Emulating Asymmetric MPSoCs on the Intel SCC Many-core Processor

Roy Bakker, Michiel W. van Tol, Andy D. Pimentel
Informatics Institute, University of Amsterdam
Science Park 904, 1098 XH Amsterdam, The Netherlands
R.Bakker@uva.nl, mwvantol@uva.nl, A.D.Pimentel@uva.nl

***Abstract*—The Single-chip Cloud Computer (SCC) is a 48-core experimental processor created by Intel Labs targeting the many-core research community. It has extensive frequency and voltage scaling support as well as on board power monitors. In this paper we present a detailed study of the power properties of the SCC. Then, we show how the SCC can be used as a substrate to emulate asymmetric multi processor systems on chip (AMPSoCs) to be used for studying power/performance trade-offs.**

I. Introduction

Single ISA, asymmetric multiprocessors can be exploited to achieve a better power/performance trade-off [1]. It is to be expected that we will see more asymmetric, single ISA, multiprocessor system on chip (or AMPSoC) solutions in the embedded domain where power and energy budgets are restricted. Currently, we see such platforms emerge in the mobile computing domain with the advent of the ARM big.LITTLE design [2] and the recent NVIDIA Tegra MPSoCs [3]. In this paper we use the term asymmetric (AMPSoCs) to indicate heterogeneous, single ISA based systems. We distinguish from general heterogeneous MPSoCs as it is common in the embedded domain to have different types of cores with different ISAs integrated in one system.

Schedulers and tools are required to efficiently exploit the capabilities of such emerging asymmetric platforms. Both during design space exploration (DSE) of a hardware-software co-design as well as at run-time we need to devise optimal mappings of applications to AMPSoCs.

To develop, train, test and validate these tools we require a platform that can deliver feedback in the form of power consumption for specific configurations. Traditionally this is done with rough estimates [4], power modeling tools such as McPAT [5], or FPGA prototypes [6]. However, these do not expose the actual properties and challenges of receiving power consumption feedback in real silicon MPSoC systems. Alternatively, evaluation boards for the aforementioned big.LITTLE or Tegra platforms, or the experimental Intel QuickIA [7] only provide fixed AMPSoC configurations.

In this paper, we propose to investigate the use of multi- or many-core processors with extensive DVFS support to act as a substrate to emulate the behavior of AMPSoCs. The platform we study here consists of the 48-core experimental Intel SCC many-core processor which is highly configurable in terms of frequency and voltage, and has a power measurement interface.

In Section II we compare our work with other asymmetric platforms as well as similar approaches to emulate asymmetry on homogeneous systems. We highlight the relevant architectural features of the SCC in Section III. Our first contribution is a characterization of the platform and its power behavior in Section IV. Then we show how we can use these properties to emulate power and performance of an MPSoC in Section V. To evaluate the proposed platform we construct a simple design space exploration experiment in Section VI which measures the power consumed by different emulated MPSoC configurations running two synthetic component based streaming applications where each component has a different workload character and we summarize our final findings and discuss future work in Section VII.

II. Related Work

The closest currently available asymmetric platforms to our approach here are the latest NVIDIA Tegra [3] platforms. Here, a quad-core MPSoC is accompanied by a fifth *companion core*, which is architecturally the same as the the four other cores, but is made with a special production process to be extra low-power and only runs at a lower clock frequency. The biggest difference with our platform is that the migration between the quad-core and companion-core is managed by the hardware; when the OS only uses one of the four cores and goes in a sufficiently low power state, the hardware will transparently migrate the state to the companion core. When the OS throttles the system up again, the state is migrated back to the quad-core. Therefore, we do not have full control over how the application is mapped and the performance settings of the system, as it is geared towards specific mobile use-case scenarios where the system is in standby most of the time, solely using the companion core.

Another available asymmetric platform is the ARM big.LITTLE architecture [2]. It consists of two cache-coherent clusters where one has high performance out-of-order ARM Cortex A15 cores, and the second has low power in-order Cortex A7 cores. Operating system support is required to migrate between the clusters, and both can be active at the same time. An evaluation board with this architecture is available and has been used in [8] to evaluate a hierarchical power management and migration approach. The downside of this platform is that frequency can only be set for a whole cluster, but the advantage is that power can be measured per cluster and not for the chip as a whole as with the SCC.

The third available asymmetric platform is the Intel QuickIA prototype board [7]. While not publicly available, it has been used [9], [10] to prototype scheduling techniques for asymmetric multicores. While [9] focuses on power efficient scheduling, a power estimation based on McPAT [5] is used instead of actual power measurements from the board.

Others [11], [12] have also emulated asymmetry by using DVFS on multicore/multiprocessor systems in the past allowing investigation and development of scheduling techniques. However, they only consider performance, and do not measure the power consumption. The generation of CPUs used in these experiments only supported clock modulation. A more advanced method was used in [13], which besides DVFS also used Intel proprietary methods to emulate in-order execution on a complex Xeon out-of-order core by restricting the number of retired operations per cycle. As the SCC cores are only in-order, we are not able to create such asymmetry. Similarly, in addition to using the QuickIA platform, [10] also used DVFS on multicores, as well as Intel proprietary methods to reduce cache sizes to introduce additional asymmetry. A future addition to our work on the SCC could be to investigate increasing asymmetry by selectively disabling the L2 cache for sets of cores.

An approach to simulate MPSoCs using the SCC was presented in [14]. However, this purely uses the SCC as a massively parallel computing platform to execute cycle accurate simulations of MPSoCs. This is unlike our approach which emulates AMPSoC behavior executing applications directly on the SCC which has less flexibility in architectural configurations, but is expected to have a better performance and accuracy trade-off.

III. Background

A. SCC platform

The Single-chip Cloud Computer (SCC) experimental processor [15] is a 48-core *concept vehicle* created by Intel Labs as a platform for many-core software research. Its features are intended to aid investigation into software and hardware technologies allowing to scale CPUs up to hundreds and potentially thousands of cores.

The 48 cores of the SCC are based on the Intel IA-32 P54C architecture, which features an in-order dual issue superscalar pipeline, 16 KB L1 instruction and data caches, and a unified 256 KB L2 cache per core. Both the L1 and the L2 are 4-way set associative with a cacheline size of 32 bytes, are *write back*, and do not allocate on write miss, i.e. are *write around*. The caches are not coherent between cores, and each core runs an individual Linux operating system instance.

The chip is divided into 24 tiles with two cores each organized on a 6x4 grid. Each tile has its own router to access the on-chip mesh network which has a 256 GB/s bisection bandwidth [16] and uses X-Y routing. The network connects the tiles to the four on-chip DDR-3 memory controllers, as well as to the voltage regulator interface and the FPGA-based

Table I
SCC supported frequencies and required voltage. Frequencies between 100-200 are left out on purpose.

Freq. div	Freq.	Volt.	Freq. div	Freq.	Volt.
2	800	1.16250	6	267	0.66875
3	533	0.85625	7	229	0.65625
4	400	0.75625	8	200	0.65625
5	320	0.69375	16	100	0.65625

chipset. The memory controllers and network bandwidth are highly over-provisioned for the cores, as we have shown in previous work [17].

The SCC is highly configurable and has many interfaces that can be mapped to memory addresses that can be accessed by a core. One example is the local 16 KB Message Passing Buffer (MPB) on every tile which is suitable for sending short messages between cores. With two cores per tile, each core has an 8 KB area in the local MPB only by convention. Other interfaces are the per-tile configuration registers, the voltage regulator, and the special control and status registers in the FPGA chipset. One of the most important registers in the FPGA is the global timestamp counter (GTSC) which can be used as a stable time source for correlating events on different cores. The FPGA also acts as a bridge between the SCC on-chip mesh and a PCI-express interface to a management PC (MCPC) which is used to initialize the SCC and to launch jobs on it.

B. DVFS Support

Voltage and frequency control are completely decoupled on the SCC. The frequencies can be controlled in every individual tile (two cores) by setting a divider from the base clock, which by default runs at 1600 MHz. This divider can be set between 2 and 16 in whole steps, which yields frequencies between 800 and 100 MHz. Setting the divider consists of writing a value to the clock configuration register of the tile, and can be changed on the fly affecting the tile clock instantaneously. The mesh network operates at either 1600 or 800 MHz and cannot be dynamically changed. Similarly, the memory controllers can be initialized at either 800 or 1066 MHz. We operate the mesh and memory controllers at 800 MHz in our experiments for this paper.

Voltages can be controlled by writing command values to the voltage regulator. The chip is divided into 8 voltage domains; one for the mesh network, one for the memory controllers and 6 domains for the cores. Each voltage domain for the cores consists of 4 tiles arranged in a 2x2 area, and its voltage can be controlled individually in 6.25 mV steps between 0 and 1.3 V. As a given frequency requires a minimum voltage, we use the values as measured by [18], shown in table I. The voltage regulator can only change the voltage of one domain at a time, and it takes around 12 ms for the voltage to settle before another change can be made.

C. Power Measurement Interface

The power infrastructure of the SCC is equipped with many sensor ADCs that can measure the supply voltages at different stages. However, the current can only be measured at a limited

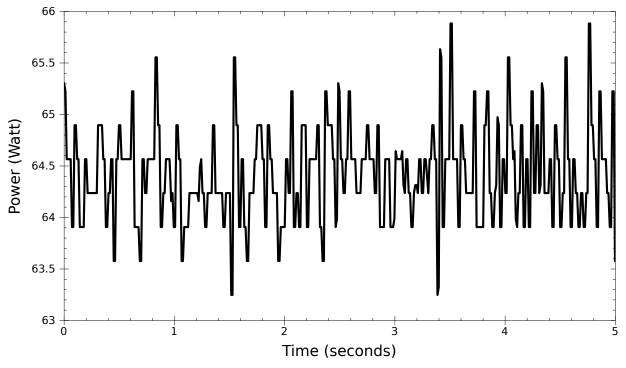

Figure 1. Plot of raw power measurement data with the cores running at 800 MHz in idle state.

set of stages, such as the current to the chip package on the 3.3 V rails. Only the voltage but not the current can be measured for the individual core and mesh voltage domains, which means power can only be measured of all cores and the mesh together as a whole.

The measurements are governed by the power measurement controller (PMC) which is situated in the FPGA. It periodically collects the data from the measurement ADCs and stores them in the power measurement registers which can be memory mapped and read by the cores or the MCPC. The controller can be programmed to read out only a select group of sensors which shortens the update cycle and therefore increases the measurement frequency.

IV. Platform Characterization

Before we can use the SCC as a substrate to emulate AMPSoCs, we investigate the method and accuracy to measure power, and determine the power characteristics of the platform.

A. Power Measurement Algorithm

In order to measure power, we need to read out both the current and the voltage, and configure the PMC accordingly to measure only these two values on the 3.3 V rail that supplies the cores and the mesh network. We measure these two values at the smallest sample time. With this setting, it turned out that for a period of 4 ms the values in the power registers contained invalid values, we assume while they are being updated by the PMC, followed by an 8 ms period with stable valid values which yields a total update frequency of 12 ms. We take this behavior of the PMC into account in the power measuring algorithm we developed; it only registers a new value after the value read from the power register changed from invalid to valid.

Using this measurement algorithm, it turned out that we cannot measure the current as accurately as we can measure the voltage. The resolution of the current is 0.1 A, while the voltage can be measured with a resolution of 0.004 V. As the power supply holds the voltage rail stable around 3.3 V, the measured voltage only varies slightly between 3.292 V and 3.308 V. However, with a resolution of 0.1 A, this results in a power measurement resolution of only 0.33 W. Increasing the sample time in the PMC did not increase the resolution of

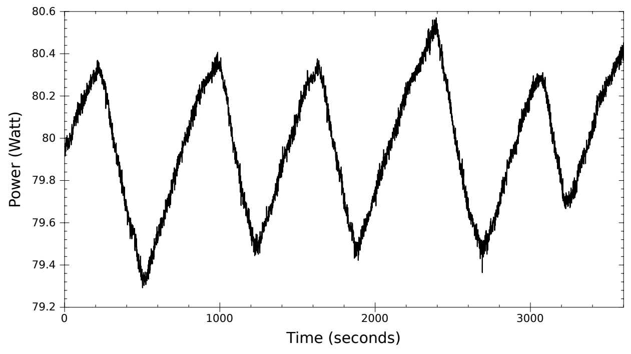

Figure 2. Long term power fluctuation pattern of the SCC chip.

the provided values. Furthermore, there is a large fluctuation in the raw measurements within a range of 0.3 A without any load on the cores. Figure 1 shows the resulting power calculated by multiplying the individual current and voltage values, measuring the power of the SCC running at 800 MHz over a period of 5 seconds.

To improve the resolution, we have adapted our algorithm to average the individual power measurements over a longer period of time in an attempt to filter out these fluctuations. Using this approach, while averaging over a 1 second time interval, we were able to detect much smaller changes in power down to approximately 0.05 W, as we will show towards the end of this section.

B. Power Fluctuations

We observed an effect while using our previously discussed techniques to collect power measurements. Our measurements showed a strange, but regular, power fluctuation over a long period of time as shown in Figure 2. In this experiment, we measure the power over one hour, while the cores run at 533 MHz without being booted. As can been seen in the figure, the cores and mesh consume around 80 W of power, which slowly ramps up for about 500 s, and then drops down again in approximately 250 s, creating a slow moving 750-second period sawtooth form, with an amplitude of almost 1 W.

We suspect, but could not get this confirmed, that this long term power fluctuation is caused by the charging and decharging of a capacitor used as a stabilizer in one of the voltage regulator circuits, which is located between the measurement ADC and the SCC chip. There, the 3.3 V supply voltage is transformed to the requested voltages that are set for the individual voltage domains on the chip, in the 0.65 V to 1.3 V range. In order to understand this behavior, and therefore the deviation in our individual power measurements, we set up an experiment to characterize this fluctuation. In this experiment we measured the power over a period of 1.5 hours, which yields 7 whole periods of the fluctuation, and repeated this for many different voltage configurations of the chip. We start with all six core voltage domains at 1.25 V, and then decrease each domain one by one to 0.0 V in steps of 0.25 V. For each power level we measured the distance between the minimum and maximum absolute amplitude from the average power, and computed the relative amplitude. These results are

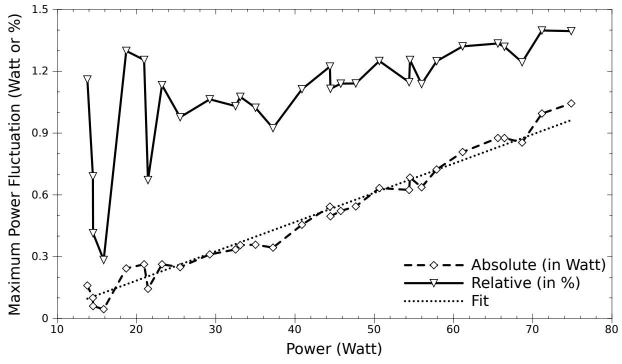

Figure 3. Fluctuation in percentage of the average for different power states.

shown in Figure 3, which shows that the amplitude of the fluctuation is proportional to the amount of power consumed by the chip.

Using the data from this experiment, we made a fit to express the relation between the power consumption and fluctuation, and in the worst case the distance between the minimum and maximum value of the deviation is 1.42% of the consumed power. This means that if we assume that the *actual* power is the average around which the power fluctuates, that the error induced in a measurement is 0.71%. Furthermore, as we know that the fluctuation moves rather slowly, we know that the fastest maximum change in observed power is 1.42% over a duration of 250 seconds, on the down going flank of the power fluctuation pattern. Therefore, if we want to compare two power consumption measurements t seconds apart, and are only interested in the relative change, we know that the maximum error introduced by the fluctuation in these t seconds is $\frac{min(t,250)}{250} \cdot 1.42\%$. To give a concrete example, if we want to measure the power difference between two consecutive measurements 5 seconds apart at a power level around 75 W, we know that the maximum error in this difference is $\frac{5}{250} \cdot 1.42\% \cdot 75\ W = 0.021\ W$, while the absolute error in the values measured is $75\ W \cdot 0.71\% = 0.5325\ W$.

As we have learned that the fluctuation is directly related to the amount of power consumed, it is important for the accuracy of future measurements to keep the total power consumption of the chip as low as possible. This means that we should reduce the voltage for components we do not use to reduce static power, and the reduce or disable clocking to reduce dynamic power. For example, we can set an unused voltage domain to the lowest value possible and shut down unused cores and their L2 caches by using a clock gating feature.

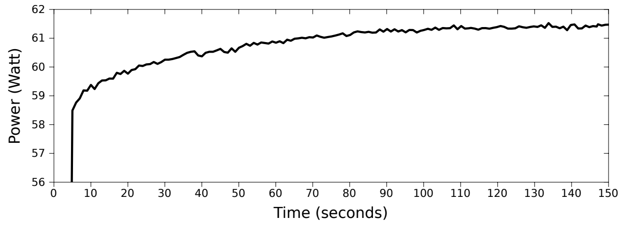

Figure 4. Effect of temperature on power consumption.

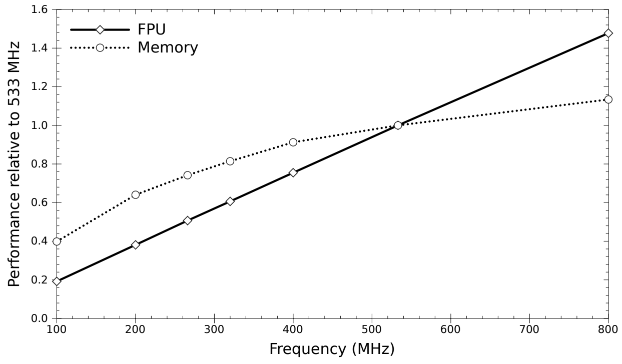

Figure 5. Relative performance of the Memory and FPU microbenchmarks running on core 0 using different frequencies, compared to 533 MHz.

C. Temperature Effect

Another physical influence on power consumption is the temperature of the chip. As conductivity increases with higher temperatures, the chip will consume more power. We can measure this effect with an experiment of which the results are shown in Figure 4. Here we use DVFS to change an idle chip from 100 MHz to 800 MHz after 5 seconds, while measuring the power consumption. We see an instantaneous increase in power from around 18 W (not shown) to 58.5 W when the frequency and voltage are changed. As a result, the chip then slowly starts to warm up, creating another gradual power increase to 61.5 W over a time of 150 seconds. After this the power consumption is stable and only shows the earlier discussed fluctuation. When we decrease the frequency and the chip cools down again, we can observe the reverse temperature related settling behavior.

D. Workload Characterization

In order to characterize the power consumption for different types of operations, we have developed 5 types of computational kernels to use as microbenchmarks. First, we have a kernel that consists mainly of FPU operations named *FPU*, and a similar kernel with Integer arithmetic named *Integer*. We have three memory bound kernels that use `memcpy()`, where we have constructed the access patterns in such a way that one purely stresses the L1 cache (*L1*), the second stresses the L2 cache (*L2*), and the third will always access the off-chip memory (*Memory*).

We have calibrated our kernels at 533 MHz to have approximately the same execution time for a given number of iterations. As the core and its L1 and L2 caches all run on the same tile clock, the performance of all kernels scales linearly with the clock frequency, except the Memory kernel. The performance of the Memory kernel depends on traversing the mesh network and the latency of the memory controller, which are independent of the clock scaling of the tile. We show the difference in performance scaling per clock frequency in Figure 5, where we only compare the Memory and FPU kernel, as the other three kernels have the same scaling behavior as FPU.

In order to measure the energy consumption effect of periodic processes we adapted our computational kernels to

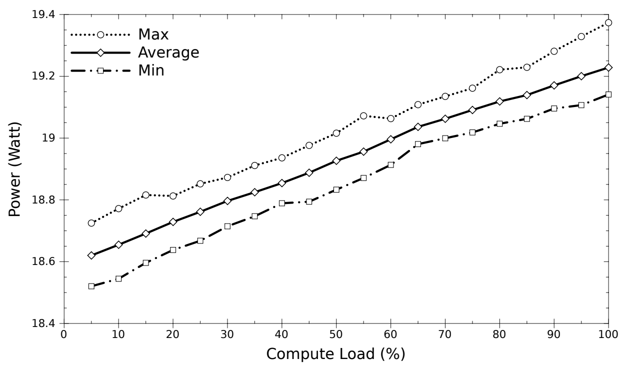

Figure 6. Power consumption of FPU kernel with increasing load on single core at 800 MHz, showing the Min/Max envelope around the average.

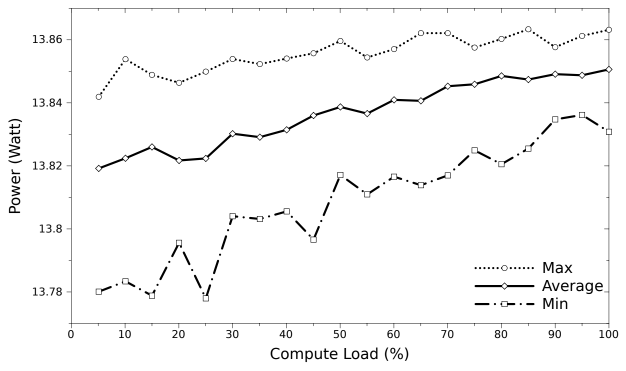

Figure 8. Power consumption of L1 kernel with increasing load on a single core at 100 MHz, showing the Min/Max envelope around the average.

execute in a duty cycle alternating with calls to `usleep()` to allow for different levels of workload. The Linux images we run on the cores are configured as minimal as possible, therefore there are no other processes to schedule and the `usleep()` call will result in the scheduler entering its idle loop. The idle loop executes the `HLT` instruction which stops the core and saves power, until the core wakes up again on the next timer interrupt which signals the start of the next time slice and our computational kernel is rescheduled again. The time spent in `usleep()` is unpredictable (in the order of 1 to 2 ms) as it depends on the time until the start of the next time slice, we made our duty cycle self calibrating by measuring the time spent in the `usleep()` call and automatically adapting the computational load. This way we can gradually and accurately vary the generated load on a core between 0 and 100%.

Using this method we ran experiments on a single SCC core (core 0) while having the rest of the chip powered down, measuring the power consumption for loads varying from 5% to 100% in 20 steps, for each of our 5 microbenchmark kernels. Every individual step is measured for 5 seconds, and is repeated for all frequency/voltage combinations listed earlier in Table I, except for 229 MHz. This whole process was repeated 10 times and averaged to mitigate the fluctuation effect described in IV-B. In Figures 6, 8 and 9 we show this average together with the minimum and maximum values.

Figure 6 shows the result for the FPU kernel at 800 MHz, and it shows that we can clearly measure the effect on power for the increasing load on the core, detecting power changes approximately as small as 0.05 W. It also shows, as expected because dynamic power is affected, that our duty cycle approach delivers a linear relation between the induced load and the power consumption. Furthermore, we were also able to measure differences between the different types of kernels. This result is shown in Figure 7, where we compare the average power consumption (over 10 runs) for increasing load for all five kernels at 800 MHz. Due to the fact that a core can have only one outstanding memory request, the Memory benchmark consumes less power as the core is stalled most of the time, waiting for the memory request to return. In contrast, the L1 benchmark consumes the most power, as it is accessing the L1 cache at a high rate while stressing the cache and virtual memory logic. Interestingly enough, we cannot observe a significant difference in power consumption between FPU and Integer arithmetic.

When we measure the power for different loads on a core on the lowest frequency setting, 100 MHz, we see that we have much less accuracy at such low power levels as the differences in power are much smaller than the fluctuation pattern. As can be seen in Figure 8 for the L1 kernel at 100 MHz, the uncertainty between the min and max values is greater than the difference between 0 and 100% load. While our average value still shows a clear linear relation to the load, it is impossible to see clear differences for real-time measurements. We can only be guaranteed to see a difference in power when the maximum value at 0% load is smaller than the minimum value at 100%

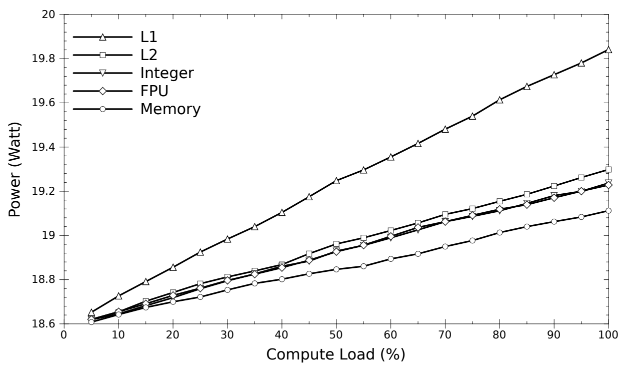

Figure 7. Power consumption with increasing load on a single SCC core running at 800 MHz.

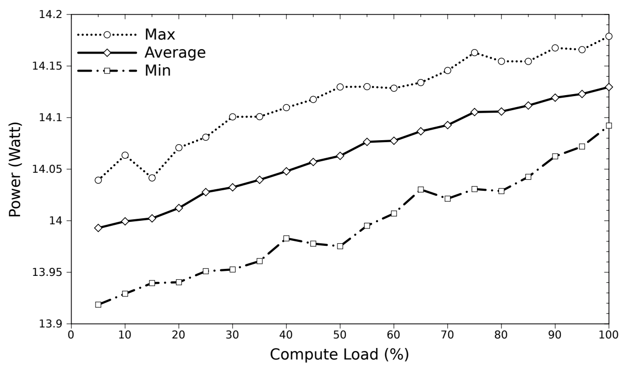

Figure 9. Power consumption of FPU kernel with increasing load on single core at 320 MHz, showing the Min/Max envelope around the average.

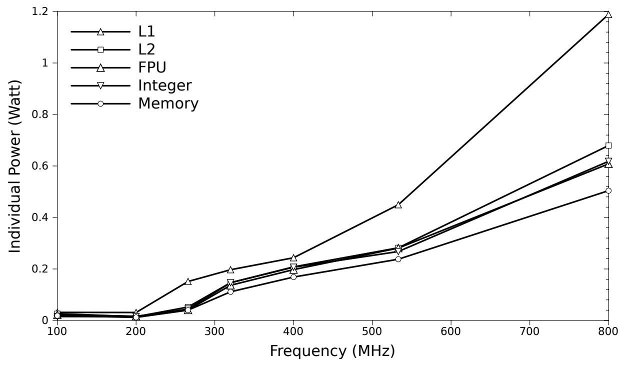

Figure 10. Power consumption of our microbenchmark kernels for different frequencies, after subtracting idle power.

load. We determined that the lowest frequency setting at which this is still the case is at 320 MHz, as shown by an experiment using the FPU kernel in Figure 9.

Using the averaged values, we can determine the power characteristics of our kernels for different frequencies. This is summarized in Figure 10. We see that using the L1 cache clearly consumes twice the amount of power than when we compute Integer of FPU or access the off-chip memory.

We have measured the mesh network to consume 11.9 W of power, when we powered down all the core voltage domains.

V. Using the SCC as an AMPSoC

By using the configurable voltage islands and fine grained frequency control per tile, we can set up the SCC in many different asymmetric configurations. Using our power measurement infrastructure, it can then act as a substrate to emulate power aware AMPSoC systems.

As a concrete example, we can emulate a 6-core asymmetric system by using one core per voltage domain, so that all six cores are able to run at different frequencies and voltages. All unused tiles are then set to 100 MHz and put in the lowest possible power state using clock gating. This will create an asymmetric architecture although the cores (ISA and pipeline) are the same. While we can not influence the routing on the mesh network, we can choose to use any core within a voltage domain. Furthermore, we can configure which of the four memory controllers are used by each core. Additional heterogeneity can be introduced by selectively disabling L2 caches and/or FPUs. However, only disabling the cache will result in a reduction in power consumption as the FPU can not be clock gated.

As an extension to this, we can allow multiple cores on the same voltage island to participate in the system, creating an asymmetric system composed of homogeneous groups of cores. Even more asymmetry can be created by setting different frequencies between these cores in a group, though the voltage is bound to the core with the highest frequency in the voltage domain, which is the most dominant factor in power consumption.

Using this technique, we are able to run and study different kinds of application mappings on a static system. In this case, a static system means that the number of cores and DVFS settings are fixed during run-time, but we can reconfigure the platform to emulate a different fixed system between runs. We did not experiment with network and memory effects by using different placements, but we expect the effect to be very small as both network and memory bandwidth are over-provisioned [17].

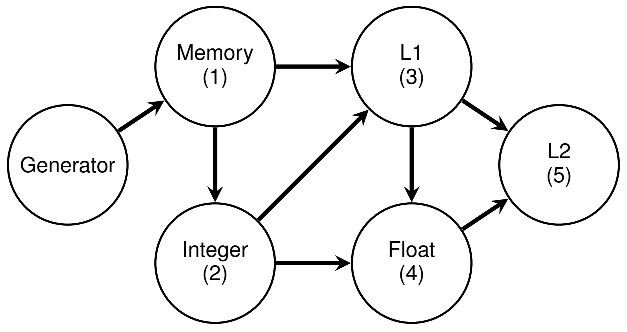

Figure 11. Example of a Process Network with 6 cores which is used in experiment 1.

A. Process Network Emulator

To validate our AMPSoC emulation approach we need to be able to run distributed applications on different configurations to show the effect on power consumption and performance. We created a Process Network Emulator that mimics an application workload specified as a Process Network. Based on a simple graph description, we instruct the SCC cores to read synchronization tokens from a *push* FIFO, implemented in their local Message Passing Buffer (MPB). The first core runs a generator that emits tokens with a configurable interval, while the others read a token from their FIFO and perform a (fixed) computation of a certain type. After each computation step it updates the token and forwards it to all destination processes located on different cores. By fixing the workload that needs to be processed per token, we can easily see what the differences are in terms of energy consumption and execution time for mappings on cores with different DVFS settings.

The Process Network Emulator can be configured to execute different kinds of workloads per process, based on the microbenchmark kernels we presented in Section IV-D. Therefore it contains the CPU intensive kernels, with either integer of floating point operations, and memory intensive kernels, for access to respectively L1, L2 and off-chip RAM. The amount of workload required to process a single token is configurable, or can be disabled to measure the raw communication latency and throughput.

Figure 11 provides a graphical representation of one of the process networks we used for the evaluation in Section VI. Each node represents a process and each edge represents a communication channel (FIFO) between processes.

VI. Evaluation

A. Emulating a 6-core asymmetric MPSoC

In this section, we describe and analyze two experiments to illustrate and verify how we can use the SCC as an emulation

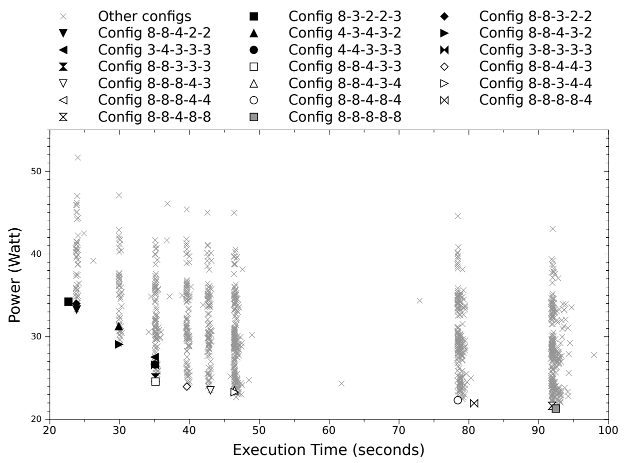

Figure 12. Results for the first Process Network execution at a permutation of the frequencies 800 (2), 533 (3), 400 (4) and 200 MHz (8).

substrate for a 6-core AMPSoC. We limit ourselves to four different DVFS settings, (800, 533, 400 and 200 MHz), in order to be able to measure every possible configuration within a reasonable amount of time. We run the token generating process always on 100 MHz as it does not perform any computation, which leaves us with $4^5 = 1024$ possible platform configurations. We keep the process network mapping static, where all processes are pre-assigned to a specific core, i.e. a one-to-one mapping. The process numbers 1 to 5 as shown in the network graphs and the corresponding frequency dividers 2, 3, 4 and 8 are used to identify the configurations in this section. For example, configuration 8-4-3-2-4 means that process 1 (Memory in both networks) runs at divider 8, i.e. an 800 MHz core, process 2 on divider 4, and so on.

The first experiment focuses on performance, i.e. a fixed workload done as fast and (energy) efficient as possible. We show the trade-off between power consumption and execution time, and the trade-off between energy and execution time. The second experiment uses a more realistic continuous streaming process network for which we check if the AMPSoC configuration is able to execute the network properly (i.e. it is able to keep up with the demands of the workload and meet the *deadline*) and explore which of the satisfactory configurations consumes the least amount of power.

B. Results

The first experiment is based on the network shown in Figure 11. We configure the computational workloads to process for 20 ms on each received token, calibrated at a frequency of 533 MHz. The FIFO size is set to 240 in this experiment, the maximum our MPB based protocol supports, and 400 tokens are put into the network. The results for all 1024 configurations consist of the measured average power consumption and time to completion. In Figure 12 the power/performance trade-offs for these 1024 mappings are shown. The Pareto-optimal configurations are individually marked and shown in the legend, while all other points are marked with a grey cross. We see that depending on the configuration there can be large differences in both power consumption and execution time.

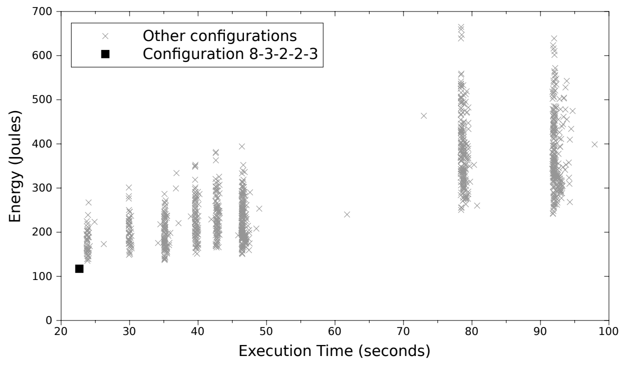

Figure 13. Energy consumption for computation, corrected for surrounding SCC cores not participating.

In this experiment we also calculated the amount of additional energy for the different configurations. In the calculation we corrected for the amount of energy that the unused cores on the same voltage island consume while their voltage level is raised up. We do this by measuring the idle power consumption for the chip at the used frequency settings and subtract an equal part for the non-participating cores. Figure 13 is generated based on this corrected power, multiplied by the execution time to calculate the total amount of energy consumed by the application. In this case, we see that there is still a significant overhead by the network and other leakage power resulting in the conclusion that running faster on a high frequency is more efficient than running for a longer period on a lower frequency.

In the second experiment we simulate a periodic application, where tokens are put into the network every 50 ms. The network has been more balanced by removing some edges from the network graph as shown in Figure 14, and uses a computational load that can easily be performed within the time the next token arrives on a sufficiently high frequency. The FIFO size is set to 5 tokens. During normal execution, the execution time for this network for 200 input tokens is just over 10 s, if there are no stalls. We use the same permutations of frequencies as in the previous experiment, again resulting in 1024 different configurations.

The results for the 1024 configurations are shown in Figure 15. We immediately see that there are several configurations that cannot keep up with the requested workload resulting in an execution time much higher than 10 s. Some have even

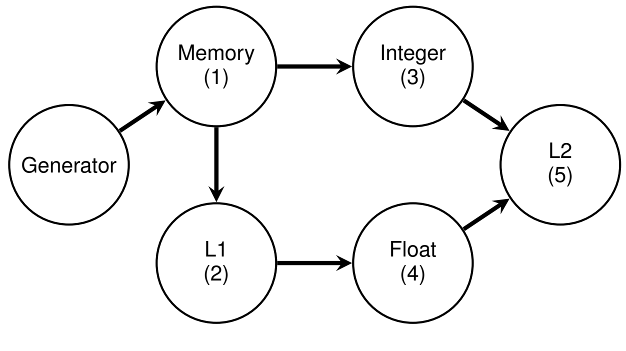

Figure 14. Process Network with 6 processes used in experiment 2.

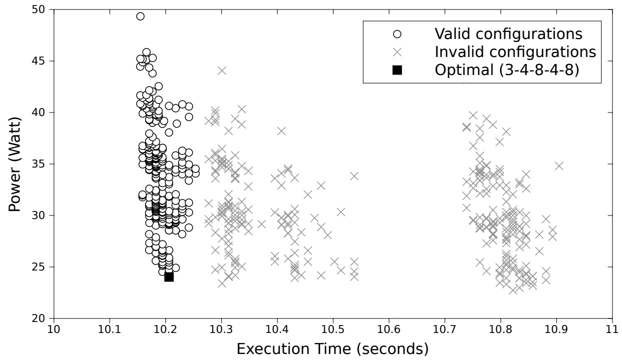

Figure 15. Results for the second Process Network experiment at a permutation of the frequencies 800, 533, 400 and 200 MHz

higher execution times that are not shown within the scale of this graph. We register the number of times a full target FIFO is encountered and when this is larger than zero it means that this configuration does not satisfy the throughput requirement and should be considered invalid.

VII. Conclusion

In this paper we have shown how we can configure and use the available infrastructure to measure power on the SCC with the most accurate results. As the SCC is an existing physical system we had to deal with some architecture specific properties and behavior of which we have shown details and solutions. We proposed a novel method to use the SCC as a substrate to emulate AMPSoCs that can be used for both validation and calibration of design-space exploration methods for power and energy efficiency. Initial experiments show that we are able to measure power changes between different workloads and configurations. The emulation system we have created for running process networks on the proposed system shows that we are able to explore the trade-off between energy or power on one hand and time or throughput on the other for different asymmetric systems and (software) mappings on real hardware.

A. Future Work

The future goals for this work are twofold. First we want to modify our tools such that we can map multiple processes on the same core and are able to run real-world streaming applications with actual data communication, such that we can validate energy models and run-time schedulers and provide our DSE toolkit with real-world energy usage data instead of rough estimations. The second goal is to explore and obtain other experimental platforms or prototype boards in which we can do fine-grained power measurements and have better control over hardware features and configuration such as the aforementioned ARM big.LITTLE, or the recent Intel Sandy Bridge processors.

Acknowledgments

We thank Clemens Grelck for his feedback on early drafts of this paper, as well as the people at Intel Labs that provided us with an Intel SCC system through their MARC program.

References

[1] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, (Washington, DC, USA), pp. 81–92, IEEE Computer Society, 2003.

[2] P. Greenhalgh, "big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7." http://www.arm.com/files/downloads/big.LITTLE_Final.pdf, September 2011.

[3] NVIDIA Corp., "Variable SMP - a multi-core CPU architecture for low power and high performance." http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911b.pdf, 2011.

[4] W. Quan and A. D. Pimentel, "A scenario-based run-time task mapping algorithm for mpsocs," in *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, (New York, NY, USA), pp. 131:1–131:6, ACM, 2013.

[5] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, (New York, NY, USA), pp. 469–480, ACM, 2009.

[6] R. Piscitelli and A. D. Pimentel, "A signature-based power model for mpsoc on fpga," *VLSI Des.*, vol. 2012, pp. 6:6–6:6, Jan. 2012.

[7] N. Chitlur, G. Srinivasa, S. Hahn, P. Gupta, D. Reddy, D. Koufaty, P. Brett, A. Prabhakaran, L. Zhao, N. Ijih, S. Subhaschandra, S. Grover, X. Jiang, and R. Iyer, "QuickIA: Exploring heterogeneous architectures on real prototypes," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pp. 1–8, 2012.

[8] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, (New York, NY, USA), pp. 174:1–174:9, ACM, 2013.

[9] J. Cong and B. Yuan, "Energy-efficient scheduling on heterogeneous multi-core architectures," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, ISLPED '12, (New York, NY, USA), pp. 345–350, ACM, 2012.

[10] V. Gupta, R. Knauerhase, P. Brett, and K. Schwan, "Kinship: efficient resource management for performance and functionally asymmetric platforms," in *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, (New York, NY, USA), pp. 16:1–16:10, ACM, 2013.

[11] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The impact of performance asymmetry in emerging multicore architectures," in *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, (Washington, DC, USA), pp. 506–517, IEEE Computer Society, 2005.

[12] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn, "Efficient operating system scheduling for performance-asymmetric multi-core architectures," in *Proc. of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, (New York, NY, USA), pp. 53:1–53:11, ACM, 2007.

[13] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multi-core architectures," in *Proc. of the 5th European conference on Computer systems*, EuroSys '10, (New York, NY, USA), pp. 125–138, ACM, 2010.

[14] C. Roth, S. Reder, G. Erdogan, O. Sander, G. M. Almeida, H. Bucher, and J. Becker, "Asynchronous parallel MPSoC simulation on the single-chip cloud computer," in *System on Chip (SoC), 2012 International Symposium on*, pp. 1–8, 2012.

[15] J. Howard et al., "A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Solid-State Circuits Conf Digest of Technical Papers (ISSCC), 2010 IEEE Int.*, pp. 108 –109, feb. 2010.

[16] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar, "A 2 Tb/s 6 $\times$ 4 mesh network for a single-chip cloud computer with DVFS in 45 nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 46, pp. 757 –766, april 2011.

[17] M. W. van Tol, R. Bakker, M. Verstraaten, C. Grelck, and C. R. Jesshope, "Efficient memory copy operations on the 48-core Intel SCC processor," in *3rd Many-core Applications Research Community (MARC) Symposium* (D. Göhringer, M. Hübner, and J. Becker, eds.), KIT Scientific Publishing, September 2011.

[18] P. Gschwandtner, T. Fahringer, and R. Prodan, "Performance analysis and benchmarking of the intel scc," in *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, (Washington, DC, USA), pp. 139–149, IEEE Computer Society, 2011.