Energy Efficiency and Performance Management of Parallel Dataflow Applications

Simon Holmbacka, Erwan Nogues, Maxime Pelcat, Sébastien Lafond, Johan Lilius

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-to-idle strategy in which the operations are executed as fast as possible for a short time, after which it idles for the rest

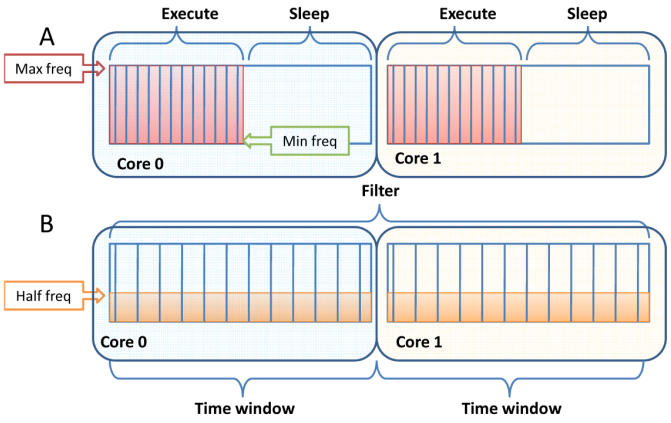

Fig. 1. Two execution strategies: A) Race-to-idle B) QP-Aware

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.

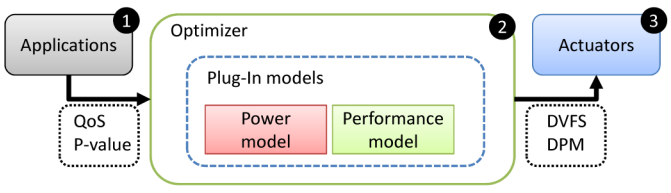

Fig. 2. Information flow from application to actuator

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

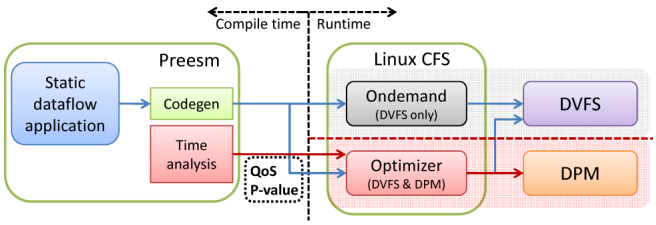

Fig. 3. Structure of the application ecosystem

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.

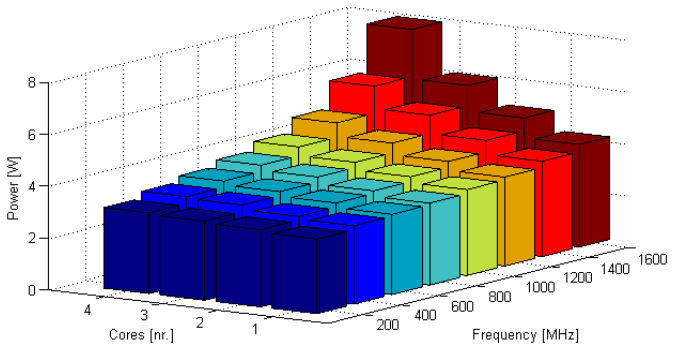

Fig. 4. Power as function of nr. of cores and clock frequency (fully loaded). Hot temperature on top and cold on bottom

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

$$P(q,c) = p_{00} + p_{10}q + p_{01}c + p_{20}q^2 + p_{11}qc + p_{30}q^3 + p_{21}q^2c \quad (1)$$

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

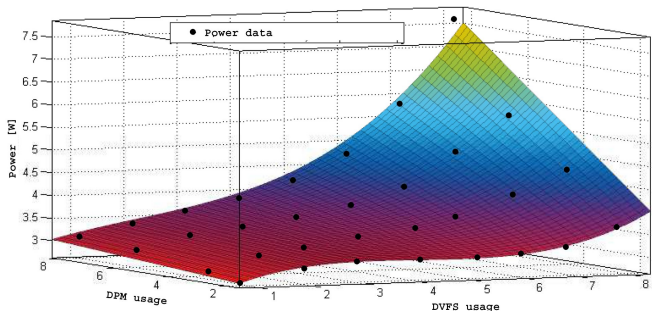

Fig. 5. Surface of the hot use case derived from Equation 1. Dots are real data measurements

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:

$$\text{Perf}_q(\text{App}_n, q) = K_q \cdot q \quad (2)$$

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:

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \quad (3)$$

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:

$$\text{Perf}_c(\text{App}_n, c) = K_c \cdot \frac{1}{(1-P) + \frac{P}{c}} \quad (4)$$

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.
- $d : E \to N \cup \{0\}$ is a function with $d(e)$ the number of initial tokens on an edge e
- $p : E \to N$ is a function with $p(e)$ the number of data tokens produced at e's source to be carried out by e
- $c : E \to N$ 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

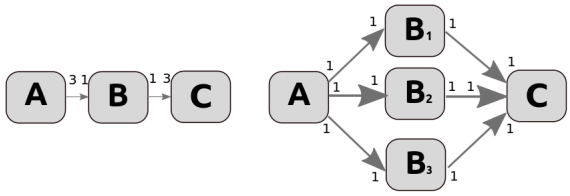

Fig. 6. A SDF graph and its srSDF transformation – multi-rate link to is transformed to several single-rate links to enable parallelism

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

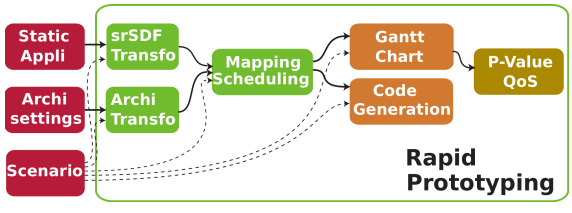

Fig. 7. Rapid prototyping description to extract QoS and P-value

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.

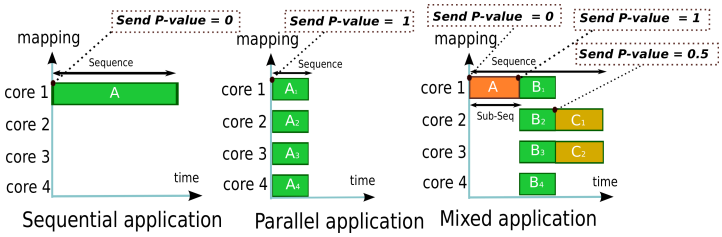

Fig. 8. Extracting P-value from the Gantt chart in PREESM

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.

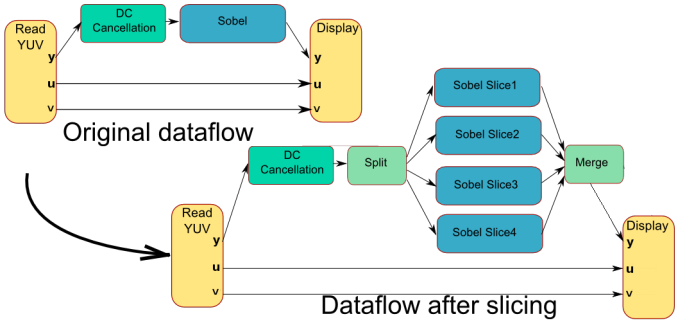

Fig. 9. Top level description - Original dataflow and after single rate transformation extracting data parallelism via slicing

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

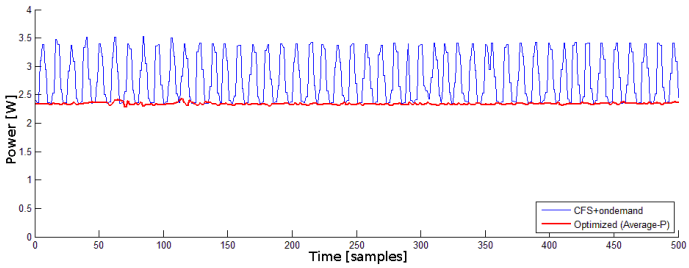

Fig. 10. Power trace from the sequential application using default CFS and with power optimizer

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

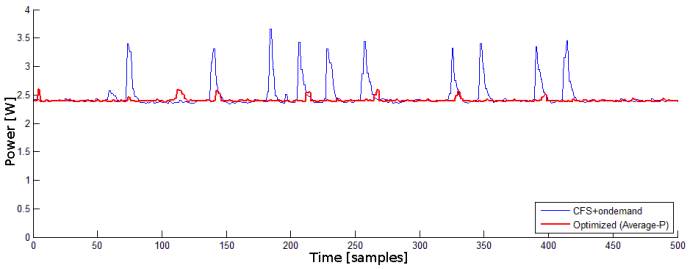

Fig. 11. Power trace from the parallel application using default CFS and with power optimizer

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

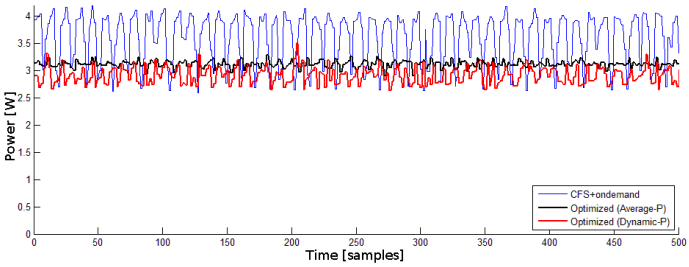

Fig. 12. Power trace from the mixed parallel application using default CFS and with power optimizer

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.

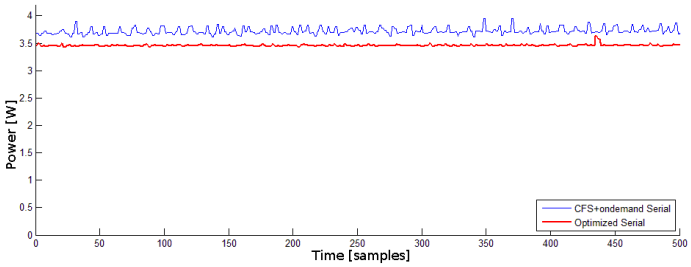

Fig. 13. Power trace from the mixed parallel application using default CFS and with power optimizer running on one thread

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.