Energy Efficient Scheduling

Lorenz Braun
lorenz.braun@stud.uni-heidelberg.de
Institute of Computer Engineering (ZITI),
Ruprecht-Karls-Universitaet Heidelberg

Abstract—As energy efficient hardware is developed and systems feature different kinds of cores, software needs to adapt to it. Especially schedulers need to acknowledge the heterogeneous computing platforms and make use of it. This paper aims to give a brief overview on different energy efficient scheduling algorithms for heterogeneous systems and help to find the right scheduler for applications running on such a platforms. The schedulers this paper covers are the ARM HMP Scheduler, a queue based scheduler, a portable c library called POET and a model based scheduler for real-time applications. The most important insight is that, independent of the scheduling strategy, the total load of the system must not be 100 % in order to optimize for energy efficiency.

Index Terms—Scheduling, Energy Efficiency, Heterogeneous Processors, ARM big.LITTLE, Embedded Computing

1 Introduction

Energy efficiency becomes more and more important these days, be it in server farms or in mobile devices. When performance is needed the output of each joule, which was sent into the CPUs, shall be maximized. But in times when the systems are idle, the power needed should be as low as possible. This is where heterogeneous computing platforms come in extremely handy. Powerful processors together with a smaller but very energy efficient CPU can fulfill the requirements mentioned before. But having the right hardware is not sufficient. The tasks running on such hardware have to be scheduled accordingly. Only good scheduling will enable the hardware to perform well and save energy in that process. This paper aims to give a brief overview on different energy efficient scheduling algorithms for heterogeneous systems.

2 Scheduling

If there are multiple processors and one or more tasks, a choice has to be made where the tasks are going to be executed. The unit that makes this choice is the scheduler. Its objective is to dispatch the tasks to the processors.

2.1 Common scheduling goals

In common scheduling there are two main goals:

- fairness
- load balancing

Depending on the environment where the scheduler is used there are additional goals. In a batch system, high throughput is demanded, whereas in interactive systems low latency is of importance. In real-time systems like self-driving cars there are deadlines, which have to be met. Therefore there is no scheduler which is suited best for all applications.

Common schedulers are not suited for heterogeneous systems, as they do not know that the cores are different, in terms of energy efficiency. Therefore the system cannot operate in a low energy mode in which only energy efficient cores are used. Also heavy tasks might not be executed on more powerful cores where they would profit of the high computational capacity. Another problem is that gang scheduling (used for collaborative threads) would waste performance because threads on the big core would always finish much earlier.

The current schedulers for homogeneous system save energy by balancing the load among the cores. By that the frequency can be lowered when the load is little. This is the only state-of-the-art technology for energy saving that homogeneous schedulers are capable of.

2.2 Energy efficient scheduling goals

In energy efficient scheduling it is desirable to reduce the total energy consumption. This focuses mostly at mobile devices. For embedded systems, the performance per joule shall be maximized. The scheduling here considers several factors. These are not only performance as before, but also energy consumption and thermal budget. Knowledge of the different power models has to be put into the scheduler, so that it can make decisions based on that data. This allows to dispatch lightweight tasks to the energy efficient cores and computationally demanding tasks to the powerful ones.

3 The ARM big.LITTLE Architecture

The ARM big.LITTLE architecture is a heterogeneous platform, which was developed in order to be energy efficient. It consists of two different clusters with four cores each. The little cluster has slow, but very energy efficient ARM Cortex-A7 cores, whereas the big cluster has faster ARM Cortex-A15 cores that have more computational power, but also need much more energy. Each cluster has access to the interconnect and a separate L2 cache, which lets them operate independently (Figure 1). Because the cores on each cluster are binary compatible, tasks can be migrated

from big to little and vice versa. Therefore this architecture can save power when only using the little cluster, without sacrificing computational power, which is provided when using the big cluster.

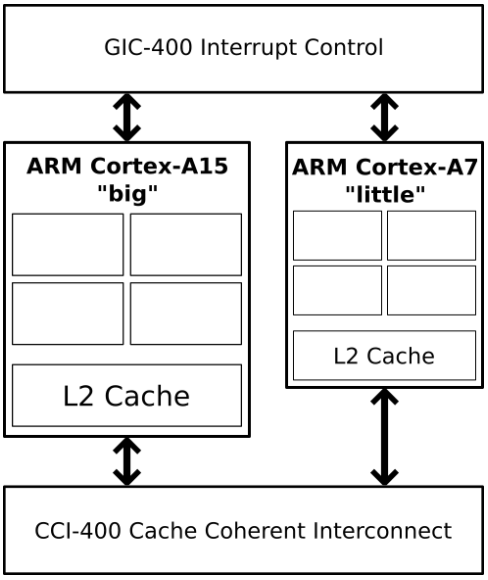

Fig. 1: Simplified figure of the big and little cores.

Both types of cores have performance ranges which overlap. In this range the little core consumes less power (Figure 2). Because the little cores are so efficient compared to the big cores, it is better to use only the little cluster up to a certain point. Only if the load requires more computational power the big cores are being used.

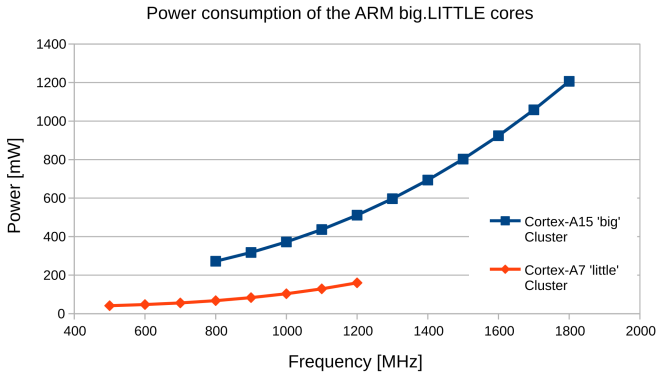

Fig. 2: Power consumption of big and little core compared to each other.

4 ARM HMP Scheduler

The ARM Heterogeneous Multi-Processing Scheduler was implemented for the Linux Kernel and is based on the Completely Fair Scheduler. It was developed to make use of heterogeneous hardware from mobile devices, which have a need to be responsive. Therefore applications need to have low latency when processing user input.

4.1 HMP Scheduling

The scheduler observes each task at runtime and tracks the load to make the scheduling decisions. The load is tracked for each time-slice and the scheduler will dispatch the tasks according to the following criteria (ARM, 2013):

- **Fork Migration**
  By default all tasks will be spawned on the big cluster. This ensures that compute intense tasks will get the power they need right from the beginning and that the latency is low.
- **Wake Migration**
  While a task is sleeping its load does not change. Therefore the tasks will be scheduled according to the last tracked load when it wakes up.
- **Forced Migration**
  Tasks which are running on the little cluster might never sleep and therefore have a high load. To boost those tasks they will be migrated to the big cluster when the load reaches the *up migration threshold*.
- **Offload Migration**
  When the load is under a certain *down migration threshold* for tasks on the big cluster, the task is migrated to the little cluster in order to save energy.

Because dynamic voltage and frequency scaling (DVFS) will lower the frequency of cores that are not fully utilized, the load can appear higher than it actually is. This could lead to unnecessary migrations. Therefore the scheduler can be improved by taking the current frequency of the core into account. If the little core is throttled down and runs a task which results in a load of 100%, it is more energy efficient to use the full performance of the little core first.

4.2 HMP Performance Evaluation

Yu et al. (2013) evaluated the scheduler with Bbench, an automatic web browsing benchmarking tool from the University of Michigan, and audio playback. The benchmark was to run both at the same time. The evaluation was executed on an Exynos5420 SMDK board, with an ARM big.LITTLE core. Figure 3 shows the improvement in energy efficiency for the ARM HMP Scheduler. Everything above the blue line is more efficient. The larger the distance the better the improvement. If the frequency is considered as well the scheduler will reduces the power consumption about 3.4 %.

5 Queue Based Scheduler

Systems like database servers which process independent and rather similar requests may make use of queue based scheduling. The idea here is to have one queue for all incoming requests. When processing them, the scheduler decides whether a big or the little core is used. Because the little cores are more energy efficient the scheduler makes use of them as much as possible.

Jain et al. (2015) developed a queue based scheduler for an embedded microserver. Besides the goal to be energy efficient, their application has soft real-time constraints. For this reason they considered throttle caused by thermal violations too, which affects the timing behavior a lot.

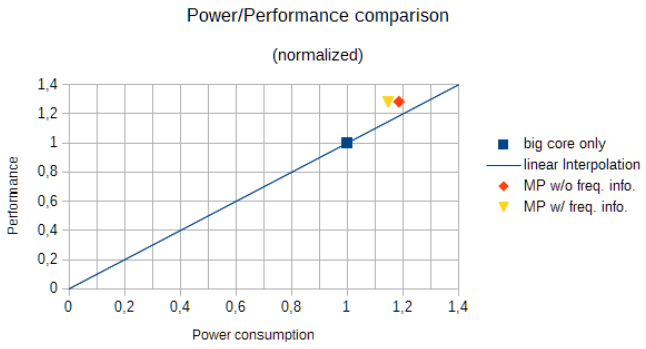

Fig. 3: Normalized power/performance evaluation of the ARM HMP Scheduler.

5.1 Queue Based Scheduling

Figure 4 shows the pseudo-code of the implementation. The preferred server are the little cores. Note that the big cores are only used when there is no little core available, no overheating (thermal violation) and the queue size has reached a certain threshold. The least energy would be consumed when only the little cores do the work. But to meet the timing constraints, the big cores are used to speed up the processing of the outstanding requests.

```
procedure SCHEDULETASK Input: PreferredServer
PreferredServer <- idle
NonPreferredServer <- idle
while TaskQueue is not empty do
    if PreferredServer is idle then
        Schedule the next job to the PreferredServer
    if (TaskQueue is not empty) AND
    (NonPreferredServer is idle) AND
    (There is no thermal violation) AND
    (TaskQueueSize >= Threshold) then
        Schedule the next job to the
            NonPreferredServer
```

Fig. 4: Pseudo-code for the scheduler (Jain et al., 2015).

The scheduler can be improved by making the threshold dynamic. If currently a lot of requests are incoming, the threshold is set down to process the requests earlier on the big cluster. This prevents long queues which can cause the system to overheat, because the big core is then used for a long period of time. Also, *Execution time prediction* and *Out-of-order execution* should lead to better energy efficiency. That is because knowing the total execution time of the queue is a better indicator for how full the queue is than just the the number of elements in it. *Out-of-order execution* makes it possible to process time critic requests earlier. In addition, pairing this with execution time prediction allows processing for time-critic requests to be accelerated.

5.2 Queue Performance Evaluation

Energy efficiency depends on the implementation used (Figure 5). Static means that the threshold was fixed, where as dynamic implies that that threshold changed with the load on the system. The evaluation was executed on an ODROID - XU3 with Samsung Exynos 5422 MpSoC. The proposed scheduler performs better than the opportunistic scheduler, which uses all the big cores before the little cores. The optimizations indeed increase the energy efficiency. Using *Out-of-order execution* and *Execution time prediction* is more than five times as efficient as the opportunistic scheduler.

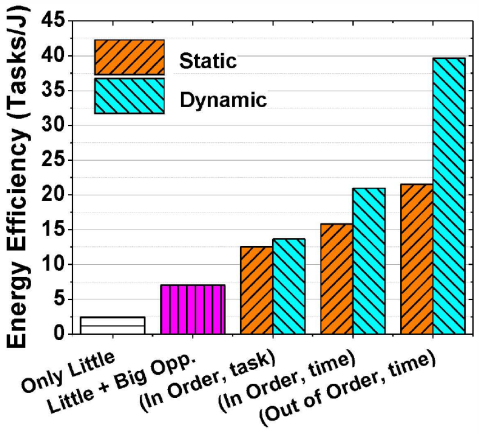

Fig. 5: Comparison of energy efficiency with the queue based scheduler (Jain et al., 2015)

6 POET: A Portable Approach to Minimizing Energy

POET (Performance with Optimal Energy Toolkit) is a portable C library to minimize energy consumption under soft real-time constraints developed by Imes et al. (2015). Its goal is to provide a portable solution which is not hardware dependent and easy to use.

6.1 POET Scheduling

The scheduling is handled by a digital control system, where the application latency is measured in order to meet the deadline. Until now, only one application can be scheduled because the dispatching of the tasks would get too complicated. Note that there can be several tasks, but all have to belong to the same application.

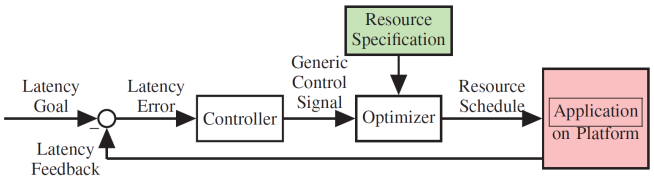

Fig. 6: The controller of the POET-library (Imes et al., 2015).

Figure 6 shows the model of the digital control system. The resource specification has to be provided by the user. It describes different core configurations with speed-ups, power usage and operating frequency. The controller processes the latency error and determines a speed-up for the application, which is given to the optimizer. This part of the system schedules the tasks to the execution units.

The optimizer has the task to find a schedule which will reach the requested speed-up of the controller with

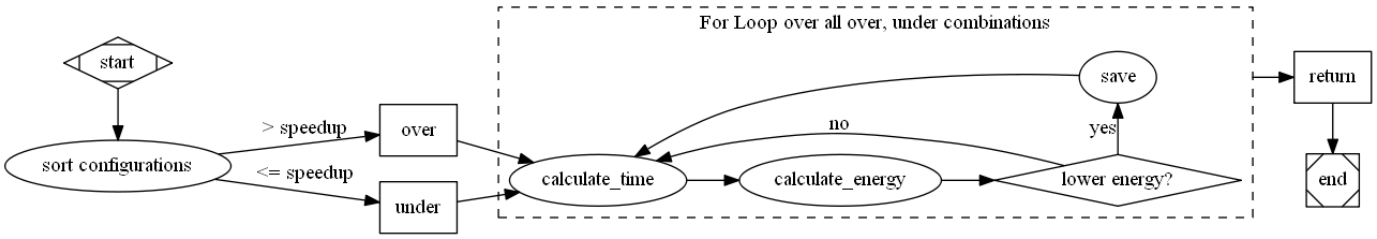

Fig. 7: Flowchart of the optimizer algorithm.

the minimal amount of energy. The flowchart in figure 7 shows how this is accomplished. First, all configurations are sorted into two sets. The *under-set* contains all configurations which are smaller or equal to the speed-up and the *over-set* contains all larger ones. For each combination of those sets, the execution time in each set and the total energy consumed is calculated. The combination with the least energy will be saved and the tasks will be executed on those two configurations. First on the under and then, if necessary, on the over configuration. Time and energy for the context switch is not considered. This overhead is modeled as inaccuracy in the specified speed-up.

6.2 POET Performance Evaluation

The library was compared to dynamic voltage and frequency scaling (DVFS) as single energy saving technique and the energy consumption was measured. Figure 8 shows the energy which is consumed as a function of the utilization intensity. 50 % means that half of the processing power of the system is used, 25 % uses only a quarter and so on. The energy scale is normalized to the optimal energy. The data was gathered from eight benchmarks: blackscholes, bodytrack, facesim, ferret, x264, dijkstra, sha and data streaming. All the benchmarks where executed on an ODROID - XU+E with Samsung Exynos 5410.

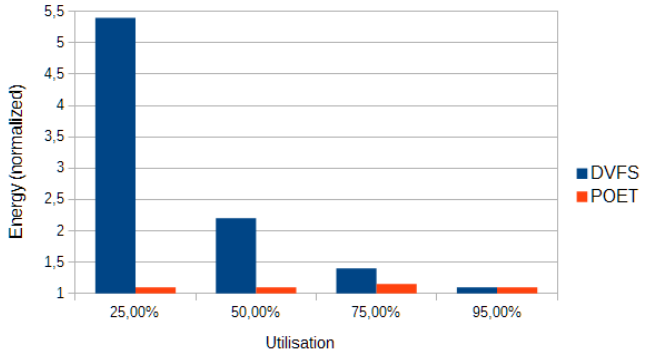

Fig. 8: Normalized Energy consumption of POET compared to DVFS

When most of the processing power is used, POET does not perform much better than DVFS. This is hardly surprising, as this case leaves little to no room for decisions for the scheduler. When turning to low utilization, POET performs very well and is almost equal to the optimal energy. Unfortunately the authors do not explain how the optimal energy is calculated. Compared to DVFS, POET needs about five times less energy which is a huge improvement, although it has to be admitted that this is a rather unfair comparison, because DVFS is not even a scheduler, it just throttles the cores when they are not under full load.

7 Model Based Scheduling

Colin et al. (2014) took a different approach to energy efficient scheduling with a model. Their focus is on real-time applications with sets of periodic tasks. The available processors and computational load (tasks) with deadlines can be modeled and used to find the optimal schedule with the lowest energy consumption.

The processors are modeled in regard to their power consumption and the according performance which is achieved with that power budget (Figure 2). It is a simplification, but if the operating frequency is assumed to be equal to the actual computational power, then it is always better to use the little core. Only in terms of energy and when more performance is needed the big core has to be used.

7.1 Workload Modeling

Benchmarks were executed to evaluate how the actual performance on both cores differs. They were used also to measure how the frequency affects required cycles on the CPU. In total 24 benchmarks were used. The results show, that in average the big core has a speed-up of 1.6 compared to the little core. The required cycles increase in average with higher frequencies. This behavior correlates for all benchmarks, with only a few exceptions. The reasons for this behavior were not investigated. A possible explanation might be that with higher frequencies memory latency is not hidden as well as with lower frequencies.

The tasks are modeled by the amount of cycles needed and a deadline. For the amount of cycles the maximum of all benchmarks is assumed, so that the inaccuracy does not affect the deadline constraints.

With all this information, the scheduling is just a matter of distributing the load to the cores in a manner that minimizes energy costs and fulfills the deadline constraints. This leads to the question, how the optimal load distribution for heterogeneous systems has to be. To answer this question, a formula that describes the power consumption of a core as a function of the frequency is needed:

$$P(f) = \kappa f^{\alpha} + \beta$$

κ, α and β were obtained by fitting a curve on real measurements on CPUs. Table 1 shows the model values

for the big and little cores. A slightly different model was taken to argue about the load distribution. Two cores were considered with $\alpha = 3$, $\beta = 0$ and varying κ. If the κ of the cores are equal (homogeneous system) the lowest power is achieved when the load is equally distributed. But if the κ of cores are different (heterogeneous system) the lowest power is achieved by giving most of the load to the core with the smaller κ (little core) and only some load to the other core. The ratio of the load distribution depends on the ratio of the two κ.

TABLE 1: Model values for the A-7 and A-15 cores (Colin et al., 2014).

Core	κ	α	β
A-7	1.00E-8	3.28	34.24
A-15	2.91E-6	2.63	146.49

This modeled load distribution is counter-intuitive, as one would usually assign as much work as possible to the small core because it is the cheaper one. But with such a simple model it is hard to make a general conclusion. The cores need different amounts of cycles for the same tasks for example. But even with a model that is not 100 % accurate the optimization problem can be solved quite good.

7.2 Model Based Scheduling

The optimal solution of the model can be found but this problem is NP-hard. Solving this in real-time is not feasible. Therefore good heuristics are needed. Two heuristics are provided and compared with a naive approach. They work in the following ways:

- **Naive (Load Balancing):**
  All tasks are sorted descending by computational demand. Each task will be scheduled to the processing unit (PE) with the least load at that point.
- **Marginal Power (M-PWR):**
  All tasks are sorted descending by computational demand. For each task, calculate the energy needed on every available PE and assign it to the PE where is will have the least power consumption.
- **Desired Load (DL-CAP):**
  Calculate the optimal load for each PE. Artificially limit the load to the optimal load. Then assign the task in the manner of the naive or the marginal power approach. After that set the limits to core capacity and assign the remaining tasks with the marginal power approach for optimal results.

Unfortunately the authors do not explain how the deadlines are taken into account.

7.3 Model Based Performance Evaluation

For evaluation, task sets were generated with more or less random workloads. They were all executed on an ODROID - XU+E with Samsung Exynos 5410. A comparison of the scheduling heuristics is shown in Figure 9. The energy is normalized to the naive approach which is represented by the yellow line. Similar to the evaluation of the POET library, the x-axis shows the utilization of the total computational capacity. Marginal Power and Desired Load perform almost equally well. From utilization 1 to 0.4, the energy saved increases because there is more room for decisions for the scheduler. The lowest energy consumption is achieved at a utilization of 0.4 and is about 72.5 %. After that, the energy saving worsens which might have to do with the deadlines or inaccuracy of the model.

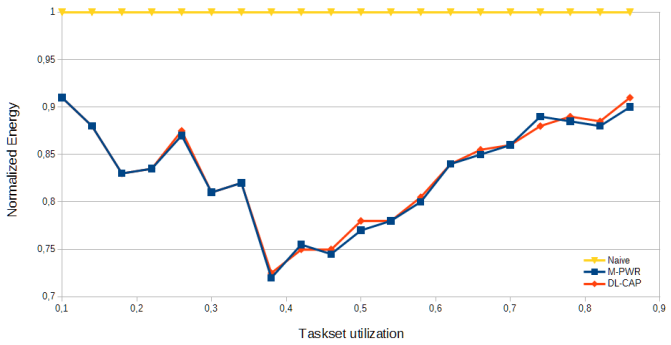

Fig. 9: Energy cost comparison of the scheduling heuristics.

8 Conclusion

The four presented types of schedulers all make use of heterogeneous processor platforms. Because they were developed for very different applications, they cannot be compared easily. Table 2 shows a comparison of the major features of the schedulers. The *best improvement* column seems to make the scheduler comparable, but they were compared to very different baselines. POET for example was compared to DVFS, which is not a fair match.

TABLE 2: Comparison of the schedulers.

Scheduler	ARM HMP Scheduler	Queue Based Scheduler	POET	Heterogeneous Load Distribution
Applications	any	applications with independent tasks of similar kind	any (only single applications)	any
Portable	yes	no	yes	no
Effort	least	moderate	little	high
Real-Time Support	no	soft real-time	soft real-time	hard real-time
Best Improvement (Energy Consumption)	$\sim$ 5 %	$\sim$ 15 %	$\sim$ 81 %	$\sim$ 38 %

The ARM HMP Scheduler is a scheduler for Linux, so it is probably a choice for heterogeneous systems with a full operating system. This scheduler has good potential for mobile devices hardware, which need only low performance

or are idle most of the time. In fact, the scheduler was developed for this use case.

When dealing with specialized embedded systems, the other three schedulers come in handy. The Model Based Scheduler is applicable for almost any use case. But when no hard real-time support is needed, the modeling is a rather high effort. When porting to other hardware this has to be done again.

POET and the Queue Based Scheduler are more specialized, but fit their niche very well. POET is suited for problem solutions which need only one application and no operating system. The amount of energy needed there can be reduced very well, especially when there are times where the load is low.

The Queue Based Scheduler is suited best for systems like micro-servers. The processed requests have to be independent to use this scheduler.

As already mentioned, there is no scheduler which is suited best for all applications, but the schedulers might be used for ranges of applications which overlap. All schedulers have in common that they can only optimize the energy efficiency when the total load of the system is a good step below 100 %. Evaluating the same or similar applications with different schedulers would be the next step in order to have an even better comparison of the schedulers and identify the most effective techniques to schedule on heterogeneous systems. This is where further investigation should be heading.

REFERENCES

ARM (2013). big.little technology: The future of mobile. https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf. [Online accessed 14.01.2016].

Colin, A., Kandhalu, A., and Rajkumar, R. (2014). Energy-efficient allocation of real-time applications onto heterogeneous processors. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–10. IEEE.

Imes, C., Kim, D. H., Maggio, M., and Hoffmann, H. (2015). Poet: a portable approach to minimizing energy under soft real-time constraints. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*, pages 75–86. IEEE.

Jain, S., Navale, H., Ogras, U., and Garg, S. (2015). Energy efficient scheduling for web search on heterogeneous microservers. In *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, pages 177–182. IEEE.

Yu, K., Han, D., Youn, C., Hwang, S., and Lee, J. (2013). Power-aware task scheduling for big. little mobile processor. In *SoC Design Conference (ISOCC), 2013 International*, pages 208–212. IEEE.