Energy-efficient Mapping of Task Collections onto Manycore Processors

Patrick Cichowski
Faculty of Mathematics and Computer Science
FernUniversität in Hagen
58084 Hagen, Germany
patrick.cichowski@fernuni-hagen.de

Jörg Keller
Faculty of Mathematics and Computer Science
FernUniversität in Hagen
58084 Hagen, Germany
joerg.keller@fernuni-hagen.de

Christoph Kessler
Dept. of Computer and Information Science (IDA)
Linköpings Universitet
58183 Linköping, Sweden
christoph.kessler@liu.se

ABSTRACT

Streaming applications consist of a number of tasks that all run concurrently, and that process data at certain rates. On manycore processors, the tasks of the streaming application must be mapped onto the cores. While load balancing of such applications has been considered, especially in the MPSoC community, we investigate energy-efficient mapping of such task collections onto manycore processors. We first derive rules that guide the mapping process and show that as long as dynamic power consumption dominates static power consumption, the latter can be ignored and the problem reduces to load balancing. When however, as expected in the coming years, static power consumption will be a notable fraction of total power consumption, then an energy-efficient mapping must take it into account, e.g. by temporary shutdown of cores or by restricting the number of cores. We validate our findings with synthetic and real-world applications on the Intel SCC manycore processor.

1. INTRODUCTION

Many applications in image processing can be described as so-called *streaming applications*. Such an application consists of a collection of tasks, each processing data at a certain (computational) rate. The tasks also forward their results to other tasks, or to outputs. A streaming application can be described as a directed graph similar to a static task graph, only that nodes and edges are anotated with rates instead of times, and that all tasks are active all the time. Surprisingly, also "classical" programs such as binary merge sort can be described as streaming applications. In the MPSoC community, efficient mapping of such applications has been the subject of intensive research, although the techniques seem often tailored to particular applications or are rather ad hoc. Our previous research [13] has investigated mapping of task collections onto manycore processors in such a way that the load is balanced between the cores.

In our current research we investigate energy-efficient mapping of task collections onto manycore processors where the different cores can be run at different frequencies, or even shut off for some time. We consider both dynamic and static power consumption, in contrast to related work that mostly focusses on dynamic power consumption alone. However, with shrinking feature sizes, the static power consumption will comprise a notable part of the total power consumption, so that it should be taken into account. We find that energy-efficient mapping basically is achieved through load balancing if dynamic power consumption alone is considered. If however static power consumption is taken into account as well, switching cores off from time to time gets more attractive, and using large numbers of cores at quite low frequencies gets less attractive.

We implement a synthetic benchmark application and a real-world application (computation of images from mandelbrot sets) on the Intel SCC manycore processor and measure power consumption in different scenarios. The measurements support our arguments qualitatively, in particular that a balanced load leads to smaller energy consumption, and that switching frequencies is not advantageous. However, as we can only measure power consumption for the complete chip, without the possibility to distinguish between cores, on-chip network and on-chip memory controllers, the measurements are not exact enough to exactly reflect the forecasts from the model.

The remainder of this paper is structured as follows. In Section 2, we briefly review related work. In Section 3, we derive requirements for an energy-efficient mapping of task collections. Section 4 presents our experimental setting and results, and Section 5 concludes.

2. RELATED WORK

Energy optimization for multiprocessors by speed scaling (frequency and/or voltage scaling) and multi-objective optimization for energy and time has been considered in the literature for various scenarios. A recent survey is given by Albers [1]. Approaches for task graphs with precedence constraints such as [14] are less related because we consider the steady state of a pipelined streaming computation where all tasks are active concurrently and the optimization goal for the time dimension is throughput rather than the makespan of a single task graph execution. Speed scaling for periodic real-time tasks with release times and deadlines comes closer to our streaming scenario and has been considered e.g. by Albers et al. [2] and Andrei [3].

Albers et al. [2] consider energy optimization for such realtime tasks scheduled preemptively using off-line and online heuristics such as round-robin and earliest-deadline first. The single-processor algorithm by Yao et al. [15] is used as a subroutine within each processor. Static power consump-

tion and shutdown / sleep mode of processors are not considered. Also, we consider a slightly different problem: in our case there are no specific deadlines, our streaming tasks are active continuously and triggered by data flow; we can thus only consider average loads for work and communication volume that are spread out equally over time. Hence, our theoretical results derived in the following section (no frequency changes within a processor for optimal energy usage) cannot be directly applied to the general scenario in [2]; in fact, [2] may perform dynamic frequency changes within processors. Albers et al. [2] give a NP-completeness proof that also includes the special case (arbirary task loads, all releases identical, all deadlines identical) that comes very close to our scenario considered in this paper.

Andrei [3] considers dynamic voltage scaling for cores and buses with communicating realtime tasks on embedded MPSoC. His energy optimization by voltage scaling considers both dynamic and static energy, and also shutdown (sleep mode) of cores. In contrast to Albers [2] he also considers the time and energy overheads of switching and shutdown. A NP-completeness proof is given, and both offline (using mixed integer linear programming) and online optimization algorithms are provided.

3. ANALYSIS

Starting from an abstract level, the power consumption of a semiconductor device (such as a processor core) can be described by a dynamic part that depends on frequency and voltage, and a static part that only depends on the supply voltage. In most devices, frequency and voltage are not set independently, but for a given frequency, the minimum possible voltage is used. As the static power consumption is linear in the supply voltage if the threshold voltage is not approached [4, Eq. 10] and assuming that the minimum possible supply voltage for a given frequency can be approximated by a linear relationship, a high-level power model of a device running at frequency f can be formulated as

$$p_{core}(f) = p_{dyn}(f) + p_{stat} = b \cdot f^a + s \cdot f \ , \qquad (1)$$

where a, b and s are device specific constants and a typically is assumed to lie between 2 and 3. For manycore processors, where multiple cores, an interconnection network and memory controllers are integrated on a chip, one has to extend this model for the other components. If we assume that the memory controllers and the network are running at a fixed frequency, their power consumption can be considered to be a constant S given that the application is fixed. Then, if the cores run at frequencies f_i

$$p_{many}(f_0, f_1, \ldots) = S + \sum_i p_{core}(f_i) \ .$$

To derive energy consumption over a time interval T, we multiply power consumption and time if the frequency is fixed. If the frequency varies, we sum up over the partial intervals where the frequency is fixed.

We begin our analysis by considering a single core. We will assume $b = 1$ for better readability. If we map a certain load f_0 to a core, then we assume that the core can carry this load if it continually runs at a frequency f_0. We restrict our analysis to a time interval T. We could vary the operating frequency of the core within this interval. If the core runs at frequency $f_1 \leq f_0$ for a fraction α of the time interval T, then for the remaining time $(1-\alpha)T$ it must run at frequency $\bar{f}_1 = (f_0 - \alpha \cdot f_1)/(1-\alpha) \geq f_0$ in order to perform the same amount of work as running with constant frequency f_0 during the whole interval T. Then the dynamic energy consumption in the case of using the constant frequency is $E_0 = f_0^a \cdot T$, while the dynamic energy consumption in the other case is

$$E_1(f_1) = \alpha \cdot T \cdot f_1^a + (1-\alpha) \cdot T \cdot \bar{f}_1{}^a \ .$$

For any fixed $\alpha \in [0;1]$, $E_1(f_1)$ is minimized for $f_1 = f_0$ as can be seen by computing the first derivation. Thus, if a core is running, it should always run at the same frequency.

If we ignore static energy consumption, the above also means that a processor should never be switched off ($f_1 = 0$). If we include static energy consumption, then a core running for a time interval T with frequency f_0 consumes energy $E_0 = (f_0^a + S) \cdot T$, while if the core is switched off ($f_1 = 0$) for a fraction α of that time interval, it must run at frequency $\bar{f}_1 = f_0/(1-\alpha)$ for the rest of the time and consumes energy

$$E_1(\beta) = \beta \cdot T \cdot ((f_0/\beta)^a + S) = T \cdot (f_0^a/\beta^{a-1} + S \cdot \beta)$$

for $\beta = 1 - \alpha$. This is minimized for

$$\beta_0 = f_0 \cdot \sqrt[a]{(a-1)/S}$$

Thus, as long as the static power consumption at frequency f_0 does not supersede the dynamic power consumption, $\beta_0 > 1$, and thus the minimum energy consumption is reached for $\beta = 1$ or $\alpha = 0$, as E_1' is negative in the interval $[0;\beta_0]$. This means, that in this case the core should not be switched off. However, if the static power consumption is more than half of the total power consumption at frequency f_0, then the core should be run at frequency f_0/β_0 for a time span $\beta_0 \cdot T$, and then be switched off.

If we consider two cores running at frequencies f_1 and f_2, where $f_1 + f_2 = c$ is a constant, then the energy consumption $f_1^a + (c - f_1)^a$ is minimized for $f_1 = f_2$, i.e. the load should be balanced between cores as far as possible. This can be seen by computing the first derivative. With respect to the number p of cores to be used in case a load can be equally partitioned among any number of cores, the energy consumption decreases with an increasing number of cores, if static power consumption is not taken into account, i.e. $E(p) = T \cdot p \cdot (f/p)^a$. However, if static power consumption is taken into account, the formula changes to

$$E(p) = T \cdot p \cdot ((f/p)^a + s) \ .$$

Now there exists a minimum at $p = \sqrt[a]{(a-1)f^a/s}$, so that there is a maximum number of cores to be used for minimum energy consumption.

So far, we have assumed that we can choose core frequencies freely, and that we can divide the load as required. If this is not the case, then the balancing problem is a variant of the variable-sized bin packing problem. The task loads correspond to the item volumes and the cores at different frequencies correspond to the bins of different sizes. The cores' power consumption corresponds to the bin costs. If there are p processors which can run at k different frequen-

cies, then there are $p \cdot k$ bins, of which only p are to be used. There are several efficient heuristics for this problem [6, 7].

4. EXPERIMENTAL RESULTS

The Intel Single Chip Cloud Computer (SCC) consists of 48 independent cores, which are organized in 24 tiles. The tiles are interconnected by a 6×4 mesh on-chip network, which also connects them to four on-chip memory controllers to access up to 64 GB of off-chip DDR3-RAM. For dynamic voltage and frequency scaling (DVFS), the cores of the SCC are organized in 24 frequency islands, one island for each tile, and 6 voltage islands, each for a group of 4 tiles. The network and memory controllers are frequency islands of their own [9].

In our previous work [5] we defined a parameterized model of power consumption by cores for the Intel SCC. In order to obtain the model's parameters, we devise microbenchmark programs and different machine settings, measure the power consumption of the SCC and subject the differences between the measurements and the power model to a least-squares error analysis. We devise, that for each core the parameter for the dynamic power consumption is around $b_c = 2.015 \cdot 10^{-9}$ Watt/MHz^3 and the parameter for the static power consumption is around $s_c = 10^{-6}$ Watt/MHz. The other components like the network and the memory controllers consume around $S = 23$ Watt. The static power consumption (leakage) of the Intel SCC comprises between 15% of the total power at high frequency (1 GHz, 125 W total) and voltage and 35% at low frequencies (125 MHz, 25 W total) and voltage [8].

In our current experiments, we employ the same microbenchmark programs and settings as in our previous work. Furthermore we implemented the parallel computation of an image representing a mandelbrot set as a realistic expensive application. To change the frequency and voltage of the cores during runtime we use the power management function RCCE_iset_power() from the RCCE library, which is provided with the SCC. With this function one is able to change the frequency of the cores and let the voltage automatically scale to the lowest stable state. In this case, there are only 6 power domains, which are equal to the voltage islands [11]. Thus we scale frequencies for groups of 8 cores. We first run all cores at the same frequency with an equal load, measure the power consumption of the entire SCC and compare it to the power consumption of the SCC with a mixed variant, where a fraction α of the cores run at frequency f_1 and $(1 - \alpha)$ cores at frequency f_2. We don't change the frequencies for the on-chip network and the memory controllers. Consequently we choose for α the values $0/6, 1/6, 2/6, ..., 6/6$, which are corresponding to the different number of power domains with the same frequency f_1. Then, the other power domains $(1 - \alpha)$ run at frequency f_2. As low frequencies (f_1) we use the values 100 and 200 MHz and as high frequencies (f_2) the values 400, 533 and 800 MHz. The cases $\alpha = 0/6$ and $\alpha = 6/6$ are special cases, in which all power domains run at the same low or high frequency, respectively.

In the microbenchmark program the two groups of high and low frequency domains are in a first step scaled to their corresponding (user-defined) frequencies. After that, we use a barrier to ensure that all voltage and frequency scalings have finished and all cores can begin with the second step simultaneously. In the second step, we perform an integer summation for 10 seconds within a loop and measure the power consumption every 10th ms with the FPGA on the Rocky Lake Board, which supports direct measurement of voltages and currents of the domains [10]. We repeat the experiment 5 times and average over all power measurements. The power consumption only varies within a 5% range around the average. We use four different settings to perform the integer summation, which differ in the use of caches, and intensity and regularity of memory access (see Tab. 1). However, our previous work shows that this has a negligible effect on the power consumption.

Table 1: Different benchmark settings

Benchmark	Description
0	Step 1: One variable, initially set to 0 Step 2: Variable is incremented by 1
1	Step 1: array[size 10^6], initially set to 0 Step 2: array elements added up successivly
2	Step 1: array[size 10^6], initially set to max_int Step 2: array elements added up successivly
3	Step 1: array[size 10^6], initially set to index Step 2: array elements added up in the following order: (7· index + rank) mod array_size

In the first setting an integer variable (initially set to 0) is incremented by 1 within the loop. This represents a microbenchmark with use of ALU and caches, and few memory accesses. In the other settings we use an array with one million elements, which are initally set to 0 for the second setting, to the maximum integer value for the third setting and to the index of the element in the fourth setting. The second and third settings add up the array elements successively. This represents microbenchmarks with cache and memory accesses, and different ALU use (adding up zeroes, adding up ones). The last setting uses a more unstructured access pattern ($7 \cdot$ index + rank mod array_size) to add up the array elements, which represents higher cache miss rate and thus higher memory traffic [5].

To measure the power consumption of the balanced frequencies f, which result from $\alpha \cdot f_1 + (1-\alpha) \cdot f_2 \sim f$, we can only use such values that are supported by the RCCE library. Thus, we use the values 200, 400 and 533 MHz and compare it with the power consumption of the measurements with mixed frequencies. As a consequence, there are only 5 pairs of f_1 and f_2, which are nearly equal to the corresponding balanced frequency f as depicted in Tab. 2.

In this table we can see that in the first two cases the power consumption of the balanced and mixed variants are nearly identical. In contrast, the power consumption of the mixed variants in the last three cases is much higher than for the balanced variants. This results from the different voltage level, which RCCE uses for the different frequencies. There are totally 6 possible voltage levels, but only 3 voltage levels (0.7V, 0.8V and 1.1V) are used for the frequency scaling with the power management function RCCE_iset_power(). 0.7V

Table 2: Comparison between the power consumption of the SCC for the microbenchmarks with a mixed and a balanced frequency for the cores

α	freq. low (f_1)	freq. high (f_2)	freq. balanced (f)	Watt balanced	Watt mixed
4/6	100	400	200	25.6324	25.5430
2/6	100	533	$388,67 \sim 400$	30.0648	29.8622
4/6	200	800	400	30.0250	42.0186
2/6	100	800	$566,67 \sim 533$	32.8098	57.0382
3/6	200	800	$500 \sim 533$	32.1484	50.4092

is used in a range from 100 to 400 MHz, 0.8V for 533 MHz and 1.1V for 800 MHz. As expected the influence of the voltage scaling on the power consumption is much higher than the influence of the frequency scaling on the power consumption. Another aspect comes from the network and memory controller, because with a higher core frequency more data can be sent over the network to the memory, such that the usage rate of both increases.

Our real-world application is structured like the microbenchmark. After the frequency scaling and the barrier, the mandelbrot set computation is done in the second step. An image of size 6000×8000 pixels is generated, for each pixel the maximum number of iterations is set to 2047. Each task, i.e. each core, computes a sequence of pixels determined statically. The sequences are geometrically interleaved in order to distribute pixels with high and low iteration counts over the cores as evenly as possible with a static distribution. During the first 10 seconds the computation, we measure the power consumption of the chip every 10 ms, i.e. 1000 times. We repeat each experiment 5 times. We compute the average power consumption for each experiment as the average of 5,000 measurements. Although the work in this application is fixed, we measure only the first part of the computation, so that we match the model of continuously running tasks with given (average) computational rates. Tab. 3 depicts the power consumption of the 5 pairs of f_1 and f_2 for the real-world application.

Table 3: Comparison between the power consumption of the SCC for the real-world application with a mixed and a balanced frequency for the cores

α	freq. low (f_1)	freq. high (f_2)	freq. balanced (f)	Watt balanced	Watt mixed
4/6	100	400	200	27.0316	26.1008
2/6	100	533	$388,67 \sim 400$	32.3016	31.6888
4/6	200	800	400	32.9238	45.4560
2/6	100	800	$566,67 \sim 533$	36.8874	64.8296
3/6	200	800	$500 \sim 533$	36.2624	55.8794

The results of the power consumption for the real-world application are very similar to the results for the microbenchmarks. They differ only in that the power consumption with a mixed and a balanced frequency for each pair is a little bit higher than for the microbenchmarks. One reason for this behavior is, that there is no time control during the calculation and thus the cores compute more instructions than in the microbenchmarks.

Another performance test is to let all cores first run with the low frequency f_1 and after that with the high frequency f_2. In this case, when we for example assume that the balanced variant runs for a time interval of one second, we can also devise the average power consumption (i.e. energy consumption divided by the timespan) for different time periods with the high or low frequency. Thus we can see, if a longer time periode with a high frequency has a positive effect on the power consumption, because the total time for a run decreases. As an example Fig. 1 depicts the energy consumption of the microbenchmark for a balanced frequency of 200 MHz, a low frequency of 100 MHz and a high frequency of 400 MHz.

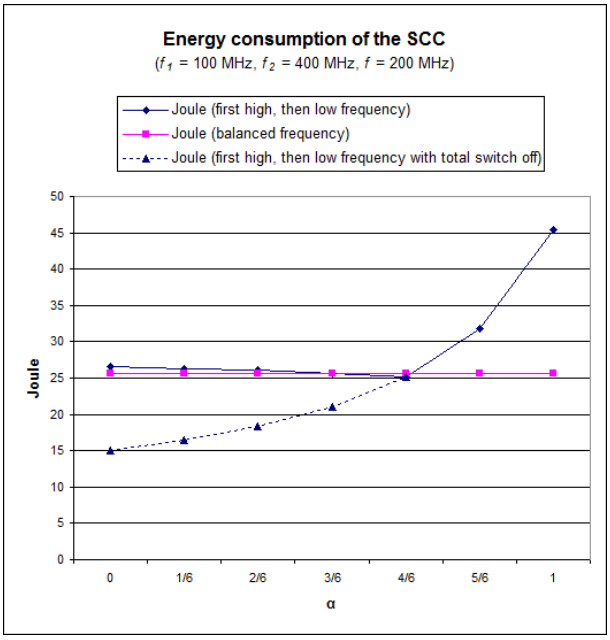

Figure 1: Energy consumption of the SCC for a microbenchmark run for one second

Seen in isolation, it is more efficient to run for some time with a high frequency and switch the cores off when the work is done. For $\alpha = 0$ (no low frequency time period) for example the energy consumption is around 15 Joule (dotted line in Fig. 1) compared to the 25 Joule of the balanced variant. The SCC cores can be turned off completely, but only under laboratory conditions and not with the RCCE library. Thus at least the energy consumption from static power must be added for the rest of the time. Even if we assume, that one could turn off the cores completely, the memory controllers can not be switched off, because in this case the memory contents would be lost and thus the cores can not be turned on again. Hence the power consumption for that time lies around 23 Watt for the network and memory controllers, contributing $\alpha \cdot 23$ Joule to the energy consumption. So in these cases, where the calculation finished earlier than with the balanced variant, the energy consumption of the mixed variant (dark solid line in Fig. 1) is slightly higher than the energy consumption of the balanced variant. Thus it is not advantageous to run a program with mixed frequencies.

5. CONCLUSIONS AND FUTURE WORK

We have investigated how to map task collections in streaming applications onto manycore processors such that energy consumption of the cores is minimized. We have treated the problem formally and experiments support our forecasts qualitatively.

So far, we have not yet taken into account the influence of the on-chip network. As a first measure, one could solve the core optimization problem, and then choose the lowest possible frequency for the on-chip network such that all tasks still perform as requested, i.e. are not penalized by waiting too long for outstanding memory requests or data communications. A more advanced measure could take the structure of the communication between the tasks into account during the mapping process, i.e. communicating tasks could be mapped onto the same core as long as the computational load remains balanced. This constitutes a multi-criterion optimization problem, which might be solved by integer linear programming similar to [12] as soon as the number of cores to be used are fixed. An even more advanced scheme would require to model the energy consumption of the on-chip network depending on load and frequency (if we assume that the operating frequency of the on-chip network can be chosen independently of the core frequencies) and solve the non-linear optimization problem of mapping the communicating tasks such that the total energy consumption by cores and network is minimized.

6. ACKNOWLEDGMENTS

The authors thank Intel for providing the opportunity to experiment with the "concept-vehicle" many-core processor "Single-Chip Cloud Computer". C. Kessler acknowledges partial funding by Vetenskapsrådet and SeRC.

7. REFERENCES

[1] S. Albers. Algorithms for dynamic speed scaling. In *Proc. 28th Symp. on Theoretical Aspects of Computer Science (STACS'11)*, pages 1–11, 2011.

[2] S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *Proc. 19th Annual Symp. on Parallelism in Algorithms and Architectures (SPAA'07)*, pages 289–298, June 2007.

[3] A. Andrei. *Energy Efficient and Predictable Design of Real-Time Embedded Systems*. PhD thesis, Dept. of Computer and Information Science, Linköping University, Oct. 2007.

[4] A. P. Chandrasakaran and R. W. Brodersen. Minimizing power consumption in digital CMOS circuits. *Proceedings of the IEEE*, 83(4):498–523, Apr. 1995.

[5] P. Cichowski, J. Keller, and C. Kessler. Modelling power consumption of the intel SCC. In *Proceedings of the 6th MARC Symposium*, pages 46–51. ONERA, July 2012.

[6] T. G. Crainic, G. Perboli, W. Rei, and R. Tadei. Efficient heuristics for the variable size bin packing problem with fixed costs. Technical Report 2010-18, CIRRELT, 2010.

[7] M. Haouari and M. Serairi. Heuristics for the variable sized bin-packing problem. *Computers & Operations Research*, 36:2877–2884, 2009.

[8] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. Van Der Wijngaart. A 48-Core IA-32 message-passing processor in 45nm CMOS using on-die message passing and DVFS for performance and power scaling. *IEEE J. of Solid-State Circuits*, 46(1):173–183, Jan. 2011.

[9] Intel Labs. SCC programmer's guide. Online documentation, November 2011.

[10] Intel Labs. The scckit 1.4.x user's guide (part 9). Online documentation, September 2011.

[11] Intel Labs. Using the RCCE power management calls. Online documentation, September 2011.

[12] J. Keller, C. Kessler, and R. Hulten. Optimized on-chip-pipelining for memory-intensive computations on multi-core processors with explicit memory hierarchy. *J. of Universal Computer Science*, 2012. to appear.

[13] C. W. Kessler and J. Keller. Optimized mapping of pipelined task graphs on the Cell BE. In *Proc. 14th Int. Workshop on Compilers for Parallel Computing (CPC-2009), Zürich, Switzerland*, Jan. 2009.

[14] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Theory Comput. Syst.*, 43:67–80, 2008.

[15] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU usage. In *Proc. 36th Ann. Symp. on Foundations of Computer Science (FOCS'95)*, pages 374–382, Oct. 1995.