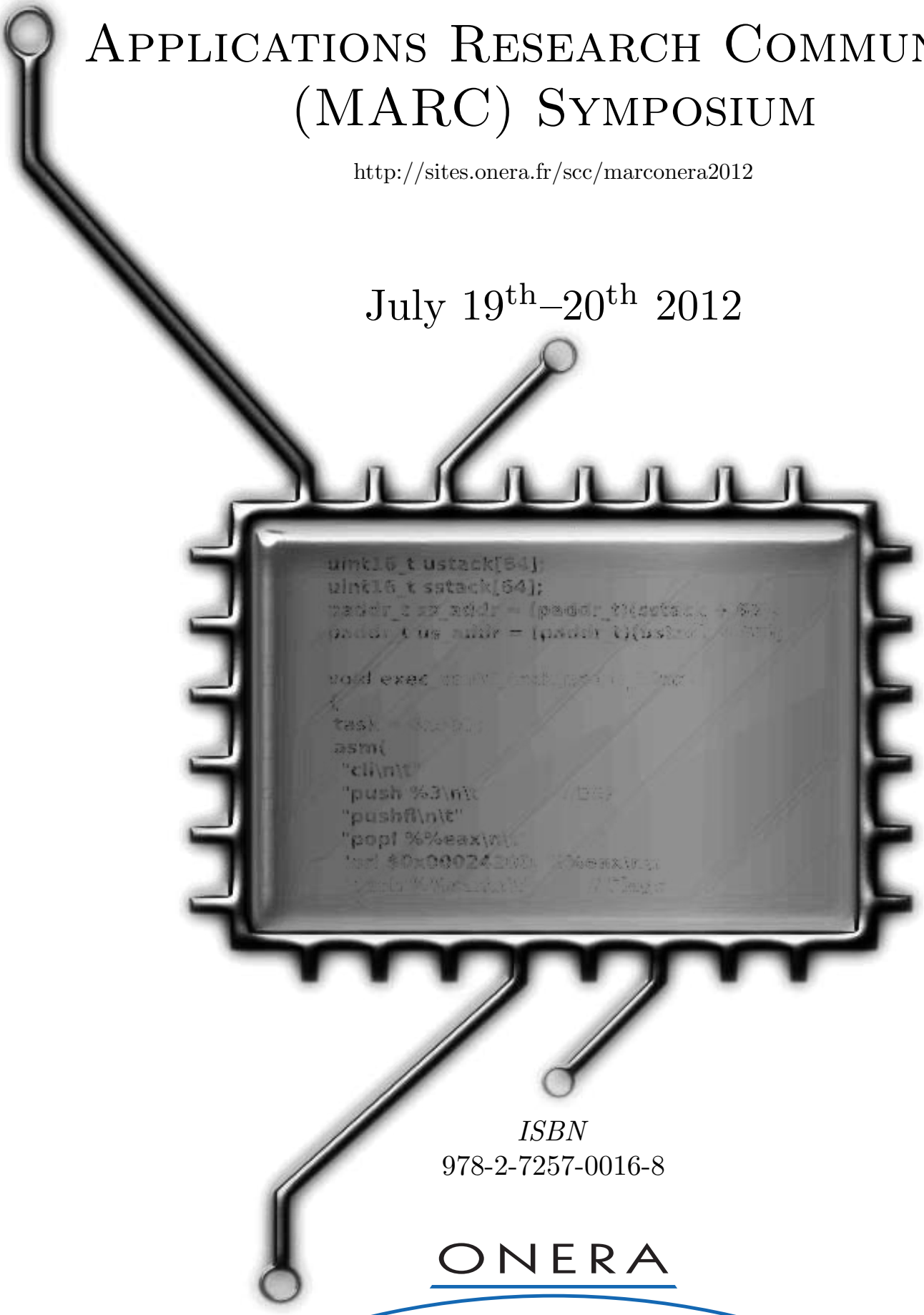
Proceedings of the 6th Many-core Applications Research Community (MARC) Symposium

http://sites.onera.fr/scc/marconera2012

July 19th–20th 2012



ISBN
978-2-7257-0016-8

ONERA
THE FRENCH AEROSPACE LAB

 ISBN: 978-2-7257-0016-8

Modelling Power Consumption of the Intel SCC

Patrick Cichowski*, Jörg Keller* and Christoph Kessler†
*Faculty of Mathematics and Computer Science
FernUniversität in Hagen, 58084 Hagen, Germany
Email: {patrick.cichowski,joerg.keller}@fernuni-hagen.de
†Dept. Computer and Information Science (IDA)
Linköpings Universitet, 58183 Linköping, Sweden
Email: christoph.kessler@liu.se

***Abstract*—The Intel SCC manycore processor supports energy-efficient computing by dynamic voltage and frequency scaling of cores on a fine-grained level. In order to enable the use of that feature in application-level energy optimizations, we report on experiments to measure power consumption in different situations. We process those measurements by a least-squares error analysis to derive the parameters of popular models for power consumption which are used on an algorithmic level. Thus, we provide a link between the worlds of hardware and high-level algorithmics.**

I. INTRODUCTION

Energy consumption by computers is continuously growing, and thus energy-efficient computing — sometimes embellished as the wider field covered by the buzzword *green computing* — gains more and more interest. The Intel Single-chip Cloud Computer (SCC) manycore processor chip supports energy efficiency by allowing the user to scale the clock frequency and the supply voltage of the cores (in groups of 2 and 8, respectively) and the on-chip network during runtime, providing many levels for both. Howard et al. [1] provide some data on power consumption of Intel SCC, but do not focus on energy efficiency. Albers provides a survey on algorithmic-level techniques for energy-efficient computing [2]. Those techniques rely on a parameterized model of power consumption by cores. Thus, in order to be able to exploit the algorithmic techniques known in the art on the SCC, it seems helpful to derive the model's parameters.

In order to obtain those parameters, we devise microbenchmark programs and machine settings for the Intel SCC and measure the power consumption. We subject the differences between those measurements and the power model to a least-squares error analysis and thus derive the model parameters. In this way we provide a missing link between the abstract algorithmic-level treatment of power consumption and the hardware-oriented view of power consumption for SCC.

The remainder of this article is structured as follows. In Section II, we briefly review the model of power consumption used on an algorithmic level. In Section III, we present the experimental setting, the measurements and their analysis. Section IV discusses related work. Section V provides a conclusion and outlook on further work.

II. POWER CONSUMPTION MODEL

Viewed from an abstract level, the dynamic power consumption of a semiconductor device (such as a processor core) is dependent both on the frequency with which the device is clocked, and on the supply voltage. Thus, for a specific device we might model its dynamic power consumption at frequency f (assuming a fixed supply voltage) by

$$p_{dyn}(f) = b \cdot f^a \ , \tag{1}$$

where b and a are device-specific constants. Typically, a is assumed to lie between 2 and 3 [2]. A semiconductor device normally is also assumed to have a frequency-independent static power consumption $p_{stat} = s$, where s is a device-specific constant. The total power consumption of the device at a fixed supply voltage then sums up to

$$p(f) = p_{dyn}(f) + p_{stat} = b \cdot f^a + s \ .$$

The static power consumption is ignored in most studies as it used to be only a minor fraction of the total power consumption. However, its importance is growing due to shrinking feature size, even in embedded systems [3].

A similar formula can be derived for the dependency on the supply voltage, given a fixed frequency. However, the two parameters voltage and frequency are not independent, as the minimum and maximum possible frequencies depend (among other things) on the supply voltage. To simplify our investigation, we will concentrate on the clock frequency in accordance with [2]. To still incorporate the voltage, we assume that for each frequency, the least possible supply voltage is used. This leads to more efficient energy use and to a more accurate model while still sticking to only one operating parameter. As the static power consumption is linear in the supply voltage, given that one does not approach the threshold voltage too much [4, Eq. 10] and as the minimum possible supply voltage for a given frequency on SCC can be approximated by a linear relationship, we get

$$p(f) = p_{dyn}(f) + p_{stat} = b \cdot f^a + s \cdot f \ . \tag{2}$$

In order to compute the energy consumption during a time interval $[t_1; t_2]$, we can either multiply the power consumption with the length $t_2 - t_1$ of the time interval if the power consumption remains constant during that interval, or we have

to split the interval into sub-intervals with constant power consumption and sum up the energy consumptions of the sub-intervals otherwise. As the power consumption turned out to be approximately constant during our experiments of fixed length, energy and power consumptions are proportional, and thus only power is considered further.

The algorithmic techniques reviewed by Albers [2] either assume that frequencies can be scaled continuously, or that a finite number of discrete frequency levels are available. Also, some techniques use the features that cores can be put into energy-saving mode or even switched off if not needed. While frequency change is normally considered to happen atomically, i.e. without a time or power penalty, switch-off of cores implies both. Yet, we will restrict ourselves to frequency scaling and not consider switch-off in our current investigation.

The Intel SCC consists of 24 dual-core tiles, i.e. 48 cores. The cores are organized in 24 frequency islands, one for each tile with 2 cores, and 6 voltage islands each comprising 8 cores, for the purpose of dynamic frequency and voltage scaling. In order to change the frequency and voltage during the runtime of a program, the RCCE library, which is provided with the SCC, offers some power management functions. It is possible to change the frequency and voltage of the different islands separately. Another and, for our experiments, more important possibility is to change only the frequency of the cores and let the voltage automatically scale to the lowest stable state. In this case there are only 6 so-called *power domains*, which are equal to the voltage islands [5]. For changing the frequency, one has to set a *frequency divider* between 2 and 16 for each power domain. The frequency divider is an integer value with which the global reference clock frequency of 1.6 GHz is divided.

Thus we can vary the frequency (and voltage) as depicted in Tab. I.

TABLE I
TILE FREQUENCIES, RCCE FREQUENCY DIVIDERS AND VOLTAGES [6, PP. 39-40]

Tile Frequency (MHz)	RCCE Frequency Divider	Voltage
800	2	1.1
533	3	0.8
400	4	0.7
320	5	0.7
266	6	0.7
228	7	0.7
200	8	0.7
178	9	0.7
160	10	0.7
145	11	0.7
133	12	0.7
123	13	0.7
114	14	0.7
106	15	0.7
100	16	0.7

The cores are interconnected by an on-chip network, which also connects them to 4 on-chip memory controllers. Both on-chip network and memory controllers are frequency islands of their own. At system start, the on-chip network can be scaled at frequencies 1.6 GHz and 800 MHz [6, p. 22]. The memory controllers' frequency normally is not scaled because of their connection to the off-chip memory banks that need a fixed operating frequency. Thus in the following, we assume the memory controllers' power consumption to be static. This is a simplifying assumption, as the power consumption not only depends on voltage and frequency, but also on the application. If the cores and network are scaled down, while the memory controllers keep their frequency, then from the controllers' perspective the application changes, i.e. does fewer accesses [1].

In the following, we will denote by indices c, n and m functions or constants belonging to cores, network, and memory controllers, respectively. Thus, if the 6 power domains are run at frequencies f_1 to f_6 and the on-chip network at frequency f_0, then the power consumption of the Intel SCC would be modelled as

$$\begin{aligned} p_{scc}(f_0,\ldots,f_6) &= p_n(f_0) + p_m + \sum_{i=1}^{6} 8 \cdot p_c(f_i) \\ &= b_n \cdot f_0^{a_n} + 8 \cdot \sum_{i=1}^{6} b_c \cdot f_i^{a_c} \\ &\quad + s_n \cdot f_0 + s_m + 8 \cdot \sum_{i=1}^{6} s_c \cdot f_i \ . \end{aligned} \quad (3)$$

The experiments of the following section are designed to experimentally derive the numerical values of b_n, b_c, a_n, a_c, s_n, s_m, and s_c. Normally, algorithmic power models [2] only consider the power consumption of complete chips. Therefore, we reduce the number of parameters by assuming $a_n = a_c$.

III. EXPERIMENTAL ANALYSIS

In each experiment, we fix the frequencies $f_0, \ldots, f_6$, run a microbenchmark program with one of four different settings on the SCC, and measure the power consumption one thousand times during the run of the program. For our measurements we use the FPGA on the Rocky Lake Board, which supports the direct measurement of voltages and currents of the domains. The program can read the actual voltage/current values through FPGA register access. Also, each experiment is repeated five times. We then compute the average power consumption from all measurements in each repeated experiment. When we put the frequency values and the measured power value into Eq. (3), we get a non-linear equation with the six unknowns b_n, b_c, a_n, s_n, s_m, s_c.

We devise a large number of frequency settings as explained below, and thus get a corresponding number of equations. We compute approximate values for the unknowns with a least-squares error analysis. Put shortly, for a set of values c_n to s_c and a set of values f_0 to f_6, the power consumption p_{scc} computed by Eq. (3) differs from the power consumption p_{exp} measured in the experiment, and thus produces a squared error $(p_{scc} - p_{exp})^2$. The analysis derives a set of values c_n to s_c such that the sum of the squared errors from all experiments is minimized.

Note that in our preliminary experiments, we did not vary the network frequency f_0, and thus the term $p_n(f_0)$ can also be considered static, which reduces the number of unkowns, and results in the following equation.

$$p_{scc}(f_1, \ldots, f_6) = 8 \cdot \sum_{i=1}^{6} (b_c \cdot f_i^{a_c} + s_c \cdot f_i) + \tilde{s} \ , \quad (4)$$

where $\tilde{s} = p_n(f_0) + s_m$.

We will explain below how to split $\tilde{s}$ into its components with the help of power measurements from [1]. In order to compute the parameters as accurately as possible, we decided to use a large range of frequencies. In order to restrict the number of experiments, we split the cores into two groups of sizes $8k$ and $48-8k$ each ($k = 0, \ldots, 6$), where one group has a higher frequency and the other group has a lower frequency. The cases $k = 0$ and $k = 6$ are special cases in which all cores have the same (higher or lower) frequency. We choose the frequencies 800 MHz, 533 MHz and 400 MHz for the high-frequency groups and 200 MHz and 100 MHz for the low-frequency groups. This results in $5 \cdot 3 \cdot 2 + 5 = 35$ experiments.

As a microbenchmark program we implemented a RCCE program whose execution can be divided into two steps that are detailed further below. In this program, one of the 48 cores is the *coordinator*, which coordinates the groups and measures the power consumption, while the other ones are the slaves. It is important to note that in every power domain only one core, the *power domain master*, can change the frequency and voltage of the corresponding domain. Hence, there are 6 power domain masters and one coordinator, which is one of the power domain masters.

The user starts the microbenchmark program with four parameters: the number of high-frequency domains, the high-frequency divider, the low-frequency divider and the setting for the microbenchmark. The settings allow to have several microbenchmarks, that differ in the use of caches, and intensity and regularity of memory access. There are four settings in total (cf. table II). In the first setting only one integer variable is used, which is initially set to 0. In the other settings we use an array with one million elements. These elements are initially 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.

TABLE II
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 step of the program all cores verify if they are a power domain master or not, and send the result (my_rank if they are a power domain master, -1 otherwise) to the coordinator. The coordinator saves the ranks of the power domain masters in an array and sends to each power domain master its array index, incremented by 1. In this way, we make sure that each power domain master has a unique *domain-master rank* between 1 and 6. Thus each power domain master can, from the user-defined number of high frequency domains, find out to which group its power domain belongs to. After that, the power domain masters of both frequency groups scale the frequency (and thus also the voltage) to the user-defined frequency. At the end of step 1, all cores are synchronized by a barrier to make sure that all frequency and voltage scalings for both groups have finished and all cores can begin with the second step simultaneously.

In the second step we use a time controlled loop and simulate an expensive calculation on each core for 10 seconds. The calculation depends on the chosen microbenchmark setting. In the first setting the integer variable is incremented by 1 within the loop. This represents a microbenchmark with use of ALU and caches, and few memory accesses. 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 (cf. table II) to add up the array elements, which represents higher cache miss rate and thus higher memory traffic. While working in the loop, the coordinator also measures the power consumption of the whole SCC every 10th millisecond. Thus we get an averaged result over 1000 measurement points for each microbenchmark setting. In this way we obtain many results for different situations and can compare these results to each other.

Figure 1 shows the results of the four microbenchmark settings for a high frequency of 800 MHz and the low frequencies of 100 MHz and 200 MHz, respectively. We can see that the results of the different microbenchmark settings are very similar to each other. The results for the other core frequencies (533 MHz and 400 MHz) differ only in that with decreasing the high frequency the power consumption also decreases with an increasing number of high frequency domains. Thus we only show the results for 800 MHz as an example for all results that we obtained. Also the workload over the 1000 measurement points within each experiment is quite stable. It mostly varies in a range of 2 Watt. There are less than 5 outliers on each experiment.

Figure 1 depicts the average power consumption from each of the 35 experiments. By a least-squares error analysis of equations obtained by inserting the frequency values and a fixed value of 3 for a_c into Eq. (4), we obtain the values $b_c \approx 2.015 \cdot 10^{-9}$ Watt/MHz^{a_c}, $s_c \approx 10^{-6}$Watt/MHz and $\tilde{s} \approx$ 23 Watt. The average error is 1.95 Watt, which is 5.58% of the average power consumption (averaged over all experiments). The relative error ranges from -14.66% to 24.73%. Thus, our model matches the experimental data quite well.

The comparison between measured and modelled power

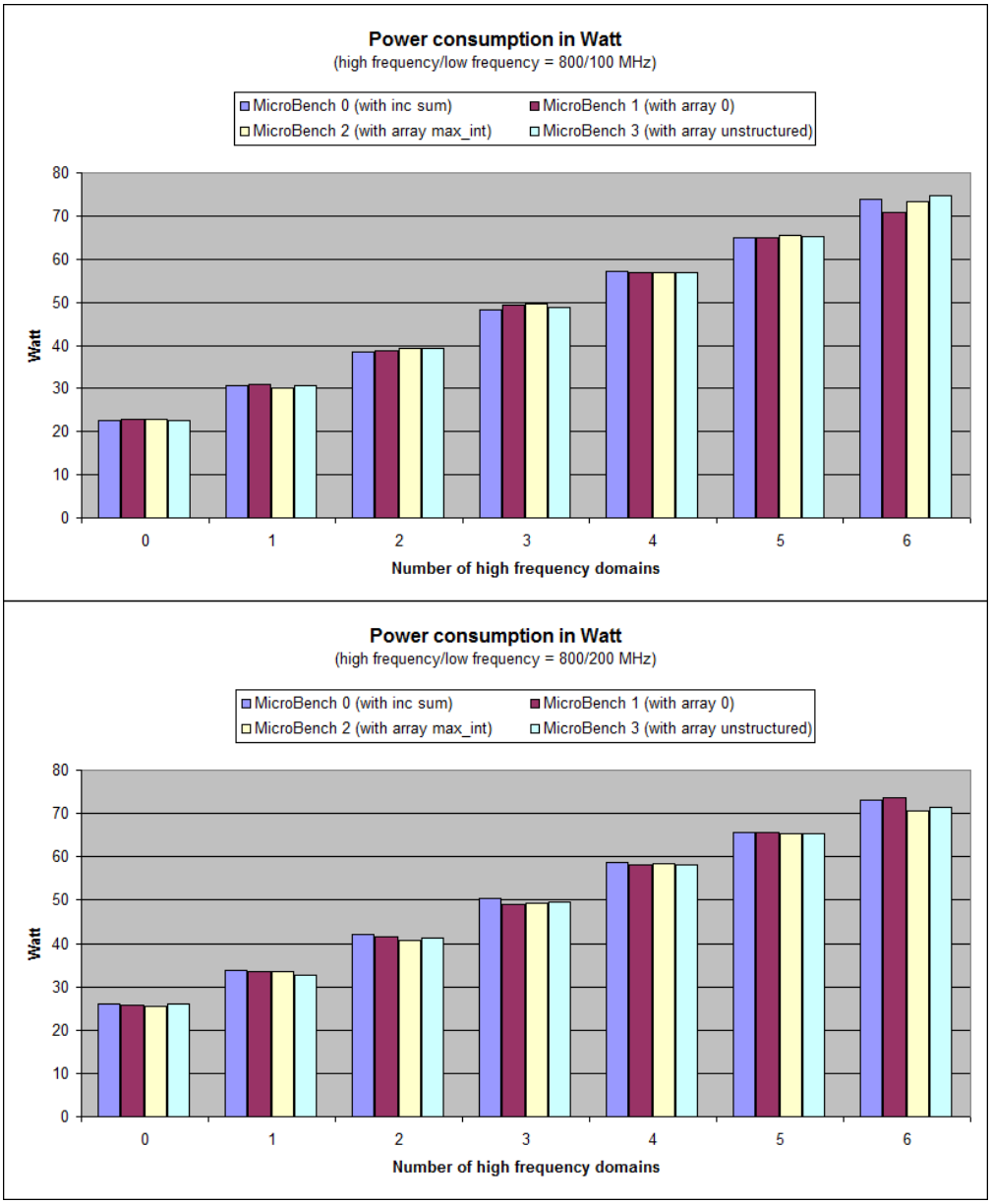


Fig. 1. Power consumption of the SCC for all microbenchmark settings with a different number of high-frequency domains, for the frequency 800 MHz. The low frequency is kept at 100 MHz (top) and 200 MHz (bottom), respectively.

consumption is depicted in Fig. 2. In this figure each curve or peak represents the results of one microbenchmark for the different numbers of high frequency domains. There are totally 24 peaks, which are organized clockwise as follows: On the right side of the circle each pair of four peaks are the results of the four microbenchmarks with a high frequency of 800 MHz, 533 MHz and 400 MHz (from top to bottom) and with a low frequency of 100 MHz. On the left side of the circle the peaks are represented in a reverse order (from bottom to top) and with a low frequency of 200 MHz.

Note that the measurements for all cores running at the same frequency ($k = 0$ or 6) correspond quite well to the numbers reported in [1, Fig. 13]. There cannot be an exact comparison as we do not know their benchmark program.

According to [1, Fig. 14], the network and memory controllers consume between 18.4 Watt (at cores 125 MHz, network 250 MHz) and 35.7 Watt (at cores 1 GHz, network 2 GHz). As we do not change the frequencies for network and memory controllers in our experiments, the value for the static power consumption of around 23 Watt seems to match an interpolation between those values quite well. One has to take into account that the frequency for the network and memory controllers in our experiments is fixed to only 800 MHz for both. Thus the voltage of these components is around 0.75 V

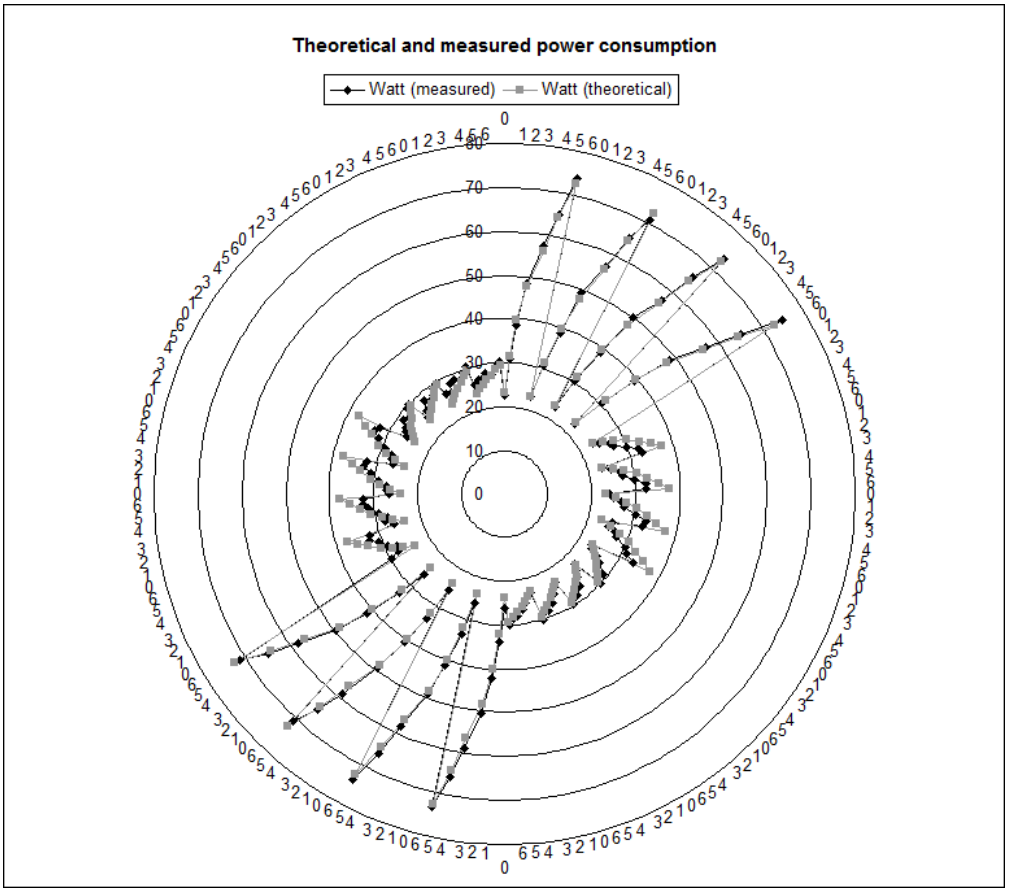


Fig. 2. Comparison of measured and modelled power consumption after least-squares error analysis.

and the static power consumption is as expected nearer to the lower bound of 18.4 Watt. As we can only measure the power consumption of the complete chip, we cannot avoid to incorporate measurements done by the chip manufacturer [1].

IV. Related Work

Ioannou et al. [7] consider dynamic voltage and frequency scaling for the SCC and implement a hierarchical power management with phase prediction to minimize energy consumption, balancing the trade-off between energy consumption and execution time for the various computation phases of MPI programs at runtime. Their approach is not based on using an explicit power model derived off-line, but on iterative adaptation of frequency and voltage within a performance window to react to changed computation patterns. The dynamic adaptation takes into account information for the local power domain as well as for groups of multiple power domains. They report on average improvements of 11.4% in the energy-delay product, with an average increase in execution time by 7.7% compared to running constantly at maximum frequency.

Putigny et al. [8] propose a performance and power model for SCC based on the core frequency. The model can be used for predicting the behavior of regular code such as dense linear algebra kernels that can be suitably characterized by a few statically accessible parameters. Their power model does not consider static power and does not model the network power. The constant coefficient of the dynamic energy consumption is not derived, and the scenario used here of combining voltage and frequency is not mentioned.

Gschwandtner et al. [9] investigate the impact of core frequency and voltage on the performance and power consumption of three major benchmark applications on SCC. They find that the benefits of core frequency and voltage scaling depends on the program, in particular whether it is compute-bound. They also demonstrate that, depending on the application, the energy-optimal frequency for an entire application can be at an intermediate level, both lower than the maximum possible frequency (which yields minimum execution time) and higher than the minimum possible frequency (which yields minimum power usage). In contrast to our approach, their work is based on measurements for entire applications and does not derive an energy model that could be used for predictions to support optimizations at a fine-grained level.

Kiertscher et al. [10] present an energy saving daemon for clusters called cherub, which can interact with different resource management systems to make them energy-aware. In this daemon they use the most important modes from standards and specifications like the Intelligent Platform Management

Interface (IMPI) or the Advanced Configuration and Power Interface (ACPI) specification to save energy. The cherub daemon polls the state of the cluster in regular intervals to gather the state of the different nodes and the load situation. With this imformation it can execute some actions to switch the states of the nodes and also distribute their load to other nodes. In this way cherub makes the use of a cluster more efficient in terms of workload and energy.

V. CONCLUSIONS AND FUTURE WORK

Our research provides the SCC-specific parameters for power consumption models used in algorithmic research on energy efficiency, thus providing a missing link between both worlds.

For the future, we plan to include changes of the network frequency to get more accurate measurements and analyses, and to use the insights gained in the present experiments to derive a power-optimal algorithmic mapping of streaming applications onto the Intel SCC.

We also plan to extend the model to include situations where cores can be switched off completely, which may be possible in future architectures [2].

Finally, we would like to refine our timing measurements to derive time penalties for changing voltage or frequency to widen the algorithmic applicability of our results.

Our power model considers the SCC at a quite high level only, and makes some simplifying assumptions to reduce the model complexity. A more detailed model constructed from microarchitectural simulation or analysis would be beyond the scope of our work that focuses on algorithmic level power modeling. Instead, a complementary, microarchitecture-agnostic approach could start from a generic but detailed algorithmic power model that includes many possibly relevant terms, involving parameters both from the algorithmic and architectural level, and which may or may not be so obvious from a high-level view onto the chip itself and its environment. The coefficients of this model could be calibrated by regression analysis (or other machine learning techniques) over training runs of benchmark programs with different characteristics, such as the Berkeley Dwarfs [11, Sect. 3] or similar computational kernels. This process automatically creates a SCC-specific power model from training data that can be used for algorithmic-level power predictions instead of our simplified model, for instance when deciding between different applicable algorithmic variants, resource allocation, choice of tunable parameters, or optimizing transformations for a computation. Similar techniques have been used for automatic model construction in other contexts, e.g. by Steinke et al. [12] for instruction-level energy modeling of an ARM7 processor or by Brewer [13] for algorithmic-level execution time modeling.

Another way of algorithmic-level power modeling can be applied after refactoring applications as combinations (such as serial, parallel and hierarchical composition) of a small set of algorithmic building blocks representing characteristic computational *patterns* [14] (where generic patterns are also known as *skeletons* in the literature [15]). For each such skeleton, a power model might be defined by interpolation of measurement tables or auto-calibration of a given generic model, and hence the power model for the overall application would be composed accordingly from the skeletons' power models

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 for this work.

REFERENCES

[1] 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*, vol. 46, no. 1, pp. 173–183, Jan. 2011.

[2] S. Albers, "Energy-efficient algorithms," *Comm. ACM*, vol. 53, no. 5, pp. 86–96, May 2010.

[3] A. Andrei, "Energy efficient and predictable design of real-time embedded systems," Ph.D. dissertation, Linköping University, The Institute of Technology, 2007, linköping Studies in Science and Technology, No. 1127.

[4] A. P. Chandrasakaran and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.

[5] Intel Labs, "Using the RCCE power management calls," Sep. 2011.

[6] ——, "The SCC programmer's guide," Nov. 2011.

[7] N. Ioannou, M. Kauschke, M. Gries, and M. Cintra, "Phase-based application-driven power management on the single-chip cloud computer," in *International Conference on Parallel Architectures and Compiler Techniques (PACT 2011), Galveston, Texas, USA*. ACM, 2011.

[8] B. Putigny, B. Goglin, and D. Barthou, "Performance modeling for power consumption reduction on SCC," in *Proc. 3rd Many-core Applications Research Community Symposium (MARC-3)*, Dec. 2011.

[9] P. Gschwandtner, T. Fahringer, and R. Prodan, "Performance analysis and benchmarking of the Intel SCC," in *Proc. 2011 IEEE Int. Conf. on Cluster Computing*. IEEE Computer Society, Sep. 2011, pp. 139–149.

[10] S. Kiertscher, J. Zinke, S. Gasterstädt, and B. Schnor, "Cherub: Power consumption aware cluster resource management," *IEEE/ACM GreenCom-CPSCom*, pp. 325–331, 2010.

[11] K. Asanovic *et al.*, "The landscape of parallel computing research: A view from Berkeley," Electrical Engineering and Computer Sciences, University of California at Berkeley, USA, Tech. Rep. UCB/EECS-2006-183, Dec. 2006.

[12] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel, "An accurate and fine grain instruction-level energy model supporting software optimizations," in *Int. Workshop on Power and Timing Modeling, Optimization and Simu lation (PATMOS)*, Sep. 2001.

[13] E. A. Brewer, "High-level optimization via automated statistical modeling," in *Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP'95)*, 1995.

[14] C. W. Keßler, "Pattern-driven Automatic Parallelization," *Scientific Programming*, vol. 5, pp. 251–274, 1996.

[15] M. I. Cole, *Algorithmic Skeletons: Structured Management of Parallel Computation*. Pitman and MIT Press, 1989.