduler which can make scheduling and load balancing decisions leading to different operating states. As example, fairly distributing all tasks on all cores will disable the possibility to exploit any processor power states but will allow the use of performance states on all cores. On the other hand, consolidating all tasks on as few cores as possible will drastically limit the use of performance states on the remaining active core, but will take full advantage of the power states on the unused cores.

However a side effect of consolidating the workload to few cores is an increase of the temperature of the active cores, while the idle cores, on the other hand, remain cool. In an extreme case, because a hot core dissipates more static leakage power [4], a positive feedback effect of thermal runaway might lead to a continuously increase of temperature and static power dissipation over time. Moreover this factor tend to become more significant as the manufacturing technology decreases [4]. Although chip aging, calculation errors and thermal breakdown are also affected by the temperature, the scope of this paper is only focused on the relation to power dissipation. In this paper we will investigate the issue of energy efficient workload mapping in many-core systems and workload mapping guidelines will be given by investigating and considering:

- The spatial location of workload
- The ambient temperature conditions
- The processor temperature

All measurements and implementations have been obtained from a benchmark running on real hardware and using Linux 3.6.11 as the underlying software platform.

With the given insight into efficient workload mapping, this paper demonstrates that under certain conditions an OS scheduler should not only consider the inherited characteristics of the measured workload, but also account for external factors such as ambient and core level temperatures.

II. Related Work

Several strategies based on load consolidation to improve the system energy efficiency have been proposed in the literature. For example load consolidation is a well studied approach to reduce the energy consumption on the level of cellular access networks [5] and datacenters [2]. Improvements in energy efficiency based on the consolidation of virtual machines on a reduced number of physical servers, as shown by [6], has even resulted in commercially available implementations [7].

Also, on the level of many-core processors techniques to improve the energy efficiency have been proposed. The optimization problem of achieving a minimum energy consumption with the use of load consolidation and performance states on a many-core processor is mathematically formulated in [8]. Previous research has demonstrated the power saving potential of load consolidation in many-core systems [9]–[11]. For example, the behavior of the currently implemented power-awareness feature in the Linux scheduler is discussed in [11]. Without taking into account the thermal state of the cores, this paper demonstrates that the effectiveness of the power saving functionality in the Linux scheduler, described in more details in [12], on a many-core processor is workload dependent. In particular the current Linux scheduler is unable to consolidate load consisting of short running jobs and high rate of process creation and deletion. This paper indicates that the consolidation of tasks to keep as many cores as possible in long idle state is needed in order to reach the most optimal processor power state.

Indeed, idle states can be disturbed by needless interrupts, even when implementing an idle friendly interrupt scheme, such as the dynamic ticks in Linux [11]. To solve this issue, a form of interrupt migration framework to remove all needless interrupts from idling cores is needed [11]. Such a mechanism can be found as part of OS:es, such as the Linux hotplug functionality, which does not only remove interrupts but, depending on the underlying architecture, can shut down an entire core and remove it from the reach of the system scheduler. Migrating interrupts also cost time and it is important that the magnitude of this cost justifies the use of the mechanism [11]. However, since this functionality is capable of removing all activity on a core and placing it in a deep sleep state, it is argued that it can be used as a form of load consolidating power saving measure. By turning off cores at times of low load the remaining load would be consolidated over the remaining active cores.

It is often assumed that consolidating tasks onto fewer cores will result in a trade off between power and performance. However, load consolidation applied in conjunction with DVFS can, in some circumstances, also increase performance [10], [11]. Nevertheless, placing the CPU in a higher performance state, as a result of the load consolidation increasing the load over the active cores, the dynamic power of these cores as well as their heat production will increase, in turn increasing their static power consumption [13]. Since load consolidation can, depending on the load situation, a) either increase or degrade the performance, b) directly affect the dynamic power and c) indirectly, through thermal fluctuations, affect the static power, a prediction of its potential power saving is challenging. The situation is further complicated by the ambient temperature, since it also directly affects the system's leakage power.

Research has shown that load consolidation is in some cases a viable power saving technique on many-core platforms [14]. However, due to different variables such as load perception, interrupt handling, core and ambient temperature, DVFS behavior, and their impact on the overall energy consumption and performance, a best practice regarding load consolidation is yet to be found. This paper discusses the load consolidation challenges on many-core processor in order to discern scenarios in which load consolidation approaches are beneficial.

III. Power Dissipation and Energy Consumption in Microprocessors

A. Power breakdown

The total power dissipated by a processing element origins from two distinct sources: a) the dynamic power dissipation P_d due to the switching activities and b) the static power dissipation P_s mainly due to leakage currents. The dynamic power is dissipated when the load capacitance of the circuit gates is charged and discharged. Such activities occur when the CPU functional units are active. Because the dynamic power is proportional to the square of the supply voltage $P_d \sim Vdd^2$, much effort was put into the design of integrated circuits being able to operate at a low supply voltage. However decreasing the supply voltage of integrated circuits increases propagation delays which force the clock frequency down accordingly. Therefore by dynamically adjusting the clock frequency along the supply voltage when using performance states maximizes the power savings. The dynamic power is given in Eq. 1.

$$P_d = C \cdot f \cdot Vdd^2 \tag{1}$$

Where C is the circuit capacitance, f is the clock frequency and Vdd is the core voltage.

The static power is dissipated due to leakage current through transistors. Moreover, when lowering the supply voltage of integrated circuits, the threshold leakage current increases which also increases the dissipated static power [15], [16]. In addition to this, scaling down the technology process of integrated circuits increases the gate tunneling current which also leads to an increased static power [15]. Until recently, the power dissipated by a processing element was mainly consisting of the switching activities i.e. $P_d \gg P_s$ [17]. However due to technology scaling, the static power dissipation is exponentially increasing and starts to dominate the overall power consumption in microprocessors [4], [15], which leads to increased research efforts in minimizing static power e.g. with the use of sleep states.

B. Thermal influence

The temperature of a microprocessor directly influences the static power dissipation of the chip since the leakage current increases with increased temperature. The rate at which the static power is increased depends on the architecture and manufacturing techniques, and in this paper we mainly focus on mobile many-core processors. In order to determine the temperature-to-power ratio, we let a quad-core processor (ARM based Exynos 4) idle with no workload in different ambient temperatures.

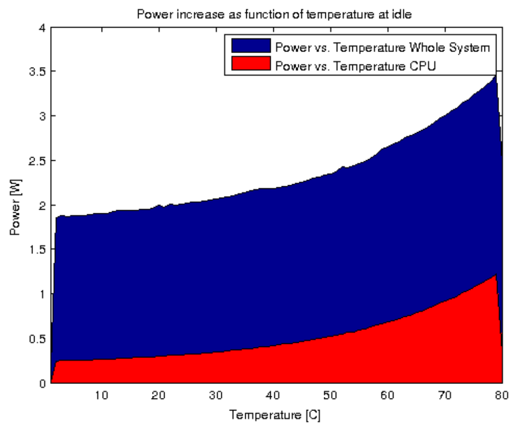


Fig. 1. Static power dissipation as function of ambient temperature for idling chip

Figure 1 shows the increase in static power as a function of the temperature for both board and CPU level measurements. At the left hand side of the curve, the chip was put in a freezer and its internal temperature was measured to be 1 °C, and afterwards it was placed in room temperature and heated up to 80 °C with an external heat source. As seen from the figure, the power dissipation of the chip increases more than twofold depending on the ambient temperature conditions. The sudden drop in chip power at the 80 °C point is due to the chip's frequency throttling mechanism, which is automatically activated at this point in order to prevent overheating leading to physical and functional damage.

C. Energy consumption

The amount of energy consumed by a processor is the product of the processors power P_{tot} and the time t as shown in Eq.2

$$E = P_{tot} \cdot t \tag{2}$$

where P_{tot} is the sum of dynamic and static power $P_{tot} = P_d + P_s$. The linear combination of power and time results in a two-variable optimization problem for minimizing the energy consumption. A strategy called race-to-idle [14] primarily used in handheld devices aims to execute work as fast as possible in order to minimize the execution time and save energy. On the other hand, decreasing the clock frequency and supply voltage at the cost of longer execution time might reduce the total energy if the processing elements are not overheated due to hot ambient temperature.

We will therefore investigate the energy consumption of different workload placement policies with respect to both power and execution time.

IV. Workload Mapping Policies and Energy Consumption

The main goal of this paper is to investigate how different workload placement policies affect the dissipated power and execution time and how it impacts the energy consumption. Practically the workload on an OS is defined in a certain work quanta called *process* or *task*. A task executing on a core may *utilize* or *load* a certain percentage of the core's capacity. The methods

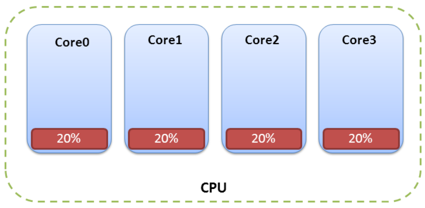


Fig. 2. Workload placement for the evenly balanced policy

for calculating load varies but are usually defined as the ratio between core execution and core idling over a certain time window.

Figure 2 shows the traditional method of task mapping in Linux [18]. Four theoretical CPU-bound tasks, each imposing 20% load, spread out evenly over a quad core CPU to create a balanced schedule.

The complete opposite mapping policy would be to pack as much work onto as few cores as possible.

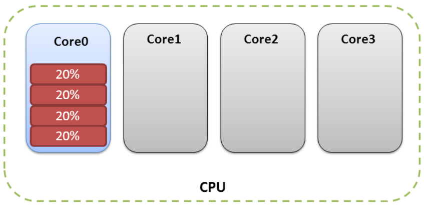


Fig. 3. Workload placement for the packing policy

Figure 3 illustrates a scenario in which four tasks have been mapped on only one core and the remaining cores are turned off. In this case the system must insert the notion of *overloading* a core because as soon as the loaded core is overloaded, a new core must be woken up to offload the overloaded core. In a non-ideal (and more realistic) case the workload is not ideally divisible over the complete platform.

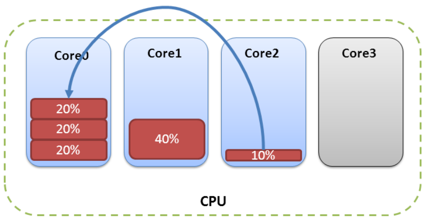


Fig. 4. Workload migration in a non-ideal workload case

Figure 4 shows how the core with the least workload offloads a task to the most loaded (but not overloaded) core, in this case Core 0. Similarly, if a core becomes overloaded, the core offloads a sufficiently large portion of the workload to the most loaded but not overloaded core.

The key issue for these different mapping policies is the power breakdown and thermal gradient of the chip. Consider a case with four tasks, each utilizing a core running at 400 MHz to 100%, and a chip implementing P- and C-states. Figure 5 illustrates the relative differences in terms of static and dynamic power dissipation for the balanced policy (leftmost part) and the packing policy (rightmost part). As for the balanced strategy, every core (in this case a quad-core) dissipates a small amount of dynamic power since the clock is only running at 400 MHz and still the core is not overloaded. At the same time, the cores also dissipate static power since all cores must be enabled to process the workload.

The same set of tasks using the packing policy is shown in Figure 5, (rightmost part). Since all four tasks

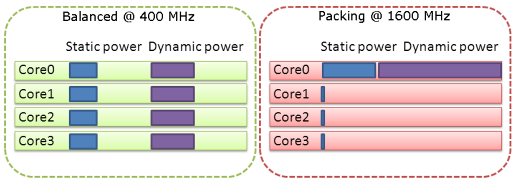


Fig. 5. Power distribution of two workload placement policies

in this case are mapped on only one core, the core must quadruple its frequency in order to not become overloaded. As the frequency increases, the dynamic power increases due to the frequency factor f and the voltage factor Vdd^2 as was shown in Eq. 1. Furthermore, when frequency and supply voltage increase the thermal dissipation also increases to form a thermal hotspot. This increases the static power dissipation because of increased leakage currents in the semiconductors.

While the packing policy results in high power dissipation for the busy core, the idle cores can be shut off and their total power dissipation is in best case zero. The following sections present a set of benchmarks to determine the most energy efficient workload mapping policy with different workload scenarios in different ambient temperatures.

V. Experiment Setup

A. Hardware platform

The hardware platform used for the experiments was an Odroid-X board equipped with a Quad-Core Exynos 4 implementation of the ARM Cortex-A9 architecture. The CPU had a maximum clock frequency of 1.6 GHz and 1 GB of DRAM. The board has 15 P-states corresponding to 15 different clock frequencies and voltage settings. The highest P-state corresponds to a frequency of 200 MHz and each P-state step changes the frequency by 100 MHz. We ran each experiment in three ambient temperature conditions: 1) hot temperature (the board was using only a passive heatsink), 2) normal temperature (an external fan was used), 3) cold temperature (the board was put in a freezer at -20°C). The power was measured with a probe attached to the current feed pins on the ARM cores and the temperature was measured by reading internal registers.

B. Software platform

Linux 3.6.11 was chosen as the underlying software platform because of the possibilities to alter the task mapping with simple scheduling tweaks and already implemented CPU hotplugging capabilities. All comparisons were run with two policies a) spread out the workload as evenly as possible, and b) packing the workload to as few cores as possible without overloading them. The used platform did have capabilities for DVFS tweaking and CPU hotplugging. The used DVFS governor was the default Linux OnDemand, which sets the clock frequency to the appropriate level depending on the measured workload by utilizing the P-states. All OnDemand parameters were constant in all sets of experiments.

C. Spurg Bench

Spurg-Bench [19] is used to generate controlled load levels on many-core processors. This benchmark is designed to test a system with different types of load levels. It consists of a load generator and a runner script able to generate one or more single-threaded load operation schedulable to any core in the system. Between each portion of operations, the benchmark idles for a set amount of time to create the desired workload percentage. This means that the test is able to generate a specific amount of operations to calculate by utilizing the CPU to a certain level.

The chosen operation we have used stresses the CPU and the CPU's floating point unit with floating-point multiplications. The C code for the operation is shown in Listing 1.

```
int operation(){
  int i; double a = 2.0;
  for (i = 0; i < 1000; i++)
    a *= 2.0;
  return 0;
}
```

Listing 1. Example of a operation using the processors floating-point units.

D. Results

Spurg-Bench was initially set to execute 100000 operations for a certain set of load level setpoints: [10, 20, 30, 40, 50, 60, 70, 80, 90]. All the tests were run in the three different ambient temperature conditions with both the consolidation policy and the balanced policy.

Figure 6 presents the performance, as the number of operations per second, per watt of CPU power, during each of the different load level set points for both scheduling policies in three different ambient temperatures. The figure shows that during lower load levels more work can be accomplished, for the same power budget of one watt, by consolidating the workload.

In situations when the load is low, consolidating does not increase dynamic power nor the temperature, and in extension the static power of the remaining cores considerably. This leads to a situation where the reduction in static power is larger than the increase in total power over the remaining cores, resulting in power savings. Furthermore, since the static power consumption is higher at higher temperatures the effect is more discernible in the high temperature case.

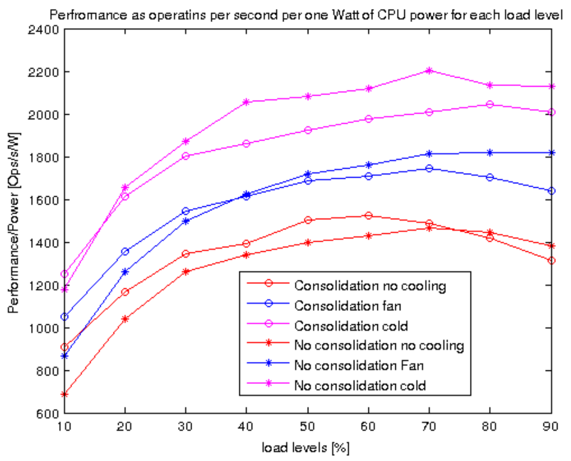


Fig. 6. Power of CPU as a function of operations per second for different ambient temperatures

From the figure it can be seen that as the load increases the performance per watt, for the consolidation policy, degrades in relation to the non consolidation policy. This results in crossover points between the performance per power of the two different policies. These crossover points depict the point at which the trade off between power and performance, when utilizing consolidation, is no longer energy efficient. This behaviour is due to the inherent trade-off between power and performance brought upon by consolidation. Even though the average power drawn by consolidation is lower, at higher loads it takes longer to complete the tasks resulting in decreased performance per watt. The decrease in performance is also partly due to the consolidation policy itself not being as fast at rearranging tasks as the default scheduling policy.

From Figure 6 we notice that the crossover point for the studied CPU with no cooling fan is around load level 70% at 1440 operations per second per watt, and at load level 40% with 1600 operations per seconds per watt if the CPU is cooled down by a fan. In case the CPU is in a freezing environment the crossover point already happens around load level 20% at 1500 operations per second per watt. The different placement of the crossover points are due the static power consumption per core being higher at higher temperatures, which effects the power saving effect of consolidation.

Even though the gains of using load consolidation on the chip level where considerably less than expected, when compared to similar techniques used on the server level [6] [7], the results indicate that consolidation on the chip level can, only in some cases, prove to be a valid measure to improve energy efficiency. Although we expect comparable behaviour on similar types of hardware, we intend to extend our analyses to other architectures as future work. The evaluation of architectures having hardware controlled P- and C- states might produce different results.

VI. Conclusion

This paper has investigated the performance and energy efficiency of a fairly distributed scheduling policy compared to a workload consolidation policy in different ambient temperature conditions. The aim of this work was to determine if under different temperature conditions the static power saved by shutting off idle cores weighs up against the increased dynamic power obtained when increasing the clock frequency and consolidating the workload on the remaining active cores.

The results show that as the workload is fairly distributed over the whole chip, the power and thermal dissipation of the chip remains rather proportional and predictable. However, this is not always the most energy efficient way of scheduling tasks in a many-core system; for low workloads consolidation provides a more energy efficient scheduling policy as the unused cores are completely shut down. Due to the power dissipation of the CPU increasing exponentially as a function of the temperature, the effects of consolidation on energy efficiency is more prominent during higher temperatures. On the other hand, the improved energy efficiency of consolidation degrades as the load increases. Reaching a point where it is more energy efficient to utilize all cores in the system at a slightly higher power dissipation.

We have found that it is difficult to apply a general scheduling policy suitable for any environment – as ambient and chip temperature conditions change, the energy efficiency of scheduling policies varies. Consequently, scheduling decisions should not only be based on internal workload measurement, but should also integrate external conditions such as ambient temperature. On the studied platform, the energy efficiency can be improved by adding a temperature sensitive consolidation policy to a modular scheduler, such as the Linux scheduler. The scheduler can use consolidation during periods of low load where the switch-over point between policies will be dependent on the chip temperature. This would enable energy efficient load balancing mechanisms under variable temperature conditions.

References

[1] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, pp. 33–37, December 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1339817.1339894

[2] C. Mastroianni, M. Meo, and G. Papuzzo, "Analysis of a self-organizing algorithm for energy saving in data centers," in *The Ninth Workshop on High-Performance, Power-Aware Computing*, 2013.

[3] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the 2008 conference on Power aware computing and systems*, ser. HotPower'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855610.1855620

[4] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[5] M. Marsan, L. Chiaraviglio, D. Ciullo, and M. Meo, "Optimal energy savings in cellular access networks," in *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*, 2009, pp. 1–5.

[6] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu, "Delivering Energy Proportionality with Non Energy-Proportional Systems Optimizing the Ensemble," in *HotPower '08: Workshop on Power Aware Computing and Systems*. ACM, Dec. 2008.

[7] [Online]. Available: http://www.eco4cloud.com/

[8] M. Ghasemazar, E. Pakbaznia, and M. Pedram, "Minimizing energy consumption of a chip multiprocessor through simultaneous core consolidation and dvfs," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 49–52.

[9] J. Hopper, "Reduce Linux power consumption, Part 3: Tuning results," 2009, accessed: September 10, 2012. [Online]. Available: http://www.ibm.com/developerworks/linux/library/l-cpufreq-3/index.html

[10] V. W. Freeh, T. Bletsch, and F. Rawson, "Scaling and packing on a chip multiprocessor," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–8.

[11] V. Srinivasan, G. R. Shenoy, S. Vaddagiri, D. Sarma, and V. Pallipadi, "Energy-Aware Task and Interrupt Management in Linux," vol. 2, Aug. 2008.

[12] V. Pallipadi, "cpuidle - Do nothing, efficiently..." [Online]. Available: http://ols.108.redhat.com/2007/Reprints/pallipadi-Reprint.pdf

[13] V. Jimenez, R. Gioiosa, E. Kursun, F. Cazorla, C.-Y. Cher, A. Buyuktosunoglu, P. Bose, and M. Valero, "Trends and techniques for energy efficient architectures," in *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, 2010, pp. 276–279.

[14] M. J. Johnson and K. A. Hawick, "Optimising energy management of mobile computing devices,," in *Proc. Int. Conf. on Computer Design (CDES12)*. Las Vegas, USA: WorldComp, 16-19 July 2012, pp. 1–7.

[15] H. Singh, K. Agarwal, D. Sylvester, and K. Nowka, "Enhanced leakage reduction techniques using intermediate strength power gating," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 11, pp. 1215 –1224, nov. 2007.

[16] S. Borkar, "Design challenges of technology scaling," *Micro, IEEE*, vol. 19, no. 4, pp. 23 –29, jul-aug 1999.

[17] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power cmos digital design," *Solid-State Circuits, IEEE Journal of*, vol. 27, no. 4, pp. 473 –484, apr 1992.

[18] M. T. Jones, "Inside the linux scheduler," Jun 2006. [Online]. Available: http://www.ibm.com/developerworks/linux/library/l-scheduler/

[19] W. Lund. Spurg-bench. https://github.com/ESLab/spurg-bench. Åbo Akademi University.

Paper V

QoS Manager for Energy Efficient Many-Core Operating Systems

Simon Holmbacka, Dag Ågren, Sébastien Lafond, Johan Lilius

Originally published *Proceedings of the 21st International Euromicro Conference on Parallel, Distributed and Network-based Processing, 2013, IEEE International Conference*, pages 318–322. Belfast, UK

QoS Manager for Energy Efficient Many-Core Operating Systems

Simon Holmbacka, Dag Ågren, Sébastien Lafond, Johan Lilius
Department of Information Technologies, Åbo Akademi University
Joukahaisenkatu 3-5 FIN-20520 Turku
Email: firstname.lastname@abo.fi

***Abstract*—The oncoming many-core platforms is a hot topic these days, and this next generation hardware sets new focus on energy and thermal awareness. With a more and more dense packing of transistors, the system must be made energy aware to not suffer from overheating and energy waste. As a step towards increased energy efficiency, we intend to add the notion of QoS handling to the OS level and to applications. We suggest the design of a QoS manager as a plug-in OS extension capable of providing applications with the necessary resources leading to better energy efficiency.**

***Keywords*-QoS, Distributed Operating Systems, Many-Core Systems, Energy Efficiency**

I. Introduction

Pollack's rule [1] describes the performance increase of a CPU as an increase proportional to the square root of the increase in core complexity. As a result of this rule, chips with less complex but more cores are becoming popular. Twice the core complexity will, according to Pollack's rule, result in only about 40% performance speed-up, while using the same amount of transistors for adding more available cores increases the performance potentially by 70-80% [1]. While processing power can be increased by simply adding more cores, developing the software for many-core chips utilizing the parallelism is not trivial. Scalability issues can arise from both performance bottlenecks and new types of power constraints these chips introduce. In this paper we tackle the energy and thermal issues present in many-core chips due to the difficulty of managing the power dissipation efficiently.

Performance is usually maximized by spreading out tasks evenly on the chip, which also results in less thermal hotspots. On the other hand if tasks are scheduled to only a few cores, the idle cores could be shut down and as a result the total energy consumption decreases. A problem arises from this dynamic management and from optimization for energy efficiency without introducing performance degradation. Applications in computer systems usually strive towards high performance; an aim in the opposite direction to lowering the energy consumption. The compromise is to lower the energy consumption as much as possible while still provide the necessary processing power. For this cause we will investigate how to allocate the right amount of resources to the applications at the right time in order to provide sufficient Quality-of-Service (QoS).

QoS is a metric describing the *level of performance compared to a stated specification* [2]. By introducing QoS awareness into the applications, resource allocation can become more energy efficient because applications can deliberately ask for only a defined amount of processing power. The notion of QoS extends applications' influence over resources, and expands energy scalability by enabling control of the resource distribution. A QoS manager is therefore vital to a system level point of QoS control.

This paper presents the design of a QoS manager capable of regulating performance of many-core operating systems. The suggested manager is an OS service to which applications and other OS services can connect and establish the information flow necessary for QoS control. Besides the QoS manager, this paper also focuses on the declaration language the applications use to express their requirements. The contributions of the paper are:

- The QoS manager: a standardized link between applications and resources
- The possibility of applications to hint their resource need to the OS

II. Notion of Performance and QoS

QoS is a term used in real-time systems [3], usually in order to describe the relation to soft deadlines. It is also a term used in cloud computing [4], to enable the *selling* of a bundle of processing power to the user with a certain quality. In both cases, QoS is used to describe the average feasibility of a system without looking at sharp deadlines – a feature we intend to extend the OS with. This notion will enable us to create a more energy efficient system.

Energy is consumed as cores dissipate power over time and by the cooling infrastructure required for actively leading the heat away. Power is required for executing tasks on the processing elements, which in turn create the waste heat. In order to create an energy efficient system, the tasks should: a) execute on the appropriate execution unit and b) be only allocated the necessary amount of resources. For this, the notion of performance is an important measurement for deriving QoS values and how well an application is able to satisfy the user.

The QoS value for an application is determined by comparing the performance requirement in the specification with the actual measured performance. The ratio between these two values is the drop in QoS. If the QoS drop is more than allowed by the specification, the system must control some actors giving the application more resources and thus higher QoS. Similarly, if the performance is too high, the system should decrease the amount of resources to the corresponding application in order to reduce energy waste. For this reason we suggest a new single entity – the QoS manager – controlling the QoS for the applications.

III. The QoS Manager

The presented QoS manager is implemented as an OS extension. The manager is able to measure QoS values from the applications (referred to as sensors), and with the obtained information control the resource allocation on system level. The structure of the manager is shown in Figure 1. It contains the manager, applications and actuators interconnected. The system is built from a *sensor-controller-actuator* structure shown in Figure 1, and is described below.

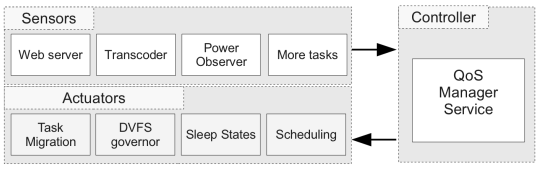


Figure 1. Structure of the OS containing the QoS Manager

Sensor: A sensor is a unit capable of connecting to the manager, expressing QoS and sending measurements to the QoS manager, hence the name *sensor*. Initially, the sensor registers its own QoS requirements and type of resources to use to the QoS manager with a declaration language described in Section IV. Afterwards, the sensor measures its own performance periodically with an implementation specific mechanism in the sensor itself. The performance values are sent to the QoS manager for QoS evaluation. As long as the values are within the specified QoS range, the task of the QoS manager is simply monitoring. In Figure 1 three sensors are connected to the controller: *web server*, *transcoder*, and *power observer*.

Controller: The controller is the part managing the link between sensors and the resources. It contains a database over all established sensor connections and the control unit. Furthermore, the controller handles the resource allocation if the measured QoS from the sensors is too low. Since the sensors are able to hint what kind of resource they lack in such a situation, the controller functions as a plug-in system connecting application directly to the right part of the hardware. All control theoretical implementations are put into the control unit. In this paper we used a simple P-controller (proportional controller) due to its simplicity. The P-controller determines how *much* more/less resources should be allocated to a certain sensor depending on its QoS value. In future work, we intend to investigate more advanced control methods.

Actuator: Actuators are units capable of indirectly altering the sensors' performance by regulating some resources (hardware or software). The way performance increases is sensor dependent, which means that such an actuators must be available, that the requests from the sensor can be fulfilled. A common actuator is the DVFS governor capable of setting the CPU voltage and frequency of a processor. Other actuators could handle sleep states, or migrate tasks from core to core to adjust the level of parallelism. Specific hardware related actuators could shut down memory banks on demand to decrease power dissipation. Even fan controllers can be used to set the fan speed for energy efficient cooling. Sensor choose which actuators to use with a declaration language describe in the following section.

IV. Declaration Language

A simple language has been derived to let the programmer determine QoS requirements for sensors and what actuators are connected to the sensors.

Overview: The declaration language is used during the implementation of the applications, and is compiled to c-code used for the registration and transmission of measurements used in the sensors. Rules and measurements are sent to the QoS manager during runtime. The sensors should therefore use the language to describe required QoS. A template for using the language is shown in Listing 1 and explained below.

```
QoS  MyTemplate {
    requirements{
        boundary:   <condition1>:   <value>;
        boundary:   <condition2>:   <value>;
        ... }
    priority   <value>
    control{
        actuator:    <Actuator candidate1>  <sign>;
        actuator:    <Actuator candidate2>  <sign>;
        ... } }
```

Listing 1. Template for specifying QoS

The declaration language used in the sensors is divided into fields for expressing the performance and QoS. A field contains an entity needed to specify what is intended from the system upon a measurement.

Requirements: The requirements field describes the actual limits for determining QoS boundaries. QoS requirements in form of a performance description is therefore inserted in this field and is compared against a selected `setpoint`. By setting a setpoint, the QoS manager can relate the performance measurements to what would be considered too low (poor performance) or too high (energy waste). In order to specify the accepted range of performance the user must also specify a QoS `limit`, which gives the lower bound of what is considered acceptable. For example the programmer of a webserver can choose a `setpoint` of 500 *requests/sec* and the QoS `limit` of 450 *requests/sec*.

Priority: The priority field determines which (if exists many) of the connected sensors have the highest weight. Situations can occur in which two different sensors' requirements completely conflict each other. In these cases the priority selects how much is weighted from which client. The priority from a thermal guard would for example be prioritized higher than a performance request from a web server if physical damage is imminent because of heat issues. Currently the weighing system is implemented to discard lower prioritized measurements in favor of measurements with higher priority. A more comprehensive way of expressing priorities is part of future work.

Control: The control field is used to describe what actuators should be used by the sensors. Actuators are chosen name wise based on available actuators in the controller database. All actuators are related to a control sign (+ or -). The sign determines in which direction the actuator should aim its output signal for the specific sensor. An example shown in Listing 2 describes is a power observer which strives to minimize the power dissipation of the system.

```
control{
    actuator:   CPU_freq,   −;
    actuator:   CPU_nr,     −;
    actuator:   Parallelize,  −;
}
```

Listing 2. Example of control for a power observer

This sensor has a negative signs on CPU frequency and number of active CPU cores as it aims to shut down and scale down cores in order to reach low power dissipation. It also tries to parallelize as little as possible in order to enable the shut down of cores. A webserver or transcoder could, on the other hand, use positive signs to request more resources if the QoS drops too low.

V. Evaluation

We evaluated a simple system with two sensors: a JPEG decoder which decodes JPEG images in an infinite loop and a power observer which is used to keep the dissipated power under a certain value in order to act as an on-demand power saving mode. The applications were run on top of FreeRTOS [5]. The system was mapped on the Versatile Express board equipped with an ARM Cortex-A9 based CoreTile 9x4 quad-core chip running at 400 MHz with 1 GB of DDR2 memory. Our FreeRTOS port is available at [6]. Figure 2 shows four CPU cores each running one separate instance of FreeRTOS. Our system consist of one master core running the QoS manager and three worker cores. Each core has one JPEG decoder task running. Each decoder task runs completely independent of the other decoders. In this architecture, we are therefore able to decode up to four pictures in parallel.

Measurement data describe how many pictures per second (`p/s`) a core is able to decode, and is sent to the master. The total sum of `p/s` of all four separate JPEG decoding

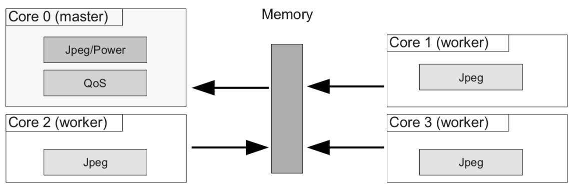


Figure 2. Mapping of the QoS manager on an ARM Cortex-A9 quad-core

instances gives the final `p/s` number for the whole system. Furthermore, the master core implements a power observer, which is used as a on-demand power saving feature. A sleep state mechanism was implemented as actuator; giving the master core the opportunity to shut down individual worker cores in order to lower the power dissipation. Experiments were conducted to show how the energy consumption behaves according to what performance requirements are set in the QoS manager.

Without power requirements: The first set of experiments were conducted without power requirements. Table I shows the requirements for the JPEG decoder in five different tests. The first test (1) has a performance requirement of 7.5 `p/s` with a QoS of 93.3% etc.

Table I
Requirements for the first experiment

Test nr.	1	2	3	4	5
Setpoint [p/s]	7.5	5.5	4.0	3.8	2.0
QoS [%]	93.3	90.9	97.5	84.2	75.0

Results from the first run is shown in Figure 3. Figure 3(A) shows the picture rate of each test run. From the figure it is clear that the cases with a steady curve are successfully provided with the demanded resources most of the execution time. The oscillating curve is a result of demanding such a picture rate that 2.5 active cores are required. To avoid the oscillations, additional actuators such as DVFS could scale down one core in order close the gap between setpoint and requirement more exact. The power dissipation was also measured during the same experiments. Figure 3(B) shows the power output from the same use cases as in Table I, with the oscillating case (test 3) removed for illustrative reasons.

Figure 4 shows the final energy consumption for a 5 minute run on the Cortex-A9 for all mentioned test cases and four additional configurations. It shows clearly how the energy consumption increases steadily as the performance requirements (`p/s`) increase. The result is a nearly linear relationship between performance and energy consumption due to the QoS manager with the sleep state actuator.

With power requirements: The next set of experiments included power requirements to give a more realistic situation with multiple parameters to match. Similarly to the first experiment, a JPEG decoder was used as application with the same range of performance requirements. A power observer application was added to the system. The power observer measures the power dissipation of the chip. Ideally

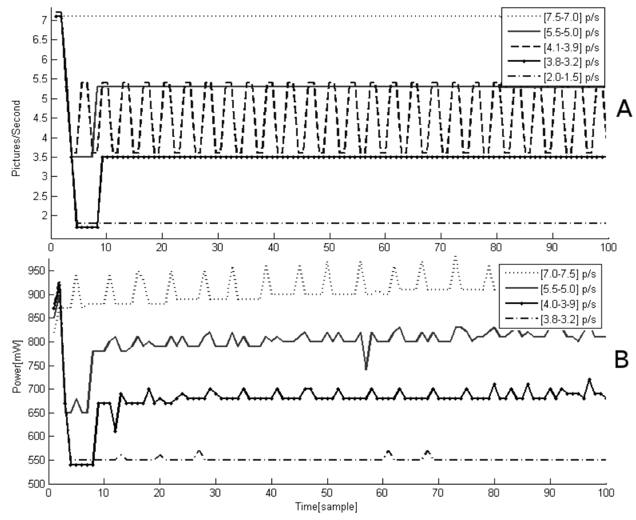


Figure 3. Power dissipation with different performance settings

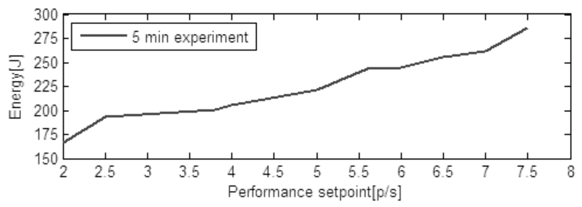


Figure 4. Energy consumption with different performance settings

it requires 650 mW of dissipated power, but accepts power dissipation up to 700 mW. The power observer uses also a higher priority than the JPEG decoder sensor to function as a power saving and heat protection feature.

With these settings experiments were run for the first settings in Table I. Without power constraints the system activated four cores during the whole test and dissipated roughly 900mW on average.

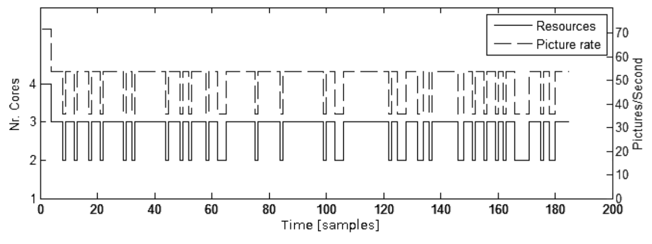


Figure 5. Results from experiment with power constraints ([7.5-7.0]$\mathrm{p/s}$)

By adding power constraints, the system is forced to shut down some cores in order to meet the higher prioritized requirement from the power observer. Figure 5 shows how the system is forced to operate mostly with three active cores with occasional usage of only two cores. This experiment shows that the system is able to override requirements on demand by higher prioritized sensors in order to obtain power saving feature etc.

VI. Related Work

QoS management and monitoring exist in different areas; from cloud infrastructures and web servers [4], [7] to OS level on single computers and real-time systems [3], [8], [9] etc. Language constructs for injecting QoS support has also earlier been presented. Aagedal presented in his PhD thesis [2] CQML; a language having the property of describing QoS requirements. In this work, applications specify what performance is to be expected from it and what is considered as performance in context of the application. Applications also monitor own performance and signal this value to the QoS manager periodically.

We use similar notations inspired by the languages to describe QoS in applications, with more focus on the OS-level support. Our manager will be implemented as an OS extension capable of system level control many-core systems. Furthermore we have added the control output, by which applications can choose which action needs to be taken if the desired QoS is not achieved.

Design choices for a run-time manager was presented in [10], consisting of a resource manager and a quality (QoS) manager. The task of the QoS manager was to optimize an operation point such that the system is maximally utilized. Utilization is controlled by adjusting quality points in the applications i.e. selecting one of many performance levels an application specifies. Video resolutions or frame rate for a transcoder are examples of such performance levels. Similarly in [11], a system PowerDial is used to insert configuration parameters (knobs) into applications and tune their values to achieve the best accuracy vs. performance trade-off. Complementary to Hoffman's work [11], his application tuning knobs can be used as a single actuator in our model, which forms an application to be both a sensor and actuator at the same time. Instead of controlling the applications, our manager is intended to only monitor the applications which indirectly influence the resources.

The managers in [9] and [10] require the application programmer to specify required processing power, memory and communication capabilities. We intend to simplify requirement notation by only requiring an abstract *quality* value freely defined by the programmer. The programmer does not need to modify the application or analyze performance points in order to use the presented QoS manager.

VII. Conclusions

In this paper we have introduced a QoS manager for improving the energy efficiency of many-core systems. The manager is intended to make the system better utilize the resources of the platform depending on the workload. Applications are referred to as sensors; actors capable of declaring performance and QoS requirements. By introducing the notion of QoS, sensors are able to signal their resource requirements and, through the QoS manager, allocate the resources. The QoS manager control a set of actuators capable of altering the performance characteristics for the sensors. Sensors also set what type of actuator is required for increasing performance of a certain type of sensor, which gives the programmer opportunities to tailor the resources more exact to the application. It also allows future optimization techniques to be plugged in to the QoS manager and used by any sensor if suitable.

The QoS manager has been evaluated on a quad-core ARM Cortex-A9 with a JPEG decoder and its picture rate as use case. The experiments have shown that the QoS manager is able to scale down the energy consumption of the chip in two different ways. Firstly, the application can by itself relax the performance requirements to a given rate and thereby request less resources. Secondly, other sensors with higher priority can force the system to allocate less resources to lower prioritized sensors.

In contrast to current systems, more awareness on the thermal distribution inside the chip must be made when using many-core systems because of the very dense packing of cores and the spacial locality. Controlling QoS will therefore be an important part of the many-core evolution. By using the system level QoS manager, the distributed many-core system can more easily be optimized for a global maximum since the applications can hint the controller of how resources should be used.

VIII. Future Work

An issue not addressed in this paper is the control theoretical view of the QoS controller. Since this part is the system level of control, methods such as PID or MP or fuzzy control should be tested and evaluated complete with stability analysis and tuning rules etc. As this system uses multiple inputs from sensors and multiple outputs to the actuators, a state spaced-based method could enable the possibility for constructing a more efficient controller. Other alternatives would be to formulate the system as a optimization problem in which the objective function minimizes the power dissipation and QoS requirements are the constraints. This would also improve the current priority model since the system would, with more rigorous methods, determine the lowest total cost (power vs. performance) of the system.

The complexity of the controller is also an important parameters especially in a large many-core system, as the number of inputs/outputs is likely to grow rapidly. As the number of cores grow towards extreme numbers (1000+) a single manager will become a bottleneck for communication even if the complexity is very low. To solve the issue, the manager must be decentralized and function as a distributed system with sub-managers handling certain *islands* of cores eventually grouped into *continents* of cores.

We intend to develop the complete environment for demonstrating the scaling effects of the QoS manager on a true many-core platform such as the SCC [12] or TilePro64 [13] and also construct the necessary actuators needed to control such a system efficiently. For example energy efficient scheduling, task migration and dynamic voltage and frequency scaling are techniques useful to create the required actuators. We also intend to use a more complex mix of applications requesting different actuators with different priorities for a more realistic conclusion.

References

[1] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the 44th annual DAC*. New York, NY, USA: ACM, 2007, pp. 746–749.

[2] J. Aagedal, "Quality of service support in development of distributed systems," Ph.D. dissertation, University of Oslo, Oslo, Norway, March 2001.

[3] F. Monaco, E. Mamani, M. Nery, and P. Nobile, "A novel qos modeling approach for soft real-time systems with performance guarantees," in *HPCS '09.*, june 2009, pp. 89 –95.

[4] P. Zhang and Z. Yan, "A qos-aware system for mobile cloud computing," in *CCIS, 2011 IEEE International Conference*, sept. 2011, pp. 518 –522.

[5] R. Barry, *FreeRTOS Reference Manual: API functions and Configuration Options*, Real Time Engineers Ltd, 2009.

[6] D. Ågren. Freertos cortex-a9 mpcore port. Åbo Akademi University. [Online]. Available: https://github.com/ESLab/FreeRTOS---ARM-Cortex-A9-VersatileExpress-Quad-Core-port

[7] T. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: A control-theoretical approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, p. 2002, 2001.

[8] B. Li and K. Nahrstedt, "A control-based middleware framework for quality-of-service adaptations," *Selected Areas in Communications, IEEE Journal on*, vol. 17, no. 9, pp. 1632 –1650, sep 1999.

[9] V. Segovia, "Adaptive cpu resource management for multicore platforms," Licentiate Thesis, Lund University, Sep. 2011.

[10] V. Nollet, D. Verkest, and H. Corporaal, "A safari through the mpsoc run-time management jungle," *Journal of Signal Processing Systems*, vol. 60, no. 2, pp. 251–268, 2008.

[11] H. Hoffmann and S. Sidiroglou, "Dynamic knobs for responsive power-aware computing," in *Proceedings of the sixteenth ASPLOS conference*. New York, NY, USA: ACM, 2011, pp. 199–212.

[12] P. Thanarungroj and C. Liu, "Power and energy consumption analysis on intel scc many-core system," in *30th (IPCCC), 2011*, nov. 2011, pp. 1 –2.

[13] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, pp. 15–31, 2007.

Paper VI

Energy Efficiency and Performance Management of Parallel Dataflow Applications

Simon Holmbacka, Erwan Nogues, Maxime Pelcat, Sébastien Lafond, Johan Lilius

Originally published *The 2014 Conference on Design & Architectures for Signal & Image Processing [BEST PAPER], ECSI International Conference*, pages 133–141. Madrid, Spain.

Energy Efficiency and Performance Management of Parallel Dataflow Applications

Simon Holmbacka*, Erwan Nogues†, Maxime Pelcat†, Sébastien Lafond* and Johan Lilius*
*Department of Information Technologies, Åbo Akademi University, Turku, Finland
Email: {sholmbac,slafond,jolilius}@abo.fi
†UMR CNRS 6164 IETR Image Group, INSA de Rennes, France
Email: {erwan.nogues,maxime.pelcat}@insa-rennes.fr

***Abstract*—Parallelizing software is a popular way of achieving high energy efficiency since parallel applications can be mapped on many cores and the clock frequency can be lowered. Perfect parallelism is, however, not often reached and different program phases usually contain different levels of parallelism due to data dependencies. Applications have currently no means of expressing the level of parallelism, and the power management is mostly done based on only the workload. In this work, we provide means of expressing QoS and levels of parallelism in applications for more tight integration with the power management to obtain optimal energy efficiency in multi-core systems. We utilize the dataflow framework PREESM to create and analyze program structures and expose the parallelism in the program phases to the power management. We use the derived parameters in a NLP (Non Linear Programming) solver to determine the minimum power for allocating resources to the applications.**

Keywords*—Power manager, Multi-core, Application Parallelism, Dataflow framework

I. Introduction

Energy efficiency in computer systems is a continuous coordination between the power dissipation of the used resources and the execution time of the applications. In multi-core systems energy efficiency is a question of both the time and space sharing of resources, and is highly dependent on the application characteristics such as its level of parallelism (referred to as P-value). Many studies have investigated the relationship between power dissipation and parallel execution [1], [11], [17]. The general solution to reach energy efficiency is to map parallel applications onto several cores to exploit the parallelism and hence enable clock frequency reduction without any performance degradation.

The parallelization will, however, in practice be restricted by the application's own internal scalability i.e. the P-value(s) in the application. This factor is a crucial parameter which describes the application's behavior and directly influences which power saving techniques to use and what resources to allocate. For example, resource control for sequential applications is only possible by scaling the clock frequency, while parallel applications are both influenced by the number of available processing elements and their clock frequency.

To extract the P-value is, however, a non-trivial task since **a)** the value depends on the programming techniques, usage of threads, tasks etc. and **b)** the P-value usually varies in the execution phases of the program because of non-parallel paths, synchronization points etc. This means that resource allocation should be done differently in different program phases.

Power saving techniques such as DVFS (Dynamic Voltage and Frequency Scaling) and DPM (Dynamic Power Management based on sleep states) can be utilized to bring the CPU into the most power efficient state, but is currently only driven by the system workload. This means that hardware resources can be over allocated even though the application does not provide useful work. To provide applications with a sufficient amount of resources, the application performance should be monitored rather than the CPU workload. For example in a parallel phase of an application, DVFS and DPM could be utilized to enable many cores and to reduce their clock frequency to save power. On the other hand during a sequential phase, DVFS could be used to increase the clock frequency on the active core to gain performance, and the unused cores could be shut down to save power. This interplay between DVFS and DPM during the program phases is only possible when describing the program performance and parallelism and when observing the program progression during runtime.

Rather than providing this information by hand, dataflow frameworks such as PREESM [16] provides tools for explicit parallelization by single rate Synchronous Data Flow (SDF) transforms, which can be exploited for extraction of the P-value in the program phases. We use this framework to show how dataflow tools can be used for energy efficient programming and tight integration of the resource management. We provide the following contributions:
a) We demonstrate the extraction of the P-values at compile-time with the PREESM framework.
b) The P-values are injected together with QoS (Quality of Service) parameters at runtime into the program phases to steer the power saving features of the multi-core hardware.
c) A NLP solver is used to allocate resources with minimum power dissipation for given QoS requirements.

Our approach demonstrates up to 19% energy savings for real world applications running on multi-core hardware and using a standard Linux OS without any modifications.

II. Related Work

Various ways of using parallelization for achieving energy efficiency have been studied in the past. The key goal has been to spread out the workload [21] on several cores in order to lower the clock frequency [9], hence lowering the dynamic power dissipation while keeping constant performance.

Video applications have been popular use cases to demonstrate such energy efficiency; Yang et. al. [23] presented smart cache usage tailored for a MPEG-2 decoder to steer cache usage for energy efficiency by utilizing application specific information during runtime. The work in [14] formulated a rigorous scheduling and DVFS policy for slice-parallel video decoders on multi-core hardware with QoS guarantees on the playback. The authors presented a two-level scheduler which firstly selects the scheduling and DVFS utilization per frame and secondly maps frames to processors and set their clock frequencies. In our work, we lift the level of abstraction to any kind of application while retaining video processing only as a suitable use-case. Our QoS and power manager is hence not tied to a certain application or system but is intended as a more generic solution for energy efficient parallel systems.

Jafri et. al. [11] presented a resource manager which maps parallel tasks on several cores in case energy efficiency can be improved. The authors used meta data in the applications to describe different application characteristics such as task communication, and based on this data determine the parallelization by calculating the corresponding energy efficiency. Complementary to this work, we inject meta data in form of QoS and the P-value, but orthogonally to compile-time information we address runtime information which requires no specific compiler knowledge and can be changed during runtime.

On a fundamental level, energy- and power efficiency is dependent on the proper balance between static and dynamic power dissipation of the CPU. Rauber et. al. [17] provided the mathematical formulation for the scheduling and the usage of clock frequency scaling to minimize the energy consumption for parallel applications. The results indicate that execution on very high clock frequencies are energy inefficient even though the execution time is minimized. This is a result of the high dynamic power dissipation when executing on high clock frequencies and the increase in static power due to high temperatures. Similarly in [1], Cho et. al. formulate mathematically the best balance between dynamic and static power to achieve minimal energy consumption. We acknowledge these findings in our work and aim to realize the outcomes by utilizing DPM and DVFS to obtain minimal power while keeping the QoS guarantees defined in the applications. Furthermore we also take the temperature into account, which significantly affects the static power dissipation [5]. We also create our power model specifically for a given CPU type, which gives us the total power dissipation as a function of resource usage.

Finally we evaluate our system on real consumer hardware to demonstrate the feasibility of integrating the proposed strategies into real-world applications.

III. QoS & Parallelism Aware Strategy

In this work we focus on general streaming applications in which **1)** QoS requirements can be defined and **2)** performance can be measured. An example is a video processing application, which processes and displays a video for a set amount of time. From this application we demand a steady playback (e.g. 25 frames per second) for the whole execution, but the execution speed of the internal mechanisms such as filtering is usually completely dependent on the hardware resource allocation.

Applications demand resources in order to perform the intended functionality, which results in a power dissipation Pw of the CPU over a time t. Since the energy consumption is the product of Pw and t, an energy efficient execution is obtained as the product is minimized. The power Pw is further divided into the sum of the dynamic power Pw_d and the static Pw_s, hence $Pw = Pw_d + Pw_s$. The dynamic power is given by $Pw_d = C \cdot f \cdot V^2$, where C is the effectively switched capacitance, f is the frequency and V is the voltage of the processor. The static power consists mainly of leakage currents in the transistors and increases with smaller manufacturing technologies and temperature [13]. The static power is hence present during the whole execution and becomes the dominating power factor as clock frequencies decrease and execution time increase [1].

The popular (and easily implementable) execution strategy called *race-to-idle* [18] was implemented to execute a task as fast as possible, after which the processor enters a sleep state (if no other tasks are available). The *ondemand* (OD) frequency governor in Linux supports this strategy by increasing the clock frequency of the CPU as long as the workload exceeds an `upthreshold` limit. Race-to-idle minimizes t, but on the other hand results in high power dissipation Pw during the execution. A strategy such as race-to-idle will have a negative impact on energy efficiency if the decrease in time is less than the increase in power i.e. $\Delta^{-}t < \Delta^{+}Pw$ compared to running on a lower clock frequency. Depending on the CPU architecture and the manufacturing technology this relation varies, but with current clock frequency levels, is it usually very energy inefficient to execute on high clock frequencies [24], [17]. It is also (usually) inefficient to execute on very low clock frequencies [5] since the execution time becomes large and the static power is dissipated during the whole execution.

Our strategy is to *execute as slow as possible while still not missing a given deadline*; we call it QP-Aware (QoS and Parallel). Figure 1 Illustrates two different execution strategies for a video processing application: Part A) illustrates the race-

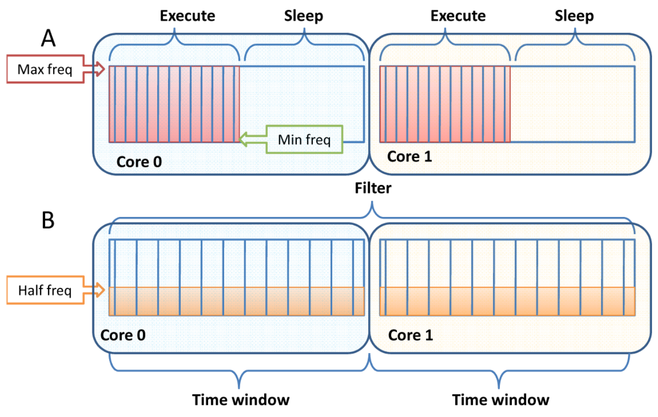


Fig. 1. Two execution strategies: A) Race-to-idle B) QP-Aware

to-idle strategy in which the operations are executed as fast as possible for a short time, after which it idles for the rest of the video frame window. Part B) illustrates the QP-Aware strategy in which the operations are executed as slowly as possible while still keeping the frame deadline of the playback. If the execution time in case A) is twice as fast but the power dissipation is more than twice as high, case B) will be more energy efficient. Moreover, frequently switching the frequency and voltage introduces some additional lag, which also impacts on the energy consumption.

We argue for the B-type of execution in streaming applications, in which the application executes on more energy efficient frequency [6] with the appropriate amount of active cores, which is dependent on the application P-values injections and the QoS requirements. In the general case a QP-aware strategy is possible whenever the performance of an application can be measured, either with an application specific metric such as the framerate or with a more generic metric such as heartbeats [7].

IV. Power Optimizer

To set QoS requirements and to scale the performance of the software according to the requirements of the application, we implemented a power optimizer to regulate the hardware such that minimal power is dissipated for the required performance. Current power managers, such as the frequency governors in Linux, base the resource allocation purely on system workload levels. Resources are allocated as the workload reaches a certain `upthreshold`, which is usually done on CPU level rather than on core level. This means that the power management has no information of the program behavior such as its parallelism, nor any notion of how the workload should be mapped on the processing elements.

The structure of our power manager supports: P-value injections and QoS declarations in the applications. The P-values are easily injected by the programmer with a function call to a provided power management library for each application. Similarly, the QoS requirements are set using any performance metric [8] with a function call to the QoS library.

Applications are provided with an interface to the power manager, which in turn regulates the power saving techniques (called *actuators*) as illustrated in Figure 2. Actuator regulation is calculated from two defined cost models describing *power* and *performance*. The models are mathematical representations of the real system used for calculating the effect of resource usage. Since different chip architectures behave differently when using various combinations of DVFS and DPM, the models are easily interchangeable and can be re-engineered for any chip architecture by a chosen system identification method. Figure 2 illustrates the information flow from application to actuator.

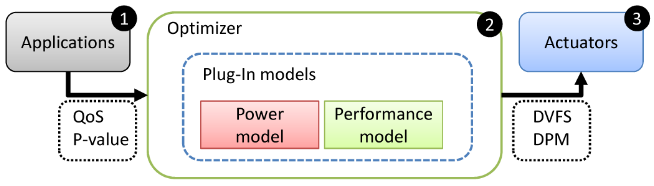


Fig. 2. Information flow from application to actuator

The blocks are defined as follows:

1) Applications are normal user space programs connected to the optimizer and are capable of expressing QoS and P-value(s)
2) The Optimizer determines the optimal combination of actuator utilization based on the QoS and P-value inputs from the Applications and the mathematical cost models
3) Actuators are power saving techniques (DPM and DVFS), with a degree of utilization determined by the Optimizer

Figure 3 illustrates the structure of the application ecosystem together with the power optimizer compared to the default Linux Completely Fair Scheduler (CFS) Scheduler and the

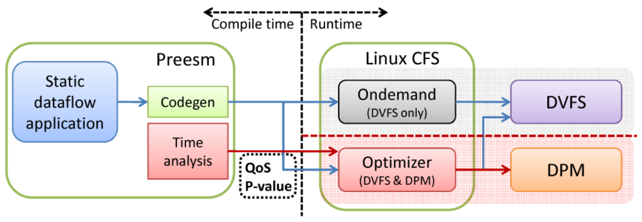


Fig. 3. Structure of the application ecosystem

ondemand (OD) frequency governor. Compile-time tools such as PREESM are used to simplify the P-value extraction (Section V) and QoS declaration in the applications by automatic time analysis of the application. While the default CFS+OD is only able to scale the system according to the workload, the power optimizer can exploit extracted P-values and QoS requirements previously defined.

A. System Identification

The key issue for model based control systems is to identify the system as a mathematical expression used for control decisions. The model should be as accurate as possible to the real case, but also remain simple in order to not introduce unnecessary computational overhead. The system identification is, in this paper, made for an Exynos 4412 microprocessor based on the quad-core ARM Cortex-A9 which is an off-the-shelf microprocessor used in many mobile phones and tablets. We show in this section how to set up the NLP solver for minimizing the power dissipation while keeping the QoS guarantees in the applications.

1) Power model identification: We trained the power model of the Exynos chip by increasing the frequency and the number of active cores step-wise while fully loading the system. As workload we ran the `stress` benchmark under Linux on four threads during all tests, which stressed all active cores on the CPU to their maximum performance.

The dissipated power was measured with hardware probes for each step and is shown Figure 4. As seen in the figure, the power dissipation of the chip peaked the highest using high clock frequency and with many cores. Even though the `stress` benchmark does not reflect the power trace of any application exactly, we still consider its power output as a sufficiently close compromise.

Since the power trace in Figure 4 is clearly not linear, we used a similar approach to [19] for deriving our power model.

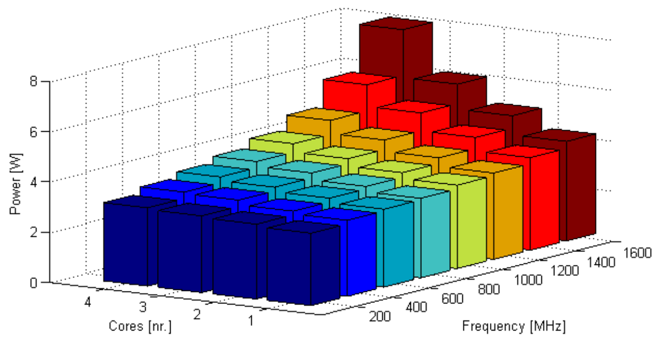


Fig. 4. Power as function of nr. of cores and clock frequency (fully loaded). Hot temperature on top and cold on bottom

We denote the control variables for DVFS and DPM as q and c respectively. Since these variables are only used as control variables in the optimization algorithm, the variables are unitless and chosen in the range [1 - 8] where 1 is minimum utilization and 8 is maximum utilization of a specific actuator. The goal is to define a surface as close as possible to the data values in Figure 4. The third degree polynomial

$$P(q,c) = p_{00} + p_{10}q + p_{01}c + p_{20}q^2 + p_{11}qc + p_{30}q^3 + p_{21}q^2c \tag{1}$$

where p_{xx} are coefficients was used to define the surface. We used Levenberg-Marquardt's algorithm [10] for multi dimensional curve fitting to find the optimal coefficients, which minimizes the error between the model and the real data. Table I shows the derived parameters and Figure 5 illustrates the model surface with the given parameters. To verify our

TABLE I. COEFFICIENTS FOR POWER MODELS

p_{00}	p_{01}	p_{10}	p_{11}	p_{20}	p_{21}	p_{30}
2.34	0.058	0.598	-0.025	-0.161	0.010	0.012

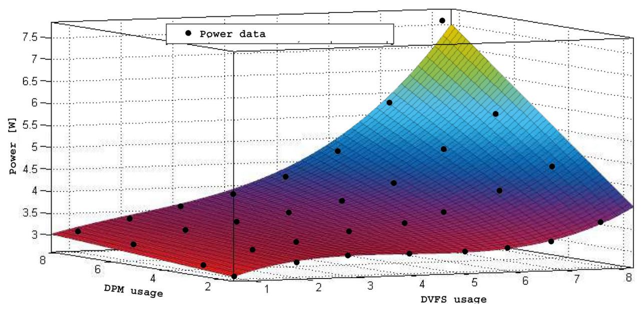


Fig. 5. Surface of the hot use case derived from Equation 1. Dots are real data measurements

model we calculated the error difference between the real data and the derived model. The maximum difference of 10,2% was obtained when using four cores and running on the highest clock frequency, while the average difference was only 0.6%. With the rather small average difference and with a computationally simple model, we considered the model feasible for our experiments.

2) Performance model identification: In order to determine which power saving technique to use, the optimizer requires knowledge on how much it *affects* the applications. For example a sequential program would not gain any performance by increasing the nr. of cores, while a parallel application might save more energy by increasing the nr. of cores instead of increasing the clock frequency. Similarly to the power model, the performance model is equally flexible and can be exchanged during runtime.

We modeled DVFS performance as a linear combination of clock frequency q as:

$$\text{Perf}_q(\text{App}_n, q) = K_q \cdot q \tag{2}$$

where K_q is a constant. This means that e.g. 2x increase in clock frequency models a double in speed-up. Even though the performance in reality could fluctuate by memory/cache latencies etc., we consider the approximation in the general case as close enough.

In contrast to the simpler relation between performance and clock frequency, modeling the performance as a function of nr. of cores is more difficult since the result depends highly on the inherited parallelism and scalability in the program.

To assist the optimizer, we added the notion of expressing parallelism (P-value) directly in the applications. The programmer is allowed to inject the P-value in any phase of a program in the range [0, 1] where 0 is a completely sequential program phase and 1 is an ideal parallel program phase. This value can either be static or change dynamically according to program phases [20]. Calculating the P-value can be done by using various methods such as [22], [2], [15], but in this paper we chose to utilize the functionality of PREESM to automatically determine the P-value directly from the dataflow graph.

Our model for DPM performance uses Amdahl's law:

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \tag{3}$$

where P is the parallel proportion of the application and N is the number of processing units. The final performance model for DPM is rewritten as:

$$\text{Perf}_c(\text{App}_n, c) = K_c \cdot \frac{1}{(1-P) + \frac{P}{c}} \tag{4}$$

where K_c is a constant and c is the number of cores. This models a higher performance increase as long as the nr. of cores is low but decreases as the nr. of cores increase. It means that as more cores are added the speed-up becomes ever more sub-linear, until increasing performance by DVFS eventually becomes more energy efficient.

To describe the performance of the whole system we calculate the sum of both DVFS and DPM performance as: Perf_{Tot} = Perf_q + Perf_c

B. NLP optimization solver

With the derived models, we adopted a non-linear Sequential quadratic programming (SQP) solver for calculating the optimal configuration of clock frequency and number of active cores (DVFS vs. DPM) under performance criteria. The required resources are given as a *setpoint* S, and the lack of resources is monitored in the applications and is sent as a positive error value E to the optimizer. The application can request more resources by setting a lower bound *QoS limit* Q, which indicates the lowest tolerable performance. We set-up the power optimization problem as follows:

$$\text{Minimize}\{\text{Power}(q,c)\}\text{Subject to:}$$
$$\forall n \in \text{Applications}: E_n - (Perf_q + Perf_c) < S_n - Q_n \quad (5)$$

where q is clock frequency, c is the number of cores and $Perf_q$ and $Perf_c$ is the performance of DVFS and DPM respectively. S_n is the performance setpoint, E_n is the difference (*error value*) between the performance setpoint and the actual performance and Q_n is the lower QoS limit. The optimization rule states to *minimize the power while still providing sufficient performance to keep above the QoS limit*. This is achieved by setting the actuators (q, c) to a level sufficiently high such that enough errors E_n are eliminated for each application n.

Our chosen baseline method implemented the SQP [4] solver with the plain objective function and side constraints given in Eq. 5. For a faster solution we added the gradient function $g = \begin{bmatrix} \frac{\partial f}{\partial q} \\ \frac{\partial f}{\partial c} \end{bmatrix}$ which approximates the search direction with a first order system. We also provided the analytical partial derivatives of the side constraints $C = \begin{bmatrix} \frac{\partial C}{\partial q, \partial c} \end{bmatrix}$ to the solver for a more accurate solution, where $\frac{\partial C}{\partial q, \partial c}$ are the first order derivative of actuators with respect to the side constraints.

The SQP solver was implemented in the c-language and compiled for the ARM platform with -O3. The time for obtaining a solution for one iteration was timed to roughly 500 - 900 μs on the ARM platform clocked to 1600 MHz, which is fast enough to not interfere with the system.

V. Parallelism and QoS in Dataflow

For rapid development and a more pragmatic view of the application, we use the dataflow framework PREESM for the software implementation. Indeed, the capabilities of dataflow programming is exposed and we show how such tools can in practice be used for integration of QoS and P-value extraction of the applications.

A. Static Dataflow

In many cases a signal processing system can work at several levels where actors fire according to their in- and output rates. The concept of SDF graphs for signal processing systems was developed and used extensively by Lee and Messerschmitt [3]; it is a modeling concept suited to describe parallelism. To enlighten the purpose of the discussed method within static parallel applications, we describe the general development stages briefly. The first step in the design process is a top-level description of the application, which is used to express the data dependency between the actors, so called *edges*. An SDF graph is used to simplify the application specifications. It represents the application behavior at a coarse grain level with data dependencies between operations. An SDF graph is a finite directed, weighted graph $G =< V, E, d, p, c >$ where:

- V is the set of nodes.
- $E \subseteq V \times V$ is the set of edges, representing channels which carry data streams.
- $d : E \rightarrow N \cup \{0\}$ is a function with $d(e)$ the number of initial tokens on an edge e
- $p : E \rightarrow N$ is a function with $p(e)$ the number of data tokens produced at e's source to be carried out by e
- $c : E \rightarrow N$ is a function with $c(e)$ representing the number of data tokens consumed from e by e's sink node.

This model offers strong compile-time predictability properties but has limited expressive capability. Several transformations are available to transform the base model and optimize the behavior of the application ([16]).

The Single rate SDF (srSDF) transformation (Figure 6) transforms the SDF model to an srSDF model in which the amount of tokens exchanged on edges are homogeneous (production = consumption), which reveals all the potential parallelism in the application. As a consequence, the system

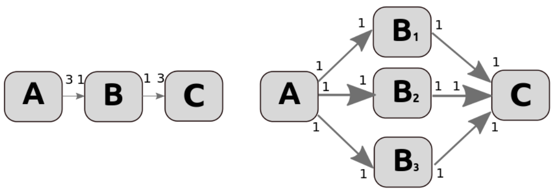


Fig. 6. A SDF graph and its srSDF transformation – multi-rate link to is transformed to several single-rate links to enable parallelism

scheduling can easily benefit of the srSDF to process data in parallel. The data edges of the original graph is used for the data synchronization of the exploded graph and is used to defined *sequences* of processing from which P-values can be extracted.

B. Extracting QoS and P-value with PREESM

A flexible prototyping process has an important role in system architecture to optimize performance and energy consumption. The purpose is to find a way of explore architecture choices with a good adequacy for the application. PREESM [16] is an opensource tool for rapid prototyping which automatically maps and schedules hierarchical SDF graphs on multi-core systems. Using what is called a scenario (Figure 7), the user can specify a set of parameters and constraints for the mapping and scheduling of tasks. This restricts for instance the mapping of an actor on a subset of cores of the architecture. The workflow is divided into several steps

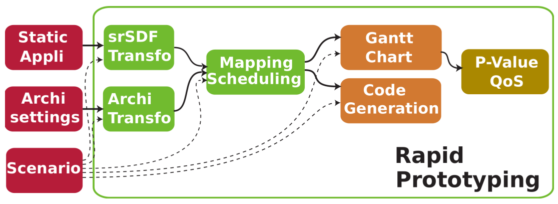


Fig. 7. Rapid prototyping description to extract QoS and P-value

as depicted in Figure 7, which can be used to extract the parallelism of the application:

- *Single rate transformation (srSDF)* exposes the possible parallelism
- *Mapping & Scheduling* finds the best adequacy between the architecture parameters and the application graph
- *Gantt chart generation* illustrates the parallelism of the application as a function of time
- *Code generation* provides a rapid test code to run on the platform using the outputs of the previous steps

Dataflow representation increases the predictability of the applications, which enables an accurate description of the parallelism. The PREESM tool was used to generate applications with different behavior and extract their P-values used by the Optimizer to design energy efficient systems.

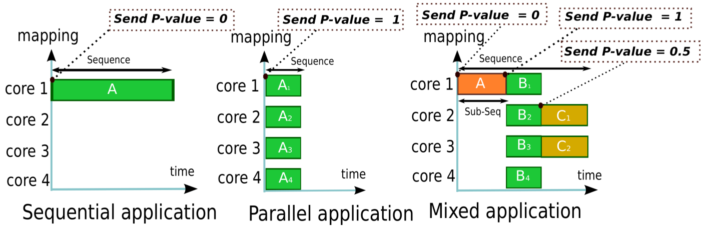


Fig. 8. Extracting P-value from the Gantt chart in PREESM

Figure 8 illustrates different considered behaviors of applications: the sequential case maps a single actor A on a single core, while in the parallel case the actor can be divided up into smaller pieces and executed on all cores. The mixed application has non-dividable actor A which must be executed on a single thread before the B actors can execute, which is a typical behavior in general parallel applications. We extract the P-value in the range $[0.0, 1.0]$, where 0.0 is a serial sequence and 1.0 is a ideal parallel sequence for the used hardware platform. Consequently a value of 0.5 describes a scalability to half of the processing elements. From the Amdahl's law (Eq. 3) and the Gantt chart (Figure 8) we calculate the *P-value* as:

$$P\text{-}value = \left(\frac{\frac{1}{S} - 1}{\frac{1}{N} - 1}\right) \tag{6}$$

where S is the speed-up factor between the sequential and optimized applications after parallelization and $N > 1$ is the total number of cores. The *P-value* can furthermore be calculated as an average of the whole sequence or dynamically for each sub-sequence for enhanced precision.

VI. Experimental Results

We evaluated a video processing application, which is a typical streaming application and is dependent on QoS guarantees to provide a requested playback rate. The evaluation platform was the previously mentioned quad-core Exynos 4412 board. We implemented and mapped the power optimizer and its infra structure on Linux (kernel version 3.7.0) with the NLP solver and communications backbone implemented in the c-language.

A. Application description

The video processing application consisted of a sequence of filters and a buffer connected to the display output. With our designing framework, we added QoS requirements on the filtering to match the intended playback rate of 25 frames per second (fps) with an additional small safety margin i.e. 26 fps to ensure that no framedrops would occur during the playback. This means that it filters frames with a rate of 26 fps and sleeps for the remaining (very short) time window; with this behavior, the filter follows the QP-Aware strategy illustrated in Figure 1 part B rather than executing as fast as possible and then sleep for a longer time (part A).

To cover the different use cases, we chose three types of video processing implementations: fully sequential, fully parallel and mixed-parallel as seen in Figure 8.

For performance evaluation an edge detection algorithm is used to filter the decoded image. The Sobel filter is an image transformation widely used in image processing to detect the edges of a 2-dimensional video. It is a particularly good application to explore architecture strategies as it can be made parallel for the filtering part and sequential for any preprocessing function [12]. Once the data is processed, the output can be displayed with a real-time video display at 25 fps. By optimizing the execution time using parallel processing, the difference between the filtering and displaying rates can be used for energy optimization.

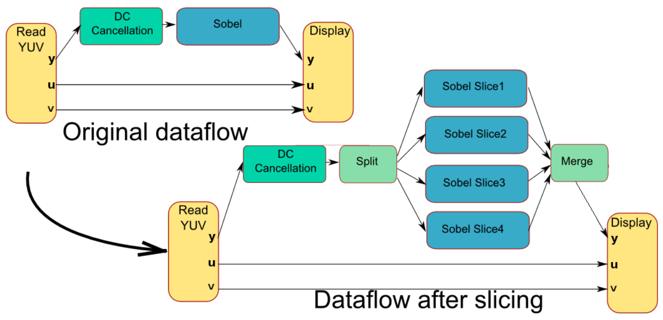


Fig. 9. Top level description - Original dataflow and after single rate transformation extracting data parallelism via slicing

Figure 9 shows the system description of an edge detection sequence for a YUV video. The video is firstly read from a source (Read YUV) after which it passes through a sequence of filters and finally is displayed (Display). The filtering part can be parallelized by multi-threaded execution [12] since the picture on which the filter is applied can be divided into several slices without internal data dependencies as seen in the right part of Figure 9. The DC Cancellation filter is an optional choice for preprocessing the video. This algorithm cannot be parallelized and was added to the third use-case, the mixed-parallel application, in order to force mixed parallelism into the application. In the other use-cases, this filter was not applied.

The three applications were generated automatically using PREESM. The P-values were injected into the automatically generated code by adding function calls for sending the P-values to the optimizer. For fully serial sequences, we injected $P = 0.0$, which (according to Amdahl's law) means a scalability up to 1 core in the 4 core system. For completely parallel sequences we naturally injected $P = 1.0$, and for mixed sequences we injected P values according to Eq. 6. With these setups we ran the three different use-cases with both the default CFS+OD and the optimizer for a 5-minute run.

B. Sequential application

We firstly evaluated the sequential implementation of the application in order to have a reference for comparison. The sequential application run only a single threaded Sobel filter (Figure 9) after which the frame is displayed. Figure 10

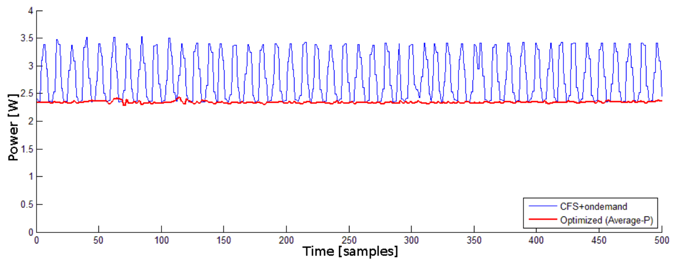


Fig. 10. Power trace from the sequential application using default CFS and with power optimizer

shows the power trace from a 500 sample part of the run. As predicted, the CFS with the OD governor decodes the video very fast for a given time after which it idles for the rest of the time frame. This is clearly seen in the figure as the power dissipation of the CFS+OD case oscillates heavily. By using the optimizer, the power dissipation is more stable and the average power dissipation is much lower partly by using the QP-Aware strategy and partly by disabling the unused cores.

C. Parallel application

The second application performed the same functionality as the sequential case, but with the Sobel phase parallelized and mapped on all four cores as the parallel case in Figure 8 and 9. This configuration would (in theory) speed-up the software roughly four times, which would allow the power saving features to scale down the hardware resources to save power. Figure 11 shows interestingly roughly the same power

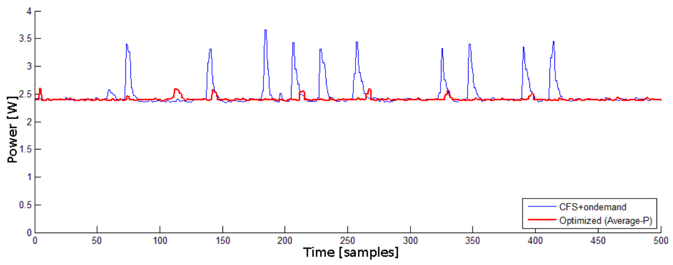


Fig. 11. Power trace from the parallel application using default CFS and with power optimizer

output for the optimized case compared to Figure 10. This is because the static power increase when using more cores is almost identical to the dynamic power decrease of reducing the clock frequency – this is an occurring phenomenon as systems run on very low clock frequencies with many cores [5]. The situation could be improved by fine tuning the power model to enable higher precision. The CFS+OD case, on the other hand, shows more power reduction since the workload of the cores most of the time is below the `upthreshold` limit for the OD governor.

D. Mixed-parallel application

The third use-case was the mixed-parallel application with a serialized DC Cancellation filter added before the parallel Sobel filter. This means that the filtering job will be more computational heavy than in the previous two cases. We profiled the execution with `gprof`, with the timing portion of 66% for the DC Cancellation filter and 25% for the Sobel filter.

We evaluated this use-case with both the *Average P-value* for the whole sequence and with *Dynamic P-values* for each sub-sequence. For the first case we calculated the average speed-up and injected the P-value $P = 0.53$ according to Eq. 6. For the dynamic case we injected $P = 0.0$ on the serial phase and $P = 1.0$ on the parallel phase. Figure 12 shows three

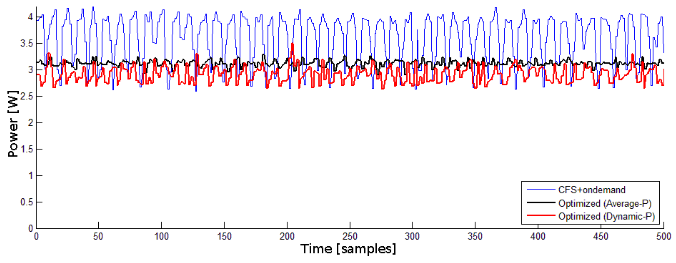


Fig. 12. Power trace from the mixed parallel application using default CFS and with power optimizer

power traces: The CFS+OD case oscillates heavily as predicted according to the race-to-idle strategy. By using one average P-value the power dissipation becomes more stable and is on average significantly lower than the CFS+OD case. By further fine tuning the application with dynamic P-values, the power optimizer is able to better scale the hardware according to the different program phases. The optimizer increases the clock frequency and shuts down cores during the serialized phase, and enables the cores during the parallel phase and decreases the clock frequency.

We also mapped the mixed application to a single thread in order to illustrate the power savings of using parallel hardware. Figure 13 shows a rather steady power trace when mapping both the DC cancellation filter and the Sobel filter on the same core. Both the optimized case and the ondemand case show a higher average power dissipation than the partly parallel case (in Figure 12) since the CPU is forced to run on the higher clock frequencies.

Table II shows the total energy consumption for all use-cases and the energy savings by using the optimizer in the last row. The energy reductions is the result of allowing applications to better express intentions and behavior. By fine tuning the application to use dynamic P-values, the energy consumption can be further decreased as the optimizer is able to scale the hardware more close to the software requirements. The optimized case was at most able to save as much as 19% for executing the same amount of work as the CFS+OD case, which can be considered as significant.

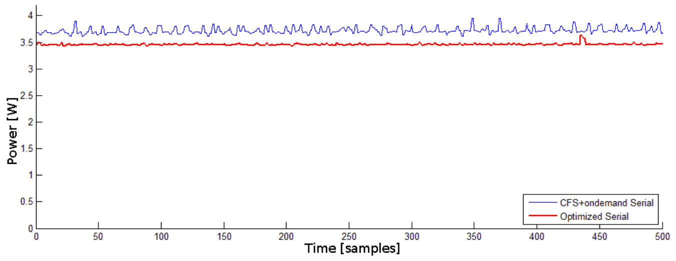


Fig. 13. Power trace from the mixed parallel application using default CFS and with power optimizer running on one thread

TABLE II. ENERGY CONSUMPTION (IN JOULES) FOR A 5 MIN RUN

	Serial	Parallel	Mixed
CFS+ondemand	839.31	735.21	1089.8
Optimized (*avg. P*)	705.03	719.91	936.1
Optimized (*dyn. P*)	n/a	n/a	874.4
Energy savings (avg. P)	16.0%	2.1%	14.1%
Energy savings (dyn. P)	n/a	n/a	19.8%

VII. CONCLUSIONS

Parallelism in software is an important parameter for efficient power management in multi-core systems. It describes the possible utilization of multiple processing elements which determines the relation between dynamic and static power dissipation. Today's power managers do not consider the static power dissipation of enabling cores which becomes more significant as the manufacturing technologies decrease and the amount of cores on a chip increase. To optimize for energy efficiency, applications should be able to express the level of parallelism (P-value) in order to select the appropriate amount of cores to execute on.

We have, in this paper, demonstrated an approach to integrate fast parallel software directly with the power management by injecting QoS guarantees and the P-value into the software as meta data to the power manager. In the presented use-case, the P-values are extracted by a dataflow programming framework, PREESM, and is injected into code segments and used as a parameter in a NLP optimization problem for minimizing total power. With our approach supporting energy efficient programming we can **a)** find the necessary performance required for an application and **b)** allocate resources optimally in multi-core hardware.

REFERENCES

[1] S. Cho and R. Melhem. On the interplay of parallelization, program performance, and energy consumption. Parallel and Distributed Systems, IEEE Transactions on, 21(3):342–353, 2010.

[2] A. Cristea and T. Okamoto. Speed-up opportunities for ann in a time-share parallel environment. In Neural Networks, 1999. IJCNN '99. International Joint Conference on, volume 4, pages 2410–2413 vol.4, 1999.

[3] D. M. E. Lee. Static scheduling of synchronous data-flow programs for digital signal processing. IEEE Transactions on Computers, pages 24–35, 1987.

[4] P. E. Gill, W. Murray, Michael, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. SIAM Journal on Optimization, 12:979–1006, 1997.

[5] F. Hällis, S. Holmbacka, W. Lund, R. Slotte, S. Lafond, and J. Lilius. Thermal influence on the energy efficiency of workload consolidation in many-core architectures. In Digital Communications - Green ICT (TIWDC), 2013 24th Tyrrhenian International Workshop on, pages 1–6, 2013.

[6] M. Haque, H. Aydin, and D. Zhu. Energy-aware task replication to manage reliability for periodic real-time applications on multicore platforms. In Green Computing Conference (IGCC), 2013 International, pages 1–11, 2013.

[7] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application heartbeats for software performance and health. SIGPLAN Not., 45(5):347–348, Jan. 2010.

[8] S. Holmbacka, D. Agren, S. Lafond, and J. Lilius. Qos manager for energy efficient many-core operating systems. In Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on, pages 318–322, 2013.

[9] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. Power optimization of variable voltage core-based systems. In Design Automation Conference, 1998. Proceedings, pages 176–181, 1998.

[10] K. Iondry. Iterative Methods for Optimization. Society for Industrial and Applied Mathematics, 1999.

[11] S. Jafri, M. Tajammul, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen. Energy-aware-task-parallelism for efficient dynamic voltage, and frequency scaling, in cgras. In Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on, pages 104–112, 2013.

[12] N. Khalid, S. Ahmad, N. Noor, A. Fadzil, and M. Taib. Parallel approach of sobel edge detector on multicore platform. International Journal of Computers and Communications Issue, 4:236–244, 2011.

[13] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. Computer, 36(12):68–75, 2003.

[14] N. Mastronarde, K. Kanoun, D. Atienza, P. Frossard, and M. van der Schaar. Markov decision process based energy-efficient on-line scheduling for slice-parallel video decoders on multicore systems. Multimedia, IEEE Transactions on, 15(2):268–278, 2013.

[15] A. M'zah and O. Hammami. Parallel programming and speed up evaluation of a noc 2-ary 4-fly. In Microelectronics (ICM), 2010 International Conference on, pages 156–159, Dec 2010.

[16] M. Pelcat, J. Piat, M. Wipliez, S. Aridhi, and J.-F. Nezan. An open framework for rapid prototyping of signal processing applications. EURASIP journal on embedded systems, 2009:11, 2009.

[17] T. Rauber and G. Runger. Energy-aware execution of fork-join-based task parallelism. In Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on, pages 231–240, 2012.

[18] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. Adagio: Making dvs practical for complex hpc applications. In Proceedings of the 23rd International Conference on Supercomputing, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM.

[19] M. Sadri, A. Bartolini, and L. Benini. Single-chip cloud computer thermal model. In Thermal Investigations of ICs and Systems (THERMINIC), 2011 17th International Workshop on, pages 1–6, 2011.

[20] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. Micro, IEEE, 23(6):84–93, Nov 2003.

[21] I. Takouna, W. Dawoud, and C. Meinel. Accurate mutlicore processor power models for power-aware resource management. In Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on, pages 419–426, 2011.

[22] C. Truchet, F. Richoux, and P. Codognet. Prediction of parallel speed-ups for las vegas algorithms. In Parallel Processing (ICPP), 2013 42nd International Conference on, pages 160–169, Oct 2013.

[23] C.-L. Yang, H.-W. Tseng, and C.-C. Ho. Smart cache: an energy-efficient d-cache for a software mpeg-2 video decoder. In Information, Communications and Signal Processing, 2003 and Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on, volume 3, pages 1660–1664 vol.3, 2003.

[24] D. Zhi-bo, C. Yun, and C. Ai-dong. The impact of the clock frequency on the power analysis attacks. In Internet Technology and Applications (iTAP), 2011 International Conference on, pages 1–4, 2011.

Paper VII

Accurate Energy Modelling for Many-Core Static Schedules

Simon Holmbacka, Jörg Keller, Patrick Eitschberger, Johan Lilius

Originally published *Proceedings of the 23rd International Euromicro Conference on Parallel, Distributed and Network-based Processing, 2015, IEEE International Conference*, pages 525–532. Turku, Finland

Accurate Energy Modelling for Many-Core Static Schedules

Simon Holmbacka*, Jörg Keller†, Patrick Eitschberger† and Johan Lilius*
*Department of Information Technologies, Åbo Akademi University, Turku, Finland
Email: firstname.lastname@abo.fi
†Faculty of Mathematics and Computer Science, FernUniversität in Hagen, Hagen, Germany
Email: firstname.lastname@fernuni-hagen.de

Abstract—**Static schedules can be a preferable alternative for applications with timing requirements and predictable behavior since the processing resources can be more precisely allocated for the given workload. Unused resources are handled by power management systems to either scale down or shut off parts of the chip to save energy. In order to efficiently implement power management, especially in many-core systems, an accurate model is important in order to make the appropriate power management decisions at the right time. For making correct decisions, practical issues such as latency for controlling the power saving techniques should be considered when deriving the system model, especially for fine timing granularity. In this paper we present an accurate energy model for many-core systems which includes switching latency of modern power saving techniques. The model is used when calculating an optimal static schedule for many-core task execution on systems with dynamic frequency levels and sleep state mechanisms. We create the model parameters for an embedded processor, and we validate it in practice with synthetic benchmarks on real hardware.**

I. Introduction

Computer systems with timing guarantees on the applications often pursue two conflicting goals: meeting the deadlines and minimizing the consumed energy. Execution speed is usually optimized by the programmer and compiler while minimizing energy is often left to the operating system which employs Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM) using sleep states. However, an operating system with a dynamic scheduler has no knowledge about the application, its behavior and its timeline. In practice, the power management for dynamic schedules is performed with respect only to the workload level, which does not describe performance requirements.

For applications consisting of a set of tasks with a predictable behavior and a known execution deadline, a schedule with the information when to execute which task at which speed can be devised at compile time (i.e. a static schedule). With hints from the application, the power management techniques can more precisely scale the hardware according to the software performance demands, and energy is minimized by eliminating unnecessary resource allocation. However, Power management is a practical interplay between software algorithms and physical hardware actions. This means that accessing power management techniques in general purpose operating systems introduces practical shortcomings such as access latency. Two separate mechanisms – DVFS and DPM – are currently used for minimizing the CPU power dissipation. As DVFS regulates voltage and frequency to minimize the dynamic power, DPM is used to switch off parts of the CPU to minimize the rapidly growing static power [12]. The techniques are therefore complementing each other and a minimal energy consumption is achieved by proper coordination of both techniques [1], [19]. While both mechanisms have been evaluated in the literature [7], [13], no work has been done to determine the practical latency of both DVFS and DPM on a single platform, and its impact on power management.

In this work we present an accurate energy model for static schedules in many-core systems using DVFS and DPM. The model is based on real hardware measurements to conform with complete platform details and a realistic view of the static and dynamic power balance. We account for the latency of using DVFS and DPM on a statically scheduled many-core system by including the timings in the decision making process of power management techniques. The model is able to accurately forecast the energy consumption of a selected static schedule under different workload configurations and different deadlines. We validate the results with an implemented benchmark framework on real hardware running an unmodified Linux OS.

II. Related Work

DVFS and its efficiency for multi-cores has been studied in the past [7], [13], but mostly the focus has been put directly on measuring the overhead of physically switching hardware states [13], [21] including PLL locking, voltage level switching etc. Mazouz et al. present in [20] a frequency transition latency estimator called FTaLaT, which chooses a frequency depending on the current phase of a program. They argue that programs mostly have either CPU intensive phases in which the CPU is running on a high clock frequency or memory intensive phases in which the clock frequency can be decreased to save power. For very small memory intensive regions, it is favorable to ignore the frequency scaling because the switching delay would be higher than the length of the memory phase. They evaluate their estimator with a few micro-benchmarks (based on OpenMP) on different Intel machines, and they show that the transition latency varies between 20 and 70 microseconds depending on the machine. As the total switching latency is the sum of both hardware and software mechanisms, we study in this paper the practical aspects of switching latency in both DVFS and DPM for off-the-shelf operating systems running on real hardware. Influences of user space interaction and the kernel threads which control the power saving mechanisms are studied, and related to the effects on the energy consumption.

In the paper of Schöne et al. [23] the implementation of the low-power states in current x86 processors are described. The wake-up latencies of various low-power states are measured and the results are compared with the vendor's specifications that are exposed to the operating system. The results show fluctuations e.g. depending on the location of the callee processor. Their work complements ours, but rather than using the x86 architecture we focus on mobile ARM processors with less support for hardware power management.

Algorithms for minimizing energy based on power and execution time have been presented in previous work such as [3], [7], [8]. Cho et al. define an analytical algorithm for expressing dynamic and static power in a multi-core system with multiple frequency levels. The minimum-energy-operation point is then calculated by determining the first order derivative of the system energy with respect to time. The mathematical expression defined in [3] exploits the task parallelism in the system to determine the amount of processing elements required, and hence influencing the static power dissipation. In our work, we define the system power model based on experiments on real hardware rather than analytical expressions in order to tailor the model closer to real-world devices.

In [16] an Energy-Aware Modeling and Optimization Methodology (E-AMOM) framework is presented. It is used to develop models of runtime and power consumption with the help of performance counters. These models are used to reduce the energy by optimizing the execution time and power consumption with focus on HPC systems and scientific applications. Our approach follows the same principle, but instead we use a top-down power model based on real experiments rather than analytical expressions. We also account for the latency of both DVFS and DPM which, as explained, becomes important when the time scale is shrinking.

While acknowledging that DVFS and DPM are possible energy savers in data centers [4], [15], [10], our work focus on core level granularity with a smaller time scale and our measurements are based on the per-core sleep state mechanism rather than suspension to RAM or CPU hibernation. Aside from the mentioned differences, none of the previous work deals with latency overhead for both DVFS and DPM on a practical level from the operating system's point of view. Without this information, it is difficult to determine the additional penalty regarding energy and performance for using power management on modern multi-core hardware using an off-the-shelf OS such as Linux.

III. Power Distribution and Latency of Power-Saving Mechanisms

Power saving techniques in microprocessors are hardware-software coordinated mechanisms used to scale up or down parts of the CPU dynamically during runtime. We outline the functionalities and current implementation in the Linux kernel to display the obstacles of practically using power management.

A. Dynamic Voltage and Frequency Scaling (DVFS)

The DVFS functionality was integrated into microprocessors to lower the dynamic power dissipation of a CPU by scaling the clock frequency and the chip voltage. Equation 1 shows the simple relation of these characteristics and the dynamic power

$$P_{dynamic} = C \cdot f \cdot V^2 \tag{1}$$

where C is the effective charging capacitance of the CPU, f is the clock frequency and V is the CPU supply voltage. Since DVFS reduces both the frequency and voltage of the chip (which is squared), the power savings are more significant when used on *high* clock frequencies [22], [25].

The relation between frequency and voltage is usually stored in a hardware specific look-up table from which the OS retrieves the values as the DVFS functionality is utilized. Since the clock frequency switching involves both hardware and software actions, we investigated the procedure in more detail to pinpoint the source of the latency. In a Linux based system the following core procedure describes how the clock frequency is scaled:

1) A change in frequency is requested by the user
2) A mutex is taken to prevent other threads from changing the frequency
3) Platform-specific routines are called from the generic interface
4) The PLL is switched out to a temporary MPLL source
5) A safe voltage level for the new clock frequency is selected
6) New values for clock divider and PLL are written to registers
7) The mutex is given back and the system returns to normal operation

1) DVFS implementations: To adjust the DVFS settings, the Linux kernel uses a frequency governor [11] to select, during run-time, the most appropriate frequency based on a set of policies. In order to not be affected by the governors, we selected the `userspace` governor for application-controlled DVFS. The DVFS functionality can be accessed either by directly writing to the `sysfs` interface or by using the system calls. By using the `sysfs`, the DVFS procedure includes file management which is expected to introduce more overhead than calling the kernel headers directly from the application. We studied, however, both options in order to validate the latency differences between the user space interface and the system call.

a) System call interface: The system call interface for DVFS under Linux is accessible directly in the Linux kernel. We measured the elapsed time between issuing the DVFS system call and the return of the call which indicates a change in clock frequency. Listing 1 outlines the pseudo code for accessing the DVFS functionality from the system call interface.

```
#include <cpufreq.h>
#include <sys/time.h>
latency_syscall(){
  gettimeofday(&time1);
  cpufreq_set_frequency(Core0, FREQ1);
  gettimeofday(&time2);
  cpufreq_set_frequency(Core0, FREQ2);
  gettimeofday(&time3);
}
```

Listing 1. Pseudo code for measuring DVFS latency using system calls

b) User space interface: The second option is to use the `sysfs` interface for accessing the DVFS functionality from user space. The CPU clock frequency is altered by writing the setpoint frequency into a `sysfs` file, which is read and consequently used to change the frequency. The kernel functionality is not directly called from the c-program, but file system I/O is required for both reads and writes to the `sysfs` filesystem. Listing 2 outlines an example for the DVFS call via the `sysfs` interface.

```
#include <sys/time.h>
latency_sysfs(){
  gettimeofday(&time1);
  system("echo FREQ1 > /sys/devices/system/cpu/cpu0/cpufreq/
       scaling_setspeed");
  gettimeofday(&time2);
  system("echo FREQ2 > /sys/devices/system/cpu/cpu0/cpufreq/
       scaling_setspeed");
  gettimeofday(&time3);
}
```

Listing 2. Pseudo code for measuring DVFS latency using `sysfs`

2) DVFS Measurement results: The user space and the kernel space mechanisms were evaluated, and the results are presented in this section. Since the DVFS mechanism is ultimately executed on kernel- and user space threads, we stressed the running system with different load levels to evaluate its impact. For this purpose, we used Spurg-Bench [18] to generate defined load levels on a set of threads executing floating point multiplications. All latency measurements were executed in a loop of 100 iterations with different frequency hops, and with a timing granularity of microseconds. We used an Exynos 4412 SoC with an ARM core capable of clock speeds in the range from 200 to 1600 MHz.

Figure 1 shows the average latency for all load levels and with frequency hops from 1600 to 200 MHz and from 200 to 1600 MHz. When using the system call interface, the average latency decreases slightly when increasing the load (left part of Figure 1). On the other hand, the switching latency has a strong correlation to current frequency and target frequency in the `sysfs` implementation. The measurements of the `sysfs` interface show a latency increase until the load is roughly 60% after which it slightly declines and finally increases when stressing the CPU to 100%. As expected, the latency is shorter as the CPU frequency jumps from 1600 to 200 MHz because most of the DVFS procedure (including the file system call) is executed on the higher frequency. Table I shows the standard deviation from samples in the same experiments. The `sysfs` experiments show a much higher standard deviation because of filesystem I/O when accessing the `sysfs` filesystem.

TABLE I. STANDARD DEVIATION OF DVFS LATENCY USING SYSTEM CALL AND SYSFS

Load	**0%**	**25%**	**50%**	**75%**	**90%**	**100%**
1600-200MHz						
System call	2%	5%	5%	6%	6%	8%
`sysfs`	27%	28%	40%	26%	20%	30%
200-1600MHz						
System call	3%	6%	6%	8%	6%	6%
`sysfs`	25%	30%	34%	39%	29%	25%

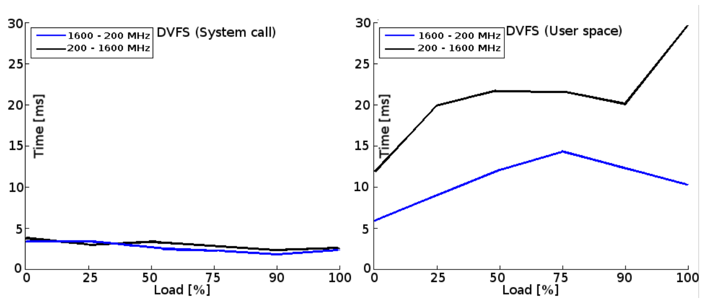


Fig. 1. Average latency for changing clock frequency under different load conditions using system call and `sysfs` mechanisms

B. Dynamic Power Management (DPM)

In older generation microprocessors, most of the power was dissipated by switching activities in the chip (dynamic power). In recent years, the static power has, on the other hand, become more dominant [12], and is even expected to dominate the total power dissipation in next generation microprocessors [17]. The static power is dissipated due to leakage currents through transistors, which is mostly a result of subthreshold and gate-oxide leakage [14]. Equation 2 shows the subthreshold leakage current

$$I_{sub} = K_1 \cdot W \cdot e^{-V_{th}/n \cdot V_\theta}(1 - e^{-V/V_\theta}) \qquad (2)$$

where K_1 and n are architecture specific constants, W is the gate width and V_θ is the thermal voltage. Hence, the silicon temperature causes an exponential increase in leakage currents [6]. Moreover, when lowering the supply voltage of integrated circuits, the subthreshold leakage current increases which also increases the dissipated static power [2], [24]. The leakage current is present as long as the chip (or parts of the chip) is connected to a power source. This means that in order to reduce the leakage current, parts of the chip must be disconnected from the power source and re-connected as the functionality is required again.

CPU sleep states (or DPM) is used to disable parts of the CPU on demand to decrease the static power consumption. The functionality is accessed in Linux by the CPU hotplug facilities, which was originally implemented to replace physical CPUs during run-time. On our test platform the hotplug functionality places the core in a Wait For Interrupt (`wfi`) state in which the core clock is shut down, and re-activated as soon as the core receives an interrupt from another core. The functionality of hotplugging a core differs depending on the state of the core designated to be turned off. In case the core is executing workload, the mechanism re-allocates the workload to another core in order to make it idle. In case the core is idle this action is not required. The hotplug functionality can be accessed in Linux either as a kernel space module or directly from user space using the `sysfs` interface.

The hotplug implementation consists of a platform-independent part and a platform-specific part, which lastly calls the CPU specific assembly routines for accessing the hotplug. The following procedure describes how the hotplug mechanism is used to shut down a core:

1) A core shutdown command is issued in the system
2) The system locks the core with a mutex in order to block

tasks from being scheduled to this core
3) A *notification* is sent to the kernel:
CPU_DOWN_PREPARE
4) A kernel thread executes a callback function and receives the notification
5) Tasks are migrated away from the core being shut down
6) A *notification* is sent to the kernel: CPU_DEAD
7) A kernel thread executes a callback function and receives the notification
8) Interrupts to the core are disabled, the cache is flushed and the cache coherency is turned off
9) The power source is removed and core is physically shut down

As seen from the procedure, the shutdown of a core is reliant on callback functionalities in the Linux kernel, which means that the system performance and current utilization will affect the response time of the scheduled kernel thread issuing this functionality. As suggested in [5], improvements can be made to decrease the hotplug latency but the current main stream kernels still rely on the aforementioned callback facilities. The wake-up procedure is, similarly to the shutdown procedure, dependent on callbacks but with an inter-core interrupt to trigger the core startup.

1) CPU hotplug implementations: Two separate experiments were conducted to determine the latency of CPU hotplug: kernel space and user space implementations. Similarly to the DVFS measurements, we measured the elapsed time between issuing the shutdown/wake-up call and the return of the call.

a) Kernel space module: In the first implementation we accessed the CPU hotplug functionality directly in a kernel module which Linux executes in kernel space with closer access to the hardware. Listing 3 outlines the functionality of accessing the CPU hotplug in kernel space.

```
#include <linux/cpu.h>
#include <linux/time.h>
latency_kernel(){
  mutex_lock(&lock);
  do_gettimeofday(&time1);
  cpu_down(1); //Core 1 is shut down
  do_gettimeofday(&time2);
  mutex_unlock(&lock);
  mutex_lock(&lock);
  do_gettimeofday(&time3);
  cpu_up(1); //Core 1 is waken up
  do_gettimeofday(&time4);
  mutex_unlock(&lock);
}
```

Listing 3. Pseudo code for measuring hotplug latency in kernel module

b) User space interface: The second mechanism for accessing the CPU hotplug functionality was implemented as a normal user space application accessing `sysfs` files. The benefit of using the user space functionality rather than the kernel space is a significantly simpler implementation and misbehavior in user space will be intercepted safely by the kernel rather than causing system crashes. The downside is an expected higher latency for accessing the hardware due to file system I/O and kernel space switches. Listing 4 outlines the functionality of accessing the CPU hotplug in user space.

```
#include <sys/time.h>
latency_user(){
  gettimeofday(&time1);
  system("echo 0 > /sys/devices/system/cpu/cpu1/online");
  gettimeofday(&time2);
  system("echo 1 > /sys/devices/system/cpu/cpu1/online");
  gettimeofday(&time3);
}
```

Listing 4. Pseudo code for measuring hotplug latency in user space

2) CPU hotplug results: Similarly to the DVFS experiments, we stressed the system with different load levels using Spurg-Bench. The system was running on a selected range of clock frequencies and the timings were measured on microsecond granularity.

Figure 2 shows the average latency for shutting down a core in kernel- and user space respectively. The axes of the figures have been fixed in order to easily compare the different configurations and implementations. From Figure 2 it is clear that the average latency for shutting down a core is rather constant in kernel space and not significantly dependent on clock frequency. The user space implementation is more dependent on the load level as the latency doubles between 0% load and 100% load.

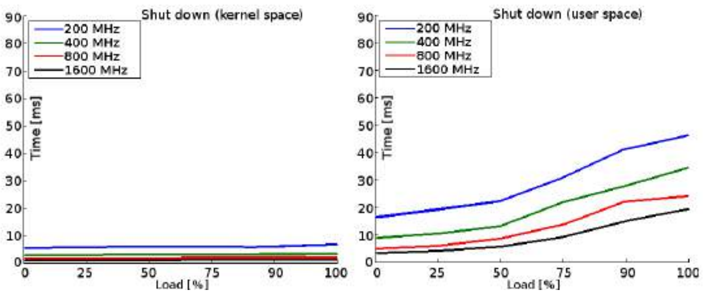


Fig. 2. Average latency for shutting down a core under different load conditions using kernel and userspace mechanisms

On the contrary, the wake-up time is dependent on the load level in both the kernel space and the user space case. As seen in Figure 3 (left), the kernel space implementation measures up to 6x higher latency for the 100% load case compared to the 0% load case. A similar ratio is seen in the user space implementation, but the latency is on average roughly 2x higher than the kernel space implementation. Similarly to the shutdown procedure, the wake-up is also dependent on several kernel notifications followed by kernel callbacks. An additional factor in waking up a core is the long process of creation and initialization of kernel threads (kthreads), which are required to start the idle loop on a newly woken-up core. Only after the kthread is running, the CPU mask for setting a core available can be set.

Since this work focus on user space implementations with static scheduling, we chose to access the hotplug functionality from the `sysfs` interface in our evaluation. However we acknowledge that a more optimized solution is possible by embedding parts of the functionality in kernel space. Table II shows the standard deviation for shutdown and wake-up from the experiments.

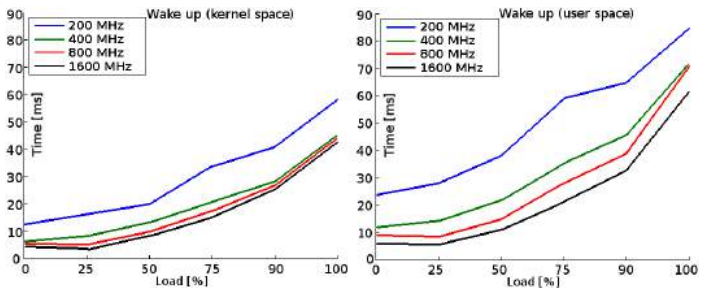


Fig. 3. Average latency for waking up a core under different load conditions using kernel and userspace mechanisms

TABLE II. STANDARD DEVIATION OF DPM LATENCY IN KERNEL- AND USERSPACE

Load	**0%**	**25%**	**50%**	**75%**	**90%**	**100%**
Shut-down						
Kernelspace	3%	8%	9%	11%	5%	9%
userspace	3%	9%	11%	12%	16%	15%
Wake-up						
Kernelspace	7%	20%	26%	27%	23%	4%
userspace	8%	13%	18%	17%	18%	28%

IV. ENERGY MODEL

An energy model is used to predict the energy consumption of the system using a set of fixed parameters. The input to our model are descriptions of the workload and the target architecture. The workload is represented by the number of instructions w to be executed, and the deadline D before which the workload must be processed as illustrated in Figure 4. In case the workload is processed sufficiently prior to the deadline, the cores can be shut down or scaled down with a given overhead penalty.

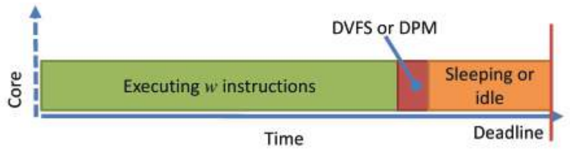


Fig. 4. Execution of workload before a given deadline

As we target compute-bound applications, we do not have to care for long I/O latencies and thus the time t to process the workload is considered inversely proportional to the core frequency. For simplicity we assume the workload being is inversely proportional to the core frequency, with a constant of 1, i.e. one instruction per cycle is processed on average. Thus:

$$t(w, f) = w/f$$

Let t_{min} be the time to process the workload at maximum possible speed, we then call $D/t_{min} \geq 1$ the pressure. If the pressure is close to 1, the CPU must run on maximum speed to meet the deadline. If the pressure is e.g. 2, the CPU can either run at half the maximum speed for time D, or run at maximum speed until time $D/2$, and then idle or shut down the cores until the deadline. We assume a workload consisting of a large number of small, independent tasks, so that the workload can be balanced among the cores.

The target architecture is a many-core CPU represented by p number of cores with frequencies $f_1, \ldots, f_k$, together with the power consumption of the chip at each frequency in idle mode and under load, i.e. $P_{idle}(j, f_i)$ and $P_{load}(j, f_i)$, where $1 \leq j \leq p$ denotes the number of active cores. As we target compute-bound applications, we assume that the cores are stressed to 100% load, however, extensions for power consumptions at different load levels are possible. We further assume that all cores run at the same frequency, because that is a feature of our target architecture. We will explain at the end of the section how to extend the model to architectures with different frequencies on different cores.

From previous sections we have obtained the latency t_{scale} of switching frequencies of the cores (the cores are assumed to be idle) from f_{i1} to f_{i2}, the power consumption during this time is assumed to be the average of $P_{idle}(f_{i1})$ and $P_{idle}(f_{i2})$. While t_{scale} might vary depending on the frequencies, the model confines it to an average value as a close enough approximation. An idle core at frequency f_i can be shut down, and later wake up again. We consider these consecutive events, as the cores must be available after the deadline. The total time for shutdown and wake-up is denoted by $t_{shut}(f_i)$, we assume that the core consumes idle power during this time.

If each of the j active cores, where $1 \leq j \leq p$ has to process the same workload w/j until the deadline, the cores must run at least at frequency $f_{min} = (w/j)/D$. Hence they can utilize any $f_i \geq f_{min}$ to process the workload in time $t_i = (w/j)/f_i$.

There are several possibilities to process the workload w: **1)** For any number j of active cores, the cores can run at any frequency $f_i \geq f_{min}$ for time t_i to process the workload and then idle at the same frequency for time $D - t_i$ consuming total energy:

$$E_1(j, f_i) = t_i \cdot P_{load}(j, f_i) + (D - t_i) \cdot P_{idle}(j, f_i)$$

2) The idle cores could also be scaled down to the lowest frequency f_1 if $D - t_i$ is larger than t_{scale}, with a resulting energy consumption of

$$E_2(j, f_i) = t_i \cdot P_{load}(j, f_i) + t_{scale} \cdot P_{idle}(j, f_i) + \\ (D - t_i - t_{scale}) \cdot P_{idle}(j, f_1)$$

3) Finally, the cores could be shut down after processing the workload, and wake up just before the deadline, if $D - t_i \geq t_{shut}(f_i)$. In our target architecture, the cores must be shut down in sequence, and the first core must remain active idle to wake the others up. However, it would be easy to extend the model for other platforms. The consumed energy can be modeled as:

$$E_3(j, f_i) = t_i \cdot P_{load}(j, f_i) + \sum_{l=2}^{j} t_{shut} \cdot P_{idle}(l, f_i) + \\ (D - t_i - (j-1)t_{shut}) \cdot P_{idle}(1, f_1)$$

Having formulated the model, and given a concrete workload, we enumerate all feasible solutions prior to execution, and choose the one with the lowest energy consumption. Hence we create a static schedule, i.e. a balanced mapping of the tasks onto the cores together with information about core speeds and necessary frequency scalings or shutdowns. If the target architecture has some other characteristics such as concurrent shutdown of cores, another energy formula can be adapted, and the number of solutions might increase. However, the core algorithm design remains, as the number of feasible solutions is still small enough for enumeration. The model can also be refined to scale the frequency of the cores prior to shutdown to a frequency level with a more favorable shutdown time.

In contrast to an analytic model, we do not have to make assumptions of convexity and continuity of power functions etc., which are often not true in practice, as well as the distinction between idle power and power under load. Yet, the model still uses optimizations, such as a non-decreasing power function with respect to frequency and number of active cores. For example, we do not scale frequencies while processing workload.

A. *Model based simulation*

We scheduled task sets with 10k, 100k and 1M synthetic jobs with pressure levels 1.1, 1.3, 1.5 and 4.0 in order to determine the *best*, *2nd best* and *worst* energy efficient configuration parameters. The number of instructions of each job were randomly chosen to obtain a [0;500ms] runtime normalized to the highest clock frequency. The system parameters were obtained from the quad-core Exynos 4412 platform by measuring the power under workload P_{load} (Table III) and the idle power P_{idle} (Table IV) for all frequency levels from 1 to 4 active cores.

TABLE III. POWER DISSIPATION (IN WATTS) FOR THE EXYNOS 4412 UNDER FULL WORKLOAD. COLUMNS ARE THE NUMBER OF ACTIVE CORES AND ROWS ARE CLOCK FREQUENCY

	200	**400**	**600**	**800**	**1000**	**1200**	**1400**	**1600**
1	2.875	3.02	3.095	3.16	3.315	3.43	3.675	3.955
2	2.975	3.125	3.275	3.375	3.55	3.775	4.22	4.715
3	3.045	3.305	3.45	3.65	3.85	4.225	4.935	5.71
4	3.105	3.365	3.6	3.845	4.185	4.745	5.795	7.615

TABLE IV. POWER DISSIPATION (IN WATTS) FOR THE EXYNOS 4412 UNDER IDLE WORKLOAD. COLUMNS ARE THE NUMBER OF ACTIVE CORES AND ROWS ARE CLOCK FREQUENCY

	200	**400**	**600**	**800**	**1000**	**1200**	**1400**	**1600**
1	2.148	2.162	2.173	2.139	2.048	2.035	2.143	2.284
2	2.152	2.163	2.179	2.133	2.11	2.057	2.202	2.381
3	2.156	2.167	2.183	2.146	2.122	2.08	2.279	2.407
4	2.158	2.173	2.181	2.155	2.172	2.105	2.33	2.503

The scheduler used four threads for execution, which models a scalability up to four cores. We executed the scheduler with different parameters, and the output shows the possible scheduling configurations which meet the deadline. Table V shows the configuration settings for three chosen outputs: best, 2nd best and worst energy efficiency with the aforementioned power values and scheduling parameters. The output is a working frequency combined with a power management feature: *DPM*, *DVFS* or *idling* the whole slack time.

Since a pressure of 1.1 poses a very tight deadline for the jobs, the only feasible clock frequency setting is 1600 MHz. The best case with this parameter uses rather DVFS than DPM because of a faster switching time which costs less energy. For pressure levels > 1.1 a more relaxed deadline allows slower execution speed, which impacts significantly on the dynamic power dissipation. Hence, the best case uses DPM rather than DVFS since the more relaxed deadline allows a longer sleep time, which reduces the energy consumption more than the cost of activating the DPM mechanism. Finally the pressure level 4.0 – with a very relaxed deadline – allows the system to execute on 1000 MHz, which is the most energy efficient clock frequency. Because of the execution model illustrated in Figure 4, the number of jobs does not affect the usage of the power management techniques, since the clock frequency scaling or core shutdown is always executed only once after the workload finishes.

TABLE V. CONFIGURATION PARAMETERS FOR DIFFERENT NUMBER OF JOBS AND DIFFERENT PRESSURE LEVELS

			Number of jobs		
		Config	10k	100k	1M
Pressure	1.1	Best	1600MHz+DVFS	1600MHz+DVFS	1600MHz+DVFS
		2nd Best	1600MHz+DPM	1600MHz+DPM	1600MHz+DPM
		Worst	1600MHz+IDLE	1600MHz+IDLE	1600MHz+IDLE
	1.3	Best	1400MHz+DPM	1400MHz+DPM	1400MHz+DPM
		2nd Best	1400MHz+DVFS	1400MHz+DVFS	1400MHz+DVFS
		Worst	1600MHz+IDLE	1600MHz+IDLE	1600MHz+IDLE
	1.5	Best	1200MHz+DPM	1200MHz+DPM	1200MHz+DPM
		2nd Best	1200MHz+IDLE	1200MHz+IDLE	1200MHz+IDLE
		Worst	1600MHz+IDLE	1600MHz+IDLE	1600MHz+IDLE
	4.0	Best	1000MHz+DPM	1000MHz+DPM	1000MHz+DPM
		2nd Best	1000MHz+DVFS	1000MHz+DVFS	1000MHz+DVFS
		Worst	1600MHz+IDLE	1600MHz+IDLE	1600MHz+IDLE

Finally, in Table V we only list configuration using all four cores since it proved most energy efficient using all settings.

V. REAL-WORLD VALIDATION

To validate the model presented in Section IV we executed real-world experiments to compare the final energy consumption with the mathematical representation.

A. *Experimental setup*

We replicated the scenarios described in Section IV by constructing experiments with: **a)** a set of configuration parameters used to time trigger the hardware power saving features according to the scheduler results **b)** one or more benchmark threads executing a selected job for a given time. In one example configuration, the benchmark executes on four threads on 1600 MHz for n milliseconds after which the clock frequency is reduced to 200 MHz for m milliseconds or the cores are shut down until the deadline. We chose `stress` as the benchmark since it was used to train the mathematical model and its behavior is easily repeatable. `stress` is also a good candidate for representing CPU intensive workloads.

We used the quad-core Exynos 4412 platform on which the experiments were earlier conducted. In order to measure the power of the Exynos board and to not disturb its performance, we added an external Raspberry Pi board to monitor the board power consumption similar to the work in [9]. Figure 5 shows the complete workflow for time synchronization and power measurements:

1) Raspberry Pi sends a start signal over UDP to Exynos
2) Raspberry Pi starts to log the power measured by an INA226 A/D converter connected to its i2c bus
3) Exynos starts the benchmark threads
4) Exynos scales frequency or shuts down cores if requested
5) Exynos finishes benchmark and sends stop signal over UDP
6) Raspberry Pi ends power monitor

In order to get an average power log as accurate as possible, the additional overhead including 2 ms ping latency between the Raspberry Pi and the Exynos was accounted for and excluded in the power monitor.

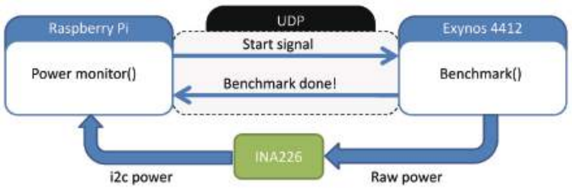


Fig. 5. Raspberry Pi connected to Exynos board with A/D converter to measure power and send data to Raspberry-Pi

B. Experimental Results

We used the task sets with 10k, 100k and 1M jobs from the previous section. The respective execution times for executing the jobs were measured and used in the benchmark framework. Furthermore, we also evaluated pressure levels: 1.1, 1.3, 1.5 and 4.0 for all task sets. For each combination of task set and pressure levels the *best case*, *2nd best case* and *worst case* energy scenarios were compared against the mathematical model.

Figure 6 shows results for 10k jobs. Both the best case

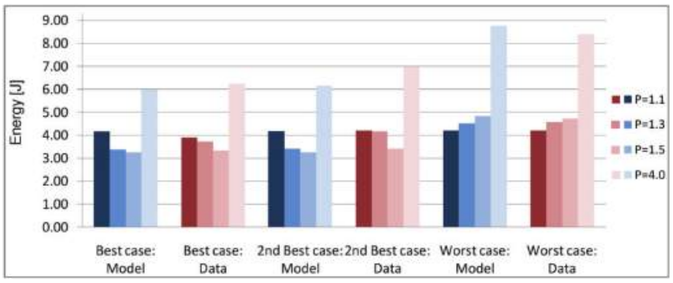


Fig. 6. Energy consumption for model and data running 10k jobs with different pressure settings

model and data show a high energy consumption for low pressure and for high pressure; the lowest energy consumption is achieved at P=1.5 for both data and model. This is a result of low pressure levels pushing the deadline very close and the CPU is hence forced to execute on a high clock frequency to meet the deadline. Even though the execution time is short, the dynamic power dissipated when executing on the highest frequency results in high energy consumption. Large values for pressure also result in high energy consumption since deadline and the execution time becomes very long and the ever present static power significantly increases the energy consumed even though the clock frequency is low.

Figure 7 shows the results of the benchmarks with 100k jobs. The relation between data and model are rather similar to Figure 6 with the exception of pressure level P=1.1. This case has a higher prediction than the actual measured data. As previously explained, a low pressure level forces a high clock frequency – and running the CPU on the maximum frequency for a long time activates the thermal throttling of the CPU as it reaches a critical temperature. The CPU is then temporarily

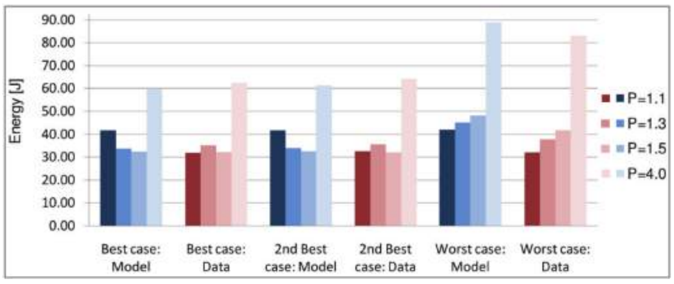


Fig. 7. Energy consumption for model and data running 100k jobs with different pressure settings

stalled resulting in lower power dissipation, which leads to low energy consumption. Naturally by using CPU throttling, fewer operations are executed in the benchmark which causes poor performance. The situation can be avoided by adding active cooling, but we chose to acknowledge this anomaly when creating mathematical power models of a thermally limited chip.

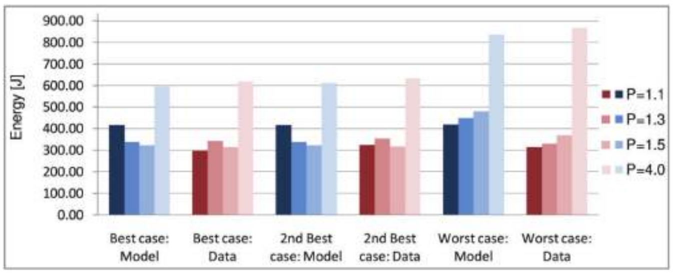


Fig. 8. Energy consumption for model and data running 1M jobs with different pressure settings

Figure 8 shows the results from the longest running experiments, i.e. 1M jobs. Similarly to Figure 7 the thermal throttling of the CPU causes a misprediction of the model.

The mean squared error between data and model is finally shown in Figure 9 for all previously mentioned experiments. The figure shows the largest misprediction in cases with P=1.1 and for long running jobs (1M and 100k). As previously concluded, the misprediction is mostly caused by the CPU thermal throttling activated when running the CPU on maximum frequency for a long time (in range 10s of seconds). Furthermore, the thermal throttling is occasionally also activated when running the CPU on the second highest frequency for a very long time (several minutes) as can be seen in 1M case with P=1.3. Hence, the model remains accurate as

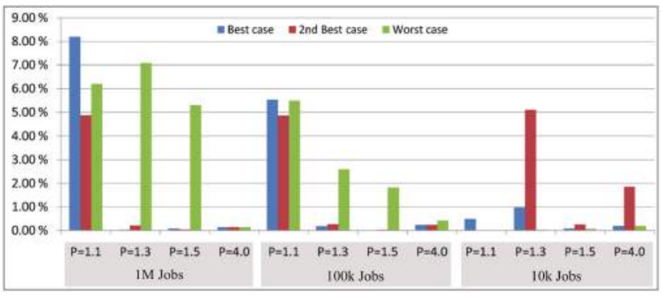


Fig. 9. Error squared between model and data for all task sets and pressures

long as the CPU remains within its thermal bounds.

VI. Conclusions

As the hardware becomes more complex and the manufacturing techniques shrink, accurate power consumption details for multi-core systems is difficult to derive from an analytical mathematical approximation. An alternative is to model the system top-down based on real experiments and include practical aspects such as such as power management overhead, which cannot be ignored for applications with deadlines in the millisecond range. We have presented an energy model derived from real-world power measurements including power management latencies from a general purpose operating system. The model can be used to calculate an energy-optimal static schedule for applications with a given deadline. We also have obtained the practical timing granularity for DVFS and DPM after which the latency of power saving techniques cannot longer be neglected.

We have validated the model with experiments on real hardware and demonstrated its accuracy. Practical anomalies such as critical temperatures can cause inconsistencies in the model and has also been acknowledged as a limitation. In future work, we would like to extend our model to Intel-based multi-core platforms with independent core frequencies, and to heterogeneous platforms such as the big.LITTLE systems. By using core independent frequency levels the model must coordinate both the location of the running task and the clock frequency of the core possibly by defining a dynamic power model. The heterogeneous platform must further also define the type of core which leads to both a dynamic power model and a dynamic performance model of the currently used core type. Moreover, we would like to extend the model to multiple applications, and to applications with partly stochastic behavior.

References

[1] K. Bhatti, C. Belleudy, and M. Auguin. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on, pages 184–191, 2010.

[2] S. Borkar. Design challenges of technology scaling. Micro, IEEE, 19(4):23 –29, jul-aug 1999.

[3] S. Cho and R. Melhem. On the interplay of parallelization, program performance, and energy consumption. Parallel and Distributed Systems, IEEE Transactions on, 21(3):342–353, 2010.

[4] A. Gandhi, M. Harchol-Balter, and M. Kozuch. Are sleep states effective in data centers? In Green Computing Conference (IGCC), 2012 International, pages 1–10, June 2012.

[5] T. Gleixner, P. E. McKenney, and V. Guittot. Cleaning up linux's cpu hotplug for real time and energy management. SIGBED Rev., 9(4):49–52, Nov. 2012.

[6] F. Hällis, S. Holmbacka, W. Lund, R. Slotte, S. Lafond, and J. Lilius. Thermal influence on the energy efficiency of workload consolidation in many-core architecture. In R. Bolla, F. Davoli, P. Tran-Gia, and T. T. Anh, editors, Proceedings of the 24th Tyrrhenian International Workshop on Digital Communications, pages 1–6. IEEE, 2013.

[7] C. Hankendi and A. K. Coskun. Adaptive power and resource management techniques for multi-threaded workloads. In Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, IPDPSW '13, pages 2302–2305, Washington, DC, USA, 2013. IEEE Computer Society.

[8] M. Haque, H. Aydin, and D. Zhu. Energy-aware task replication to manage reliability for periodic real-time applications on multicore platforms. In Green Computing Conference (IGCC), 2013 International, pages 1–11, June 2013.

[9] S. Holmbacka, F. Hällis, W. Lund, S. Lafond, and J. Lilius. Energy and power management, measurement and analysis for multi-core processors. Technical Report 1117, 2014.

[10] T. Horvath and K. Skadron. Multi-mode energy management for multi-tier server clusters. In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT '08, pages 270–279, New York, NY, USA, 2008. ACM.

[11] IBM Corporation. Blueprints: Using the linux cpufreq subsystem for energy management. Technical report, June 2009.

[12] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In Proceedings of the 41st annual Design Automation Conference, DAC '04, pages 275–280, New York, NY, USA, 2004. ACM.

[13] A. Kahng, S. Kang, R. Kumar, and J. Sartori. Enhancing the efficiency of energy-constrained dvfs designs. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 21(10):1769–1782, Oct 2013.

[14] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. Computer, 36(12):68–75, 2003.

[15] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. H. Katz. Napsac: Design and implementation of a power-proportional web cluster. In Proceedings of the First ACM SIGCOMM Workshop on Green Networking, Green Networking '10, pages 15–22, New York, NY, USA, 2010. ACM.

[16] C. Lively, V. Taylor, X. Wu, H.-C. Chang, C.-Y. Su, K. Cameron, S. Moore, and D. Terpstra. E-amom: an energy-aware modeling and optimization methodology for scientific applications. Computer Science - Research and Development, 29(3-4):197–210, 2014.

[17] W. Lockhart. How low can you go? http://chipdesignmag.com/display.php?articleId=3310, 2014.

[18] W. Lund. Spurg-bench: Q&d microbenchmark software. https://github.com/ESLab/spurg-bench, May 2013.

[19] M. Marinoni, M. Bambagini, F. Prosperi, F. Esposito, G. Franchino, L. Santinelli, and G. Buttazzo. Platform-aware bandwidth-oriented energy management algorithm for real-time embedded systems. In ETFA, 2011 IEEE 16th Conference on, pages 1–8, 2011.

[20] A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby. Evaluation of cpu frequency transition latency. Comput. Sci., 29(3-4), Aug. 2014.

[21] J. Park, D. Shin, N. Chang, and M. Pedram. Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors. In Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on, pages 419–424, Aug 2010.

[22] T. Rauber and G. Rünger. Energy-aware execution of fork-join-based task parallelism. In Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on, pages 231–240, 2012.

[23] R. Schöne, D. Molka, and M. Werner. Wake-up latencies for processor idle states on current x86 processors. Computer Science - Research and Development, pages 1–9, 2014.

[24] H. Singh, K. Agarwal, D. Sylvester, and K. Nowka. Enhanced leakage reduction techniques using intermediate strength power gating. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 15(11):1215 –1224, nov. 2007.

[25] D. Zhi-bo, C. Yun, and C. Ai-dong. The impact of the clock frequency on the power analysis attacks. In Internet Technology and Applications (iTAP), 2011 International Conference on, pages 1–4, 2011.

Paper VIII

Performance Monitor Based Power Management for big.LITTLE Platforms

Simon Holmbacka, Sébastien Lafond, Johan Lilius

Originally published *Proceedings of the HIPEAC Workshop on Energy Efficiency with Heterogeneous Computing, 2015*, pages 1–6. Amsterdam, Netherlands

Performance Monitor Based Power Management for big.LITTLE Platforms

Simon Holmbacka, Sébastien Lafond, Johan Lilius
Department of Information Technologies, Åbo Akademi University
20530 Turku, Finland
firstname.lastname@abo.fi

ABSTRACT

Recently new heterogeneous computing architectures, coupling low-power low-end cores with powerful, power-hungry cores, appeared on the market. From a power management point of view, and compared to traditional homogeneous multi-core architectures, such architectures provide one more variable: the core type to map applications on. At the same time conventional power managers drives the DVFS mechanism based on the notion of workload. This means that as long as the CPU is capable of executing work, a workload increase will result in a frequency increase. In practice this results in a Race-to-Idle execution which mostly uses high clock frequencies. In this work we propose a performance monitor based power manager for cluster switched ARM big.LITTLE architectures. The proposed power manager allocates resources based on application performance rather than workload levels, which allows the hardware to adapt closer to software requirements. The presented power manager is capable of saving up to 57% of energy with the addition of one line of c-code in legacy applications.

1. INTRODUCTION

The big.LITTLE architecture [1] using one cluster of high performance cores and one cluster of energy efficient cores is becoming popular in mobile devices such as mobile phones and tablets. The big cores are designed for high performance calculations using high clock frequencies, deep pipelines, large caches and out-of-order execution. In case high performance is not required, the system can shut down the big cores and activate the energy efficient LITTLE cores. The LITTLE cores in e.g. the Exynos 5410 SoC utilizes four A7 cores with short pipelines, small caches and in-order execution, which reduces the power dissipation significantly. The core selection – in this SoC which we consider – is based on cluster switching [12], which enables either the cluster of A7 or A15 cores, but not both types simultaneously. The core types are automatically switched from LITTLE to big as the clock frequency of the CPU is increased beyond a certain threshold.

While the hardware shows a great potential in energy savings, the software is usually unable to utilize such an architecture efficiently. Optimally, the system should not allocate more CPU resources than what the application requires. This aim can be achieved with a power manager monitoring the system and adjusting the clock frequency accordingly. Currently, the power managers controlling CPU resources use only *workload* as the metric for resource allocation [2]. Workload is defined in Linux as the ratio between CPU execution and CPU idle states for a given time window. It is, however, a poor metric for controlling CPU resources because it does not describe application *performance*. When applying high workload on a CPU, power managers will increase the clock frequency as long as the workload remains high, and since applications execute as long as work is available for execution, the workload will remain high for the whole execution. This leads to Race-to-Idle [11] conditions, in which the CPU is executing the work as fast as possible in order to reach the idle state. Consequently, by using high clock frequencies, it leads to unnecessary execution on the big cores after which the system idles until more work is available.

In this work we investigate whether a power manager driven by performance monitoring in the applications is able to more efficiently manage big.LITTLE architectures. The CPU allocation is directly based on application performance monitored by a new kind of power manager. We use a big.LITTLE power model created from real-world experiments to obtain the most power efficient execution at run-time. The model is able to predict the optimal clock frequency to satisfy the performance requirements of the applications.

We evaluate the system with typical legacy applications, and up to 57% of energy savings have been obtained with executions on real hardware using an unmodified Linux OS.

2. RELATED WORK

Power optimization of DVFS in multi-core systems has been extensively studied in the past [5, 10, 15]. A critical difference between traditional multi-cores and big.LITTLE multi-cores is the significant power reduction potential of executing tasks on the LITTLE cores. A utilization-aware load balancer for big.LITTLE system was presented in [9]. The balancer implemented a processor utilization estimator for determining the most optimal clock frequency for a given set of tasks without loosing performance. We argue that a utilization-based metric alone is not sufficient to efficiently control big.LITTLE power management. Instead we focus on performance monitoring *in the applications* in order to allocate the resources directly based on software demands.

The work in [3] presents the partition of real-time tasks onto heterogeneous cores such that energy is minimized by an optimal load distribution. The scheduling decisions were based on an analytical power model and an energy model based on the load distribution of tasks. Minimum energy consumption was calculated by modeling tasks executing on cores with given clock frequencies. Our work is focused on non real-time or soft real-time tasks without a given deadlines but with performance requirements in the applications. The power model we rely on is, in contrast to [3], derived from real-world experiments and not from analytical bottom-up models.

C-3PO [13] is a power manager used to maximize performance under power constraints and minimize peak power to reduce energy consumption. Applications are given a power budget, which is used for resource allocation in form of clock frequency and the number of cores. Orthogonally, we aim to minimize power under performance constraints. This means that our notion of constraints relate to the execution of applications rather than the power dissipation of the hardware. We further aim to implement this practice on big.LITTLE CPUs on which power is significantly reduced as long as the execution can take place on the LITTLE cores.

3. EXECUTION MODEL

The consequence of using workload-based power management is in often an execution model called "Race-to-Idle" [11]. Its behavior is to execute a job as fast as possible in order for the CPU to minimize the execution time and to maximize the idle time. The popularity of this execution model relates to simple programming; the programmer specifies only the program functionality, and the OS scales the clock frequency indirectly according to the workload.

Ondemand power management. Clock frequency in Linux based systems is driven by a kernel module called *frequency governor*. A frequency governor is monitoring the workload of the system and adjusts the clock frequency according to the policy for the governor in question. A number of different governors can be installed on a system, but usually the default governor is called *Ondemand* [14]. The Ondemand governor monitors an *upthreshold* value after which the workload is considered "too high". As the threshold value is reached, the governor switches the clock frequency automatically to the highest value (as illustrated in Figure 1). After the maximum value is reached, the governor decreases the clock frequency step-wise in order to find the most suitable frequency.

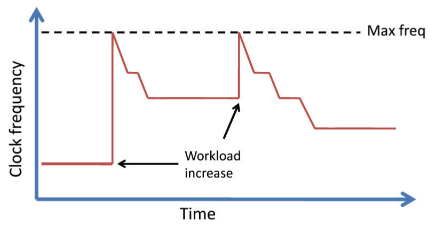


Figure 1: Illustration of the clock frequency scaling strategy of the Ondemand governor

The strategy of the governor was designed to rapidly respond to changes in workload without performance penalty, and to save power by step-wise scaling down. However, this strategy **a)** forces the CPU to always execute some part of the workload on the maximum clock frequency if the threshold is reached and **b)** for Race-to-Idle conditions, most of the workload will execute on the maximum (or a high) frequency since the workload will remain high as long as jobs are available for execution. For big.LITTLE systems, this strategy is contradictory to the intentions of the hardware since much time is spent on executing on high frequencies (with big cores) even if the system has significant idle time.

QoS driven power management. We argue that workload alone is not a sufficient metric to efficiently control big.LITTLE systems, instead the system should measure application performance for driving the power management.

As example illustrated in Figure 2, a video decoder decodes a number of frames and puts them in a display buffer. When the buffer is full, the decoder waits until the buffer is emptied. Since the output is usually determined by a fixed framerate, e.g. 25 frames per seconds (fps), the decoder is only required to decode frames at the same rate as the output display is using. Part (A) illustrates the Race-to-Idle strategy in which the CPU executes on maximum clock frequency for half a time window, after which it idles on the lowest clock frequency. The decoding process is hence producing 50 fps while the required rate would be 25 fps. Even though the power dissipation of the CPU is low on the idle part, the decoding part uses only the big cores even if the LITTLE cores would be sufficient when stretching the execution.

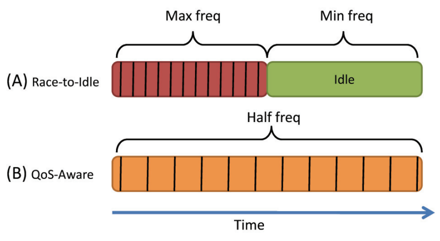


Figure 2: Illustration of (A) Race-to-Idle strategy and (B) QoS-Aware strategy

To create a system controlled by software requirements, we implemented a framework [6] to inject application specific performance directly into a new type of power manager (further explained in [7]). The power manager monitors the *performance* of the applications to determine the magnitude of the CPU resource allocation.

The power manager supports an execution strategy called *QoS-Aware*. The strategy is illustrated in Figure 2 (B), in which the execution time is stretched out over the whole time window. By executing only at the *required* clock frequency, the LITTLE cores are utilized as long as the performance is sufficient. The power manager is re-evaluating the performance measurements periodically, and the effort of the programmer is to suitably assist the power manager with the *performance* parameter. Practically, one line of c-code must be added to the applications:
`fmonitor(<performance>);`. This function calls the power management library and provides the run-time information, for example the current decoding framerate (fps).

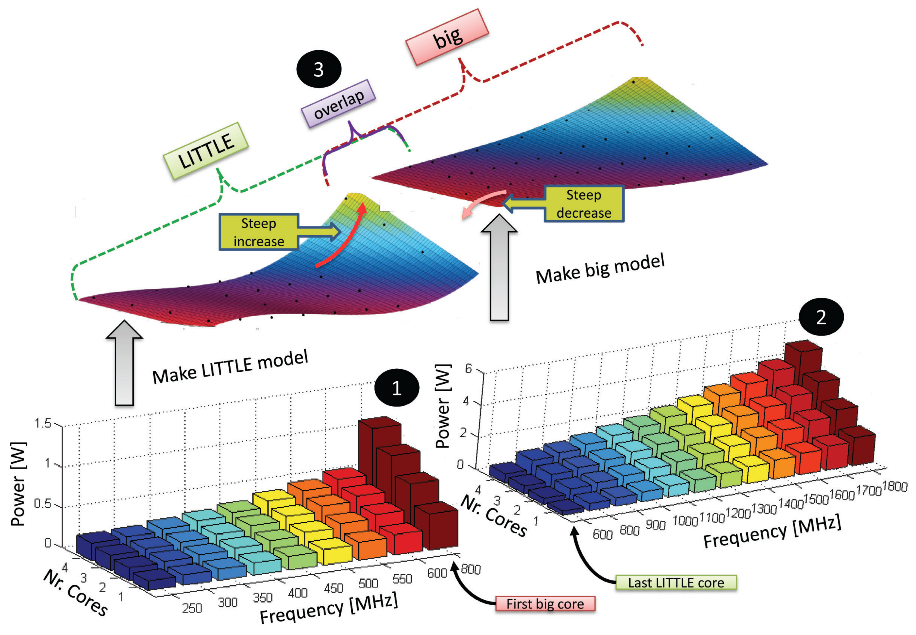


Figure 3: Creation of big.LITTLE power model. Separate reference measurements on the LITTLE and the big cores are used to generate a mathematical model which overlaps in the [600 800] MHz range.

4. BIG.LITTLE POWER MODEL

The power manager uses a power model to determine the increase in power by increasing/decreasing the clock frequency one step. The performance values given by the `fmonitor` library call are compared against a power model in order for the power manager to determine the power output caused by the CPU allocation.

As an application demands more resources, the aim is to chose a frequency which results in minimum power increase and sufficient performance increase. In contrast to our previous work on homogeneous systems [7], we require a dynamic model for describing the big.LITTLE architecture in which two types of cores can be used. As the model is constructed by mathematical expressions including architecture based parameters, the power manager must be able to adjust the dynamic parameters based on the core type currently in use. Since we use a big.LITTLE system with cluster switching [12], we consider only one type of core active at one time.

Similarly to [7], we stressed the physical system to maximum CPU load with the *stress* benchmark under Linux. Under full load we increased the number of cores and the clock frequency step-wise until all configurations were obtained. The power was physically measured after each step by reading internal power measurement registers in the chip.

By using the real-world measurements, we transformed the results into two mathematical functions using plane fitting methods [8] into a third degree polynomial[1]: $P(q, c) = p_{00} + p_{10}q + p_{01}c + p_{20}q^2 + p_{11}qc + p_{30}q^3 + p_{21}q^2c$ where P is the power, q is the clock frequency and c is the number of cores. With traditional non-linear optimization methods [4], we can minimize the cost (power) by selecting the optimal clock frequency for a given application based on performance requirements and the number of cores in use.

The studied architecture is a big.LITTLE configuration with two different types of cores, and the types are selected based on the clock frequency transition between 600 MHz and 800 MHz. We therefore created two separate power models for each core type based on the *stress* measurements. Figure 3 **(1)** shows the LITTLE measurements from 250 MHz to 600 MHz and **(2)** the big measurements from 800 MHz to 1800 MHz.

Because the aim is to keep the system executing on the LITTLE cores as much as possible, we overlapped the LITTLE and the big power models by including the lowest frequency of big cores in the LITTLE measurements (seen in Figure 3 **(1)**). This generates a steep cost increase when transitioning from the LITTLE to the big model (Figure 3 **(3)**), and pushes the optimizer to avoid the big cores if possible. Similarly, the highest clock frequency setting (600 MHz) of the LITTLE cores was included in the big-core measurement profile (seen in Figure 3 **(2)**), which drives the optimizer to descend to this setting if performance is sufficient. The result is a surface defined by the previously described third degree polynomial with one step overlapping (seen in Figure 3 **(3)**). The selected model (and p_{xy} parameters defined in the polynomial) is chosen based on the current core type in use, which can be monitored with Linux `sysfs`.

5. PRIORITY WEIGHT INTERFACE

As long as only one application has exclusive control over the power manager, no control conflicts can occur. However, as soon as several applications compete over the same resources, two applications could output conflicting execution conditions to the power manager. Conflicting information can result in wrong control settings for both appli-

[1]Further details in [7]

cations, instability in the resource allocation or diverging control output favoring one of the applications.

In order to increase the predictability of the control output which allocates CPU resources to the applications, the notion of priority weights in the applications was included in case several applications input conflicting information. The basic notation behind CPU allocation is the measured performance P_n of application n. P_n is compared to a user defined *setpoint* S_n, which marks the *desired* performance of application n. In case $P_n < S_n$, the application is given a positive *error value* E_n by the power manager, which signals for increased resource allocation. Similarly, in case $P_n > S_n$ the application is given a negative error value, which corresponds to resource waste and resource deallocation.

The magnitude of the error values determines the amount of resources to allocate/deallocate. With no notion of priority, the difference between setpoint and measured performance alone determines the error. By manipulating the magnitude of the error values, it is hence possible to alter the priority weight of an application error E_n, and increase the influence of important applications.

Application priorities in the Linux kernel are set by manipulating the run-time information of the tasks. The execution time of a task is simply replaced by a virtual time, which is manipulated according to priority weights. In other words, a high priority task will receive a slowly incrementing virtual time, which means that the scheduler will keep the task under execution for a longer "real" time.

We applied the same concept by replacing the error values with virtual errors vE_n to increase the influence of important tasks. The virtual errors of the applications were determined by sending all errors E_n and their respective priorities R_n to an error transformation function. Listing 1 outlines this procedure: **(2)** The system is monitoring all applications and calculate their respective error values based on the performance, **(3)** error values are replaced with virtual errors based on priorities, **(4)** the virtual errors are sent to the power manager which allocates the resources. Listing 2 shows the algorithm: **(1–4)** All applications are iterated over and a sum of all weights (priorities) for the current applications is calculated, **(5–6)** for each application, the virtual error is determined as the error multiplied with a weight determined by the priority in relation to all other applications (`weightsum`).

```
1   LoopForever{
2     <Apps><Errors><Priorities> = getMeasurements()
3     <vErrors> = veTrans(<Apps><Errors><Priorities>)
4     PowerManagement(<vErrors>)
5   }
```

Listing 1: Pseudo code for measurement procedure

The weight values were extracted from the Linux kernel source and are shown in Table 1. There are currently forty different priority levels defined by the weights where a higher weight means higher priority.

```
1   for(j=0; j<num_apps; j++){
2     weightsum = 0.0;
3     for(i=0; i<num_apps; i++){
4       weightsum += weights[priorities[i]];
5     }
6     verrors[j] = 2*errors[j]*weights[priorities[j]]/
          weightsum; }
```

Listing 2: c-code for generating virtual errors

Table 1: Weight values

15	18	23	29	36	45	56
70	87	110	137	172	215	272
335	423	526	655	820	1024	1277
1586	1991	2501	3121	3906	4904	6100
7620	9548	11916	14949	18705	23254	29154
36291	46273	56483	71755	88761		

6. EXPERIMENTAL RESULTS

For evaluation we required a benchmark with variable load, yet repeatable and multi threaded.

We chose video decoding using Mplayer[2] as basis for the evaluation. Further, we added a Facedetection application sharing the resources with Mplayer to create a mixed-priority scenario. Both applications were run with the Ondemand governor and with our optimized power manager under Linux 3.7.0. Our test platform was the 28 nm octa-core Exynos 5410 SoC based on the big.LITTLE configuration with four ARM Cortex-A15 cores and four ARM Cortex-A7 cores.

Mplayer. The first experiment was set up to use only the Mplayer application. Mplayer was set to decode and play a 720p video for 5 minutes using the h.264 video codec. Since the playback is executed with a steady framerate of 25 fps, we added a QoS requirement of 30 fps on the decoder by using our power management library. This means that the decoding process is slightly faster than the playback in order to keep up with occasional buffer underruns.

Figure 4 (A) shows the power dissipation for using Ondemand with a power sample rate of 250 ms. The dark gray curve is the A15 power, the black curve is the A7 power and the light gray curve is the memory power. With the resource requirement for decoding the 720p video, the workload exceeds the threshold used by the governor. Because of the Race-to-Idle strategy, the system is forced to stress the CPU to decode the frames as fast as possible and the core type in use is mostly the big A15 even though the performance of using a lower clock frequency would be sufficient.

By regulating the system according to the application specific performance (fps) instead of the workload, the CPU is allowed to stretch the decoding window while the output framerate is still met. Instead of racing to idle, a clock frequency below the core transition limit (800 MHz) is used which allows the system to execute on the LITTLE A7 cores. With this strategy there is almost no idle time in the system, but the execution is performed more energy efficiently and the performance requirements are met. Figure 4 (B)

[2]http://www.mplayerhq.hu/

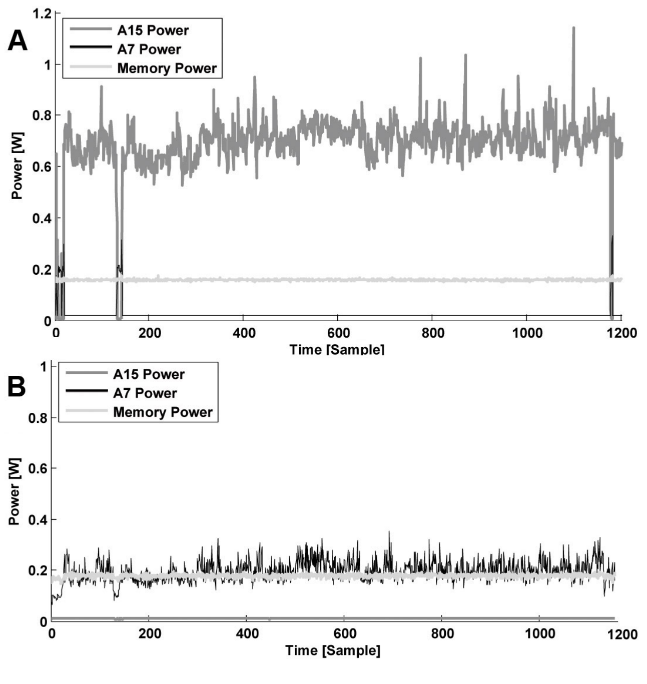


Figure 4: Power dissipation for Mplayer using (A) Ondemand (B) Optimizer

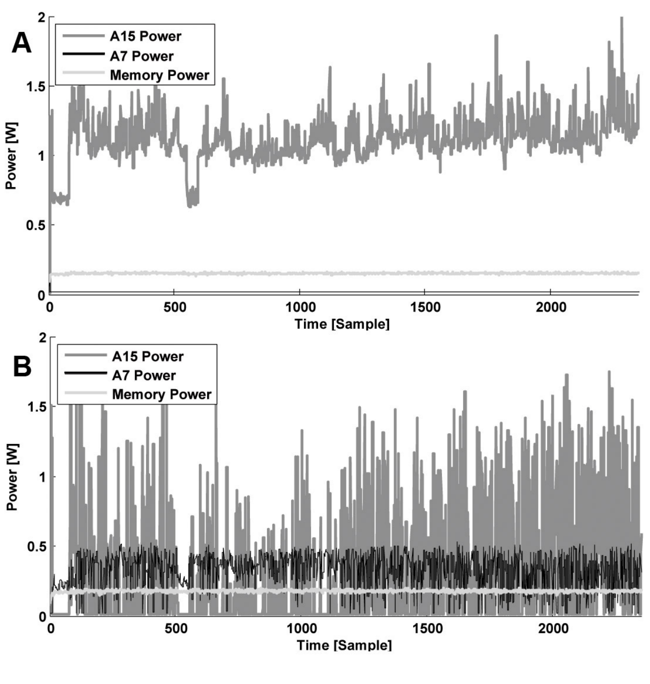


Figure 5: Power dissipation for Mplayer and Facedetection using (A) Ondemand (B) Optimized

shows the optimized execution in which the A7 cores are mostly used for processing the same video as in Figure 4 (A). The Ondemand governor consumed in total 103.96 Joules of energy while the optimized power manager consumed only 43.88 Joules, which is a reduction of 57% for executing the same amount of work with 0% performance degradation.

Mplayer + Facedetection. In the second evaluation we extended the use case to a mixed-priority scenario with several applications. Similarly to the previous evaluation we executed a 720p video with a required decoding rate of 30 fps. Furthermore, we added a Facedetection application used for video surveillance. The Facedetection application reads the input of a video stream, scans the current frame for the occurrence of one or more faces and draws a rectangle of the found face on the video stream. The QoS requirements added to this application was to scan 10 video frames per second for faces i.e. "10 Scans per Second (SPS)".

Since this application was used for surveillance, its performance was more critical than the video player. The priority for Facedetection was therefore set to 30 while Mplayer used a priority of 9. With a higher priority on Facedetection, it was expected for framedrops to occasionally occur in the video playback. We therefore executed Mplayer with parameters `-framedrop` and `-benchmark` in order to measure the number of dropped frames as well as the power.

Figure 5 (A) shows the power dissipation for using Ondemand and Figure 5 (B) for using the optimizer. Similarly to the Mplayer-only use case, the Race-to-Idle conditions of Ondemand forces a mostly high clock frequency and the workload is executed exclusively on the big A15 cores. The optimizer (in part (B) of Figure 5) shows a rather spiky output since the added Facedetection application occasionally requires more resources than what can be achieved on the LITTLE A7 cores. The system rushes to meet the performance requirements by temporarily using the A15 cores after which is it able to scale down to the A7 cores.

The mixed-application scenario occasionally imposes conflicting control signals based on the performance requirements. For example, while Mplayer is decoding very light frames and measures a "too high" framerate, Facedetection is under utilized and requires more resources. In order to verify the priority interface, we also plotted the scanrate for Facedetection during the whole experiment. Figure 6 (A) shows the scanrate for using Ondemand and (B) for using the optimizer. With a setpoint of 10 SPS we marked our acceptable lower and upper QoS limits for the application at 9 SPS and 11 SPS respectively. Since Ondemand is able to use the full power of the CPU all the time, it is expected to reach a more stable scanrate than the optimizer which can be seen in the figure. In case a better QoS is required using the optimizer, the user can either increase the performance setpoint to e.g. 12 SPS or increase the application priority with the cost of increased power dissipation.

Table 2 finally summarizes the mixed-scenario experiments. The optimized power manager was able to save roughly 40% of energy while imposing only a 1% QoS degradation on Mplayer and 6% QoS degradation on Facedetection compared to Ondemand.

Table 2: Energy (in Joules) and QoS (in %)

	Energy	QoS Mplayer	QoS Facedetection
Ondemand	334.3	100 (1 drop)	92 (52 late frames)
Optimized	201.5	99 (97 drops)	86 (108 late frames)

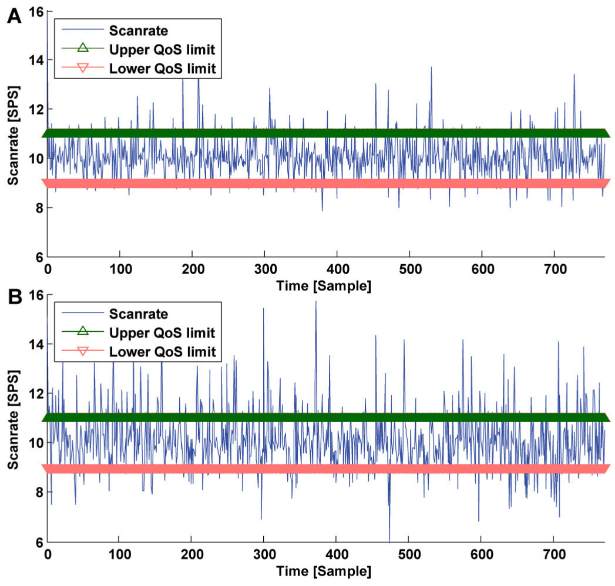


Figure 6: Scanrate and QoS for Facedetection using (A) Ondemand (B) Optimized

7. CONCLUSION

Workload alone is not a sufficient metric for driving power management in modern big.LITTLE systems. Since workload only expresses CPU utilization and not application performance, the execution is forced to Race-to-Idle as long as the workload remains high. By measuring the application performance and regulating the CPU allocation based on application requirements, the system is able to keep the execution of jobs on the energy efficient LITTLE cores for a longer time. We have presented a power manager utilizing a dynamic big.LITTLE power model for maximizing the LITTLE core usage. The usage is maximized by minimizing the idle time; allowing the system to execute on the lowest possible clock frequency without performance penalties. With an implemented library, applications can set performance requirements and input run-time information to influence the control decisions. Applications are further able to express their importance and the relation to CPU allocation in resource sharing scenarios involving several applications.

With real-world measurements using Linux running on big. LITTLE hardware we have obtained up to 57% of energy reduction for decoding typical HD videos with no performance degradation. Further on a mixed-priority scenario using one critical and one best effort application, we obtain energy savings up to 40% with minor QoS degradation compared to the default power management system. We plan to integrate the system into embedded devices such as mobile phones to increase the battery time when using typical every-day applications. We are also targeting *global task scheduling* systems in which both the big and the LITTLE cores are available at the same time.

8. REFERENCES

[1] ARM Corp. big.little processing witharm cortex-a15 & cortex-a7. `http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf`, 2011.

[2] D. Brodowski. Cpu frequency and voltage scaling code in the linux(tm) kernel. `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`, 2013.

[3] A. Colin, A. Kandhalu, and R. Rajkumar. Energy-efficient allocation of real-time applications onto heterogeneous processors. In *RTCSA, 2014 IEEE 20th International Conference on*, pages 1–10, Aug 2014.

[4] P. E. Gill, W. Murray, Michael, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 1997.

[5] M. Haque, H. Aydin, and D. Zhu. Energy-aware task replication to manage reliability for periodic real-time applications on multicore platforms. In *Green Computing Conference (IGCC), 2013 International*, pages 1–11, 2013.

[6] S. Holmbacka, D. Ågren, S. Lafond, and J. Lilius. Qos manager for energy efficient many-core operating systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 318–322, 2013.

[7] S. Holmbacka, E. Nogues, M. Pelcat, S. Lafond, and J. Lilius. Energy efficiency and performance management of parallel dataflow applications. In A. Pinzari and A. Morawiec, editors, *The 2014 Conference on Design & Architectures for Signal & Image Processing*, pages 1 – 8, 2014.

[8] K. Iondry. *Iterative Methods for Optimization.* Society for Industrial and Applied Mathematics, 1999.

[9] M. Kim, K. Kim, J. Geraci, and S. Hong. Utilization-aware load balancing for the energy efficient operation of the big.little processor. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4, March 2014.

[10] T. Rauber and G. Runger. Energy-aware execution of fork-join-based task parallelism. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 231–240, 2012.

[11] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. Adagio: Making dvs practical for complex hpc applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM.

[12] Samsung Corp. Heterogeneous multi-processing solution of exynos 5 octa with arm big.little technology. `https://events.linuxfoundation.org/images/stories/slides/elc2013_poirier.pdf`, 2013.

[13] H. Sasaki, S. Imamura, and K. Inoue. Coordinated power-performance optimization in manycores. In *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*, pages 51–61, 2013.

[14] V. P. A. Starikovskiy. The ondemand governor. In *Proceedings of theLinux Symposium*, 2006.

[15] I. Takouna, W. Dawoud, and C. Meinel. Accurate mutlicore processor power models for power-aware resource management. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth*

207. **Jongyun Moon**, Hydrogen Sensor Application of Anodic Titanium Oxide Nanostructures
208. **Simon Holmbacka**, Energy Aware Software for Many-Core Systems

ISBN 978-952-12-3305-0
ISSN 1239-1883

Simon Holmbacka

Energy Aware Software for Many-Core Systems

Simon Holmbacka

Energy Aware Software for Many-Core Systems