Comparing the Power and Performance of Intel's SCC to State-of-the-Art CPUs and GPUs

Ehsan Totoni, Babak Behzad, Swapnil Ghike, Josep Torrellas
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
E-mail: {totoni2, bbehza2, ghike2, torrella}@illinois.edu

Abstract—**Power dissipation and energy consumption are becoming increasingly important architectural design constraints in different types of computers, from embedded systems to large-scale supercomputers. To continue the scaling of performance, it is essential that we build parallel processor chips that make the best use of exponentially increasing numbers of transistors within the power and energy budgets. Intel SCC is an appealing option for future many-core architectures. In this paper, we use various scalable applications to quantitatively compare and analyze the performance, power consumption and energy efficiency of different cutting-edge platforms that differ in architectural build. These platforms include the Intel Single-Chip Cloud Computer (SCC) many-core, the Intel Core i7 general-purpose multi-core, the Intel Atom low-power processor, and the Nvidia ION2 GPGPU. Our results show that the GPGPU has outstanding results in performance, power consumption and energy efficiency for many applications, but it requires significant programming effort and is not general enough to show the same level of efficiency for all the applications. The "light-weight" many-core presents an opportunity for better performance per watt over the "heavy-weight" multi-core, although the multi-core is still very effective for some sophisticated applications. In addition, the low-power processor is not necessarily energy-efficient, since the runtime delay effect can be greater than the power savings.**

I. Introduction

Following Moore's law, the number of transistors that can be placed on a chip keeps increasing rapidly with each technology generation. Not surprisingly, users expect the performance to also improve over time with the increase in the number of transistors. However, performance depends on multiple factors beyond the transistor count.

The architecture community has historically tried to turn the higher transistor count into higher performance. For example, during the single-thread era, excess transistors were used for architectural features such as pipelining, branch prediction, or out-of-order execution. These features were able to improve performance while largely keeping the application unchanged.

However, in the last several years, some key technology parameters such as the supply voltage have stopped scaling. This has led to chips with unsustainably-high power, power density and energy consumption. This fact combined with the diminishing returns from single-thread architectural improvements, has pushed forward thread parallelism as the only solution to make effective use of the large number of transistors available. The result has been a paradigm shift toward parallelism.

A key architectural challenge now is how to support increasing parallelism and scale performance, while being power and energy efficient. There are multiple options on the table, namely "heavy-weight" multi-cores (such as general purpose processors), "light-weight" many-cores (such as Intel's Single-Chip Cloud Computer (SCC) [1]), low-power processors (such as embedded processors), and SIMD-like highly-parallel architectures (such as General-Purpose Graphics Processing Units (GPGPUs)).

The Intel SCC [1] is a research chip made by Intel Labs to explore future many-core architectures. It has 48 Pentium (P54C) cores in 24 tiles of two cores each. The tiles are connected by a four by six mesh in the chip. The SCC naturally supports the message passing programming model, as it is not cache-coherent in hardware. We have ported CHARM++[2] and Adaptive MPI (AMPI) [2] to this platform to be able to run existing sophisticated applications without any change.

The goal of this paper is to explore various trade-offs between the SCC and the other types of processors. We use five applications to study their power and performance: NAMD, Jacobi, NQueens, Sort and CG (conjugate gradient). These applications exhibit different characteristics in terms of both computation and communication. The processors used are the Intel SCC as a light-weight many-core, the Intel Core i7 as a heavy-weight multi-core, the Intel Atom as a low-power processor, and the Nvidia ION2 as a GPGPU. These processors represent different cutting-edge architectures. To compare these architectures, the applications are executed with the same input parameters and we measure speed, power, and energy consumption.

Our results show that each of the designs is effective in some metric or condition and there is no single best solution. For example, the GPGPU provides a significant advantage in terms of power, speed and energy in many cases, but its architecture is not general enough to fit all the applications efficiently. In addition, the GPGPU requires significant programming effort to achieve this efficiency (as we had to use different codes to run on the GPGPU) and cannot run legacy codes.

The Intel SCC results suggest that light-weight many-cores are an opportunity for the future. The SCC has lower power than the heavy-weight multi-core and runs faster than the low-power design. Also, the light-weight many-core is general enough to run legacy code and is easy to program (in contrast to the GPGPU). However, some weaknesses of the platform should be addressed in future designs to make it competitive with sophisticated multi-cores. One such weakness that we identified is slow floating-point performance.

This paper also proves that the low-power processor does not necessarily result in less energy consumption. As shown by our data on the Intel Atom platform, the extra delay has a greater effect than the power savings achieved.

The rest of this paper is organized as follows. §II describes the architecture of the platforms that we study. §III briefly introduces the applications, as their characteristics are very important to understand their scaling and power consumption on different platforms. §IV evaluates the platforms using the applications. We compare the architectures in §V using the results of the previous section and analyze the tradeoffs of each one. We discuss the related work in §VI and conclude in §VII.

II. Platforms

Here we describe the platforms that we evaluate, with a focus on their design concept and level of parallelism. Among these platforms, the SCC is a research chip while the others are examples of commodity platforms, which are being widely used in different machines.

A. *Intel Single-chip Cloud Computer*

The "Single-Chip Cloud Computer" (SCC) is Intel's new research many-core architecture. It has 48 Pentium cores connected through a mesh interconnect. It has been created by Intel Labs to facilitate software research on future many-core platforms. This chip is not cache coherent and it naturally supports the message passing parallel programming paradigm.

Figure 1 shows the architecture overview of the SCC [1]. The cores are arranged in groups of two in 24 tiles. The tiles are connected in a four by six mesh configuration. The cores are simple second-generation off-the-shelf Pentium cores (P54C). Each core has 16KB L1 data and 16KB L1 instruction caches as well as a 256KB unified L2 cache. Each tile has a 16KB SRAM called Message Passing Buffer (MPB), which is used for communication inside the chip. These MPBs form a shared address space used for data exchange. The cores and MPB of a tile are connected to a router by Mesh Interface (I/F) unit. The SCC also has four DDR3 memory controllers in the four corners of the mesh network to connect cores to memory.

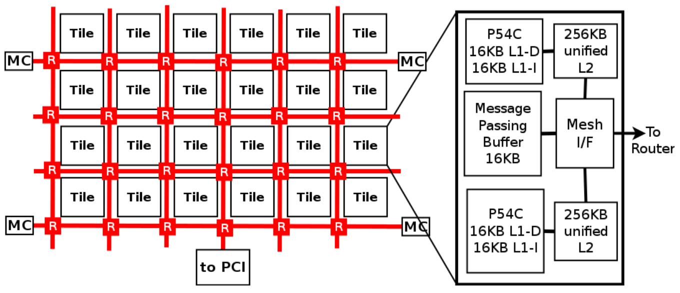


Fig. 1. Architecture overview of the SCC.

The SCC is implemented in 45nm CMOS technology and has 1.3 billion transistors. The area of each tile is 18 mm^2 with a total die area of 567 mm^2. Power for the full chip ranges from 25W to 125W. It consumes 25W at 0.7V, with 125MHz cores, 250MHz mesh, and 50°C. It consumes 125W at 1.14V, with 1GHz cores, 2GHz mesh, and 50°C. Power for the on-die network is 6W for a 1.5 Tb/s bisection bandwidth and 12 W for a 2 Tb/s bisection bandwidth. Figure 2 shows the power breakdown of the chip in two different modes: full power and low power [3].

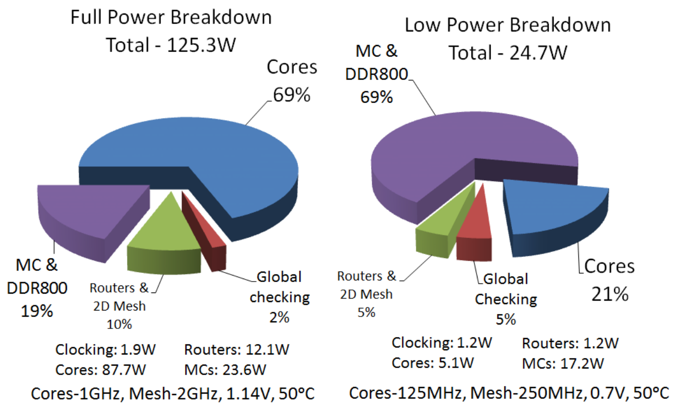


Fig. 2. Power breakdown of the SCC in full-power and low-power mode (from J. Howard et al. [3]).

The SCC was designed with power management in mind. It includes instructions that let programmers control voltage and frequency. There are 8 voltage domains on a chip: one for the memory controllers, one for the mesh, and six to control voltage for the tiles (at the granularity of 4-tile blocks). Frequency is controllable at the granularity of an individual tile, with a separate setting for the mesh, thereby providing a total of 25 distinct frequency domains. The RCCE library [4] provides easy access to these features, but it is more limited than the available hardware features [3].

As mentioned, message passing is the natural way of programming this non-cache-coherent chip. For this purpose, there is a low level message passing interface called RCCE that provides low level access to the communication features of the SCC. It was designed so that the chip can still operate without any operating system ("bare metal mode") to reduce the overheads [4]. However, Linux can also be run on the SCC, which is the most common usage, and we ran our experiments in this mode. In addition, there is another interface called Rckmb, which provides the data link layer for running network services such as TCP/IP. We used the latter to port Charm++ and run existing applications. Using other layers to port Charm++ could result in some communication performance improvement; however, it would not change our conclusions.

Porting Charm++ and the applications did not involve any conceptual difficulty as the SCC can be easily viewed as a cluster on a chip. However, there are many technical issues involved in working with it. These include dealing with old compilers, an old operating system, and unavailability of some standard software and libraries. Unfortunately, we could not perform many of the intended experiments because of these technical issues and others took much more time than expected.

B. Other Platforms

Intel Core i7 Processor: The Intel Core i7 is a 64-bit x86-64 processor. We have used the Core i7 860 Nehalem processor chip, which has four CPU cores and on-chip cache memory on one 45nm die. Hyperthreading support allows it to appear to the OS as eight processing elements. The cores cycle at 2.8GHz (disregarding Turbo Mode), which is a relatively high frequency. Each of the four cores has 32KB instruction and 32KB data Level 1 caches, and 256KB of Level 2 cache. The four cores share an inclusive 8MB Level 3 cache. The specification of the Intel Core i7 is shown in Table I.

TABLE I
INTEL CORE I7 PROCESSOR SPECIFICATIONS

Processor Number	i7-860
# of Cores	4
# of Threads	8
Clock Speed	2.8 GHz
Cache Size	8 MB
Lithography	45 nm
Max TDP	95W
VID Voltage Range	0.65V-1.40V
Processing Die Size	296 mm^2
# of Processing Transistors on Die	774 million

Intel Atom D525: The Intel Atom is the ultra-low-voltage x86-64 CPU series from Intel. It is designed in 45 nm CMOS and used mainly in low power and mobile devices. Hyperthreading is also supported in this processor. However, there is no instruction reordering, speculative execution or register renaming.

Due to its modest 1.8 GHz clock speed, even the fastest Atom D525 is still much slower than any desktop processor. The main reason behind our selection of the Atom for our experiments is that, while desktop chips have a higher frequency, the Atom is hard to beat when it comes to power consumption. Atom allows manufacturers to create low-power systems. However, low power does not always translate into high efficiency, meaning that Atom may have low performance per watt consumed. We have explored this issue in our experiments.

The specification of the Intel Atom processor that we used is shown in Table II.

TABLE II
INTEL ATOM D525 PROCESSOR SPECIFICATIONS

Processor Number	D525
# of Cores	2
# of Threads	4
Clock Speed	1.80 GHz
Cache Size	512 KB
Lithography	45 nm
Max TDP	13W
VID Voltage Range	0.800V-1.175V
Processing Die Size	87 mm^2
# of Processing Transistors on Die	176 million

Nvidia ION2 Platform: Nvidia ION is a system/motherboard platform that includes Nvidia's GPU, DDR3 or DDR2 SDRAM, and the Intel Atom processor. The Nvidia ION2 has a dedicated graphics card for the new Atom CPUs. The ION2 is based on the GT218 chip (GeForce 305M, 310M) with dedicated memory (compared to the old ION that was a chipset graphics card). ION2 systems can use CUDA (Nvidia's General-Purpose Computing on Graphics Processing Units technology) as well as OpenCL (Open Computing Language), to exploit the parallelism offered by the CPU and the GPU together. This platform is used in low-power devices, and yet is equipped with a GPU. Hence it has the potential to offer great benefits in performance and power at the same time. We have used a 12" ION2 Pinetrail netbook platform for our experiments.

The specification of the CPU was mentioned in Table II. The specification of the graphics processor is shown in Table III.

TABLE III
NVIDIA ION2 GRAPHICS CARD SPECIFICATIONS

ION Series	ION2
GPU Number	GT218
# of CUDA Cores	16
Clock Speed	475 MHz
Memory	256 MB
Memory bus width	64-bit
Power consumption	12W

III. APPLICATIONS

The characteristics of the applications are important to understand their different behavior and derive conclusions about the architectures. In this section, we describe the applications we used to examine the different parallel architectures. We choose scalable parallel applications that use CHARM++ [2] or MPI message passing paradigms. For the GPU platform, we use appropriate versions of the applications based on the OpenCL or CUDA models. The benchmarks are reasonably optimized but not highly optimized for any particular architecture. These applications represent different classes of programs with different characteristics to stress the platforms.

Iterative Jacobi: The Jacobi calculation is a useful benchmark that is widely used to evaluate many platforms and programming strategies. A data set (2D array of values in our case) is divided among processors and is updated in an iterative process until a condition is met. The communication is mostly a nearest neighbor exchange of values. An OpenCL implementation was used for ION2, while a CHARM++ version was used for the other platforms.

We selected Jacobi in our experiments since it is representative of stencil computations, which are widely used in scientific programs.

NAMD: NAMD is a highly scalable application for Molecular Dynamics simulations [5]. It uses hybrid spatial and force decomposition to simulate large molecular systems with high performance. It is written in CHARM++ to exploit its benefits such as portability, adaptivity and dynamic load balancing. It typically has a local neighbors communication pattern without bulk synchronization and benefits from communication and computation overlap. It also tries to keep its memory footprint small in order to utilize caches and achieve better memory performance. We chose this application to represent dynamic and complicated scientific applications. We used the ApoA1 system as input, which has 92,224 atoms.

NQueens: The NQueens puzzle is the problem of placing n chess queens on an $n \times n$ chessboard so that no two queens attack each other. The program will find the number of unique solutions in which such valid placement of queens is possible. This problem is a classic example of the state space search problems. Since NQueens is an all-solutions problem, the entire solution tree needs to be explored. Thus, this problem also presents a great opportunity for parallelization with minimum communication.

We selected this problem since it is an integer program, as opposed to the other floating-point problems, and it represents state space search applications.

CG: Conjugate Gradient is one of the NAS Parallel Benchmarks (NPB) [6], which are widely used to evaluate the performance of different parallel systems. CG is an iterative method and involves lots of communication and data movement. We chose it to stress the communication capabilities of our platforms. Unfortunately, we did not have a version of it available on our GPU system.

Integer Sort: Parallel sorting is a widely used kernel to benchmark parallel systems because it represents commercial workloads with few computation and heavy communication. It does not have the massive floating point computations and regular communication patterns (such as nearest neighbors) of many scientific workloads. We use a straightforward implementation of Radix Sort in OpenCL and a similar CHARM++ version.

Table IV shows some performance counter values for the applications. They are obtained using the PAPI library running on the Core i7 system. The numbers are normalized with respect to the number of dynamic instructions executed. Since the applications are already well-known, these few numbers give enough insight to help explain the results.

TABLE IV
PERFORMANCE COUNTERS FOR THE APPLICATIONS

	Jacobi	NAMD	NQueens	CG	Sort
L1 Data Misses	0.0053	0.0053	0.0003	0.0698	0.0066
Cond. Branches	0.0205	0.0899	0.1073	0.1208	0.0556
Float-pt operations	0.0430	0.3375	0.0004	0.2078	0.0001

IV. EVALUATION RESULTS

We now run the applications on the proposed platforms and measure the scaling behavior and the power and energy consumption. The data provides insight into the effectiveness of different architectural approaches. We focus on the ability of the applications to exploit the parallelism of the platform and how using more parallelism will affect power and energy consumption. We connected power meters to the whole systems (rather than just the processor chips) to measure the power consumption.

A. Intel SCC

We have ported CHARM++ to the SCC using the network layer (TCP/IP) provided by Rckmb. Thus, CHARM++ programs can run on the SCC hardware without any change to the source code. This simplifies the porting of software significantly. We used the highest performance options available in the SCC software toolkit (1.3) to run the experiments and characterize the system. Table V shows the SCC configuration used in the experiments.

TABLE V
SCC CONFIGURATION USED IN THE EXPERIMENTS

Operating mode	Linux
Communication mechanism	TCP/IP
Tile frequency	800 MHz
Mesh frequency	1600 MHz
Memory controller frequency	1066 MHz
Supply Voltage	1.1 V
Idle power	39.25 W

Figure 3 shows the speedup of the five applications on different numbers of cores. It can be observed that using more cores improves performance, and that all the applications except CG are scalable on this platform. In addition, these CHARM++ applications are scalable without any change to the source code. In Jacobi, the execution time per step on one core is $7.87s$, and on the full machine is $0.32s$. This corresponds to a 24.6 speedup on 48 cores. NQueens takes $568.36s$ to execute on one core and $19.66s$ on 48 cores. The resulting speedup is 28, which is even higher than Jacobi. In NAMD, the time per step is $34.96s$ on one core and $1.23s$ on all the cores. This also corresponds to a speedup of 28 on 48 cores. Thus, NAMD is an example of a full-fledged application that can use many-core architectures effectively.

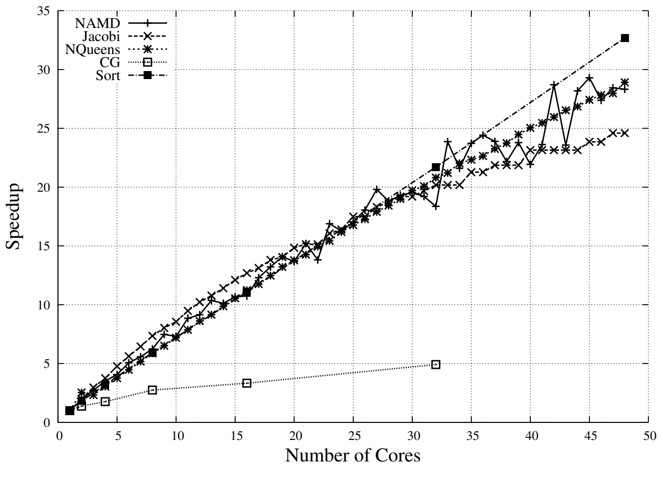


Fig. 3. Speedup of the applications on different numbers of SCC cores.

Sort is the most scalable application, with a 32.7 speedup. This shows that the network can handle the communication effectively.

On the other hand, CG is the worst-scaling application, with a speedup of just 4.91 on 32 cores. We could not run it on more cores because this application requires the number of cores to be a power of two. The reason why CG is not scalable is the fine-grain global communication present in the algorithm. For example, there is a global reduction after each phase. Specifically, the performance counter data of Table IV shows that CG has a high number of L1 cache misses on Core i7. Since the problem size is small and fits in the cache, the misses are caused by high communication. Thus, the SCC communication system (network and software) may not be suitable for this application. Optimizing the network for global communication such as by adding a collectives network can help significantly in this case. Also, tuning the runtime system such as by tuning the collective algorithms may help.

As indicated above, CG requires a power-of-two number of cores. This is an important consideration when designing many-cores, since some programmers may assume the number of cores to be a power of two for simplicity. A promising solution is to use virtualization, and use any number of virtual processors that the application needs on the available physical processors. This feature is available in CHARM++ but we do not evaluate it here.

Finally, we ran four of the applications on every possible number of cores – from 1 to 48. The speedups are consistent in all the applications. In NAMD, there is some noise due to the application's dynamic and adaptive behavior.

Figure 4 shows the power consumption of the platform using different numbers of cores. These values are the maximum values seen during the run time of each individual application. For Jacobi, the power goes from 39.58W using just one core to 73.18W using all the 48 cores. NAMD's power consumption is similar to Jacobi. NQueens consumes more power, and goes up to 85.5W.

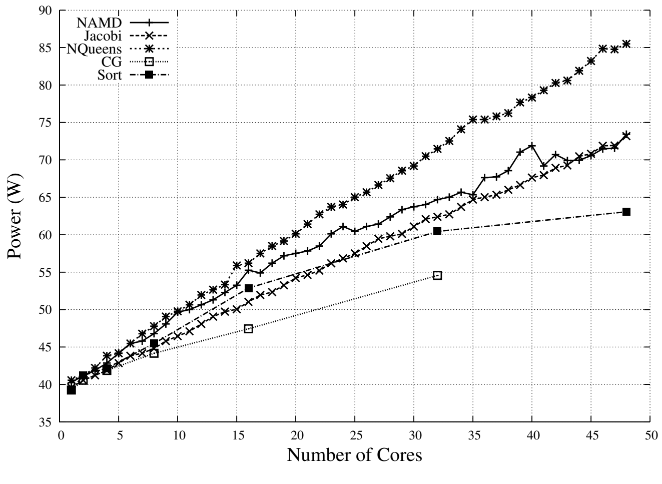


Fig. 4. Power consumption of the applications on different numbers of SCC cores.

CG and Parallel Sort consume less power compared to the other applications. This is mainly because these applications are communication-bound and processors often stall, waiting for the communications to be completed. CG consumes the least power because it has the most stall time.

Figure 5 shows the energy consumed by each application, which is the power multiplied by time of execution. The energy of each application with a given number of cores is normalized to the application's energy using all the cores. This allows us to compare the applications. The general trend shows that using more cores to run the applications results in less energy consumption on the SCC. This is because the performance improvement attained by the added cores is higher than the power increases. Note that on the left side of Figure 5 there is a large reduction in energy as we use more cores. This is because the idle power of the system is high compared to the power added by adding one core, while the execution time decreases notably. For example, when going from one to two cores, the execution time drops to nearly half, while the power difference is small. Therefore, the energy consumption drops to nearly half.

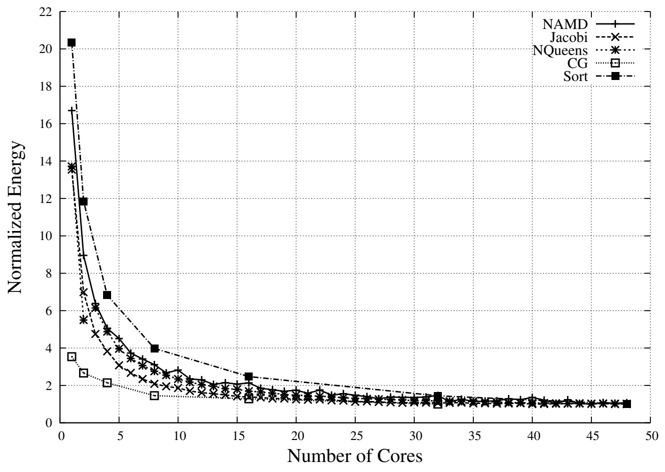


Fig. 5. Energy consumption of the applications on different numbers of SCC cores. The data is normalized to the energy consumed with all the cores.

The highest energy drop is for Sort and the lowest is for CG. This is because Sort is scalable, and the time savings is more significant than the power added by using more cores. Conversely, the inferior scalability of CG results in a small drop in energy consumption. Again, one should keep in mind the idle power when analyzing power and energy consumption because it offsets the power increase with more cores, and the time savings become more important.

B. Intel Core i7 Processor

We have used the CHARM++ infrastructure for the Intel x86_64 platform. In addition, we have used the same Jacobi, NAMD, NQueens, and Sort programs written in CHARM++ and the same CG written in MPI as the ones we ran on the SCC. Since the Intel Core i7 is a quad core processor with hyperthreading, we can run up to 8 threads.

Figure 6 shows the speedup of these applications on different numbers of threads. By increasing the number of threads, we initially observe good speedups. Note the reduction in speedup when 5 threads are used as opposed to 4 threads. This is probably because at least two threads have to share the resources of one core. Therefore, they become slower and slow down the whole application. However, with increased parallelism, the application becomes faster and the slowdown is compensated.

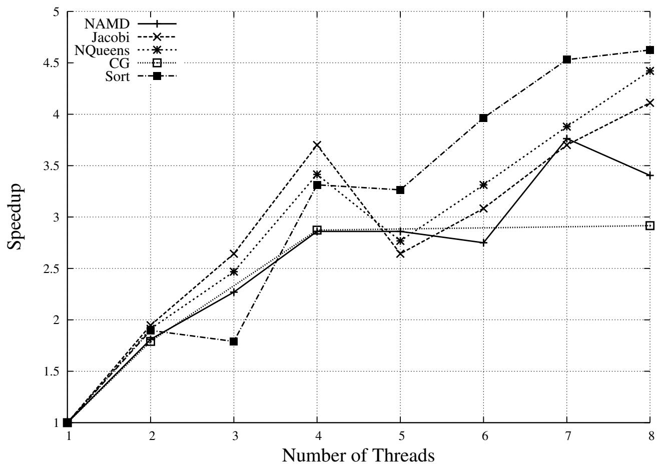


Fig. 6. Speedup of the applications on different numbers of threads in the Intel Core i7.

Figure 7 shows the power consumption of the Core i7 platform using different numbers of threads. As expected, the power used by this processor increases as the number of threads increases. However, the increase is much higher than in other platforms. The power range of the Core i7 system is from around 51W for the idle power up to 150W, which is much higher than the SCC power.

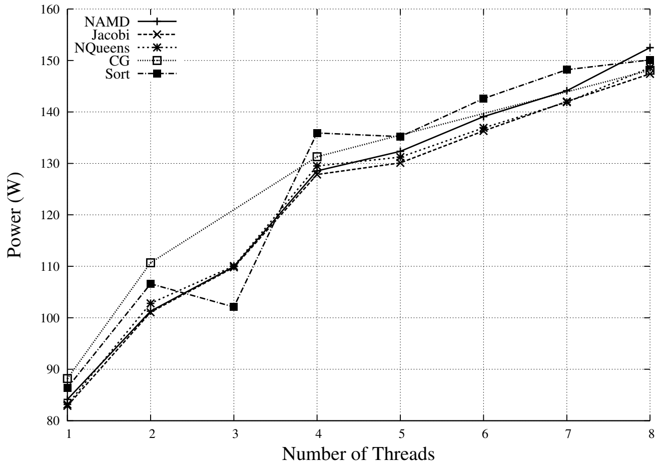


Fig. 7. Power consumption of the applications on different numbers of threads in the Intel Core i7.

Figure 8 presents the energy consumed by each application, which is the power multiplied by the execution time. As in the case of the SCC, the energy is normalized to the energy consumed when running all the threads. The general trend is that with more cores, the energy consumption goes down. Again, as we saw in the case of speedup, when using 5 threads, we had some increase in the runtime and, therefore, in energy consumption. Note that the reduction in energy is not as large as in the SCC case.

C. Intel Atom D525

The Atom D525 is a dual core processor in which each core is 2-way hyperthreaded. All the 4 threads can execute

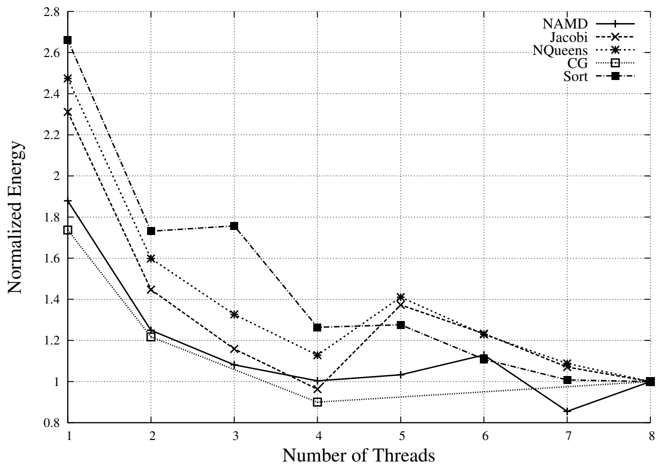


Fig. 8. Energy consumption of the applications on different numbers of threads in the Intel Core i7. The data is normalized to the energy consumed with all the threads.

independently. Hence we can specify up to 4 processors to CHARM++. In Figure 9, we observe good speedups with the increase in the number of threads for several programs.

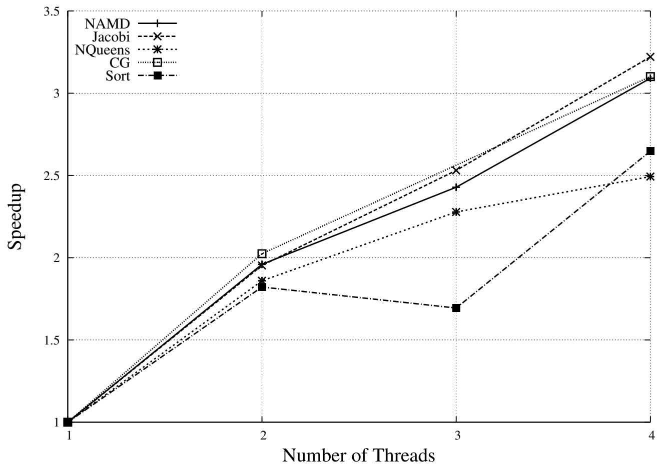


Fig. 9. Speedup of the applications on different numbers of threads in the Atom processor.

Figure 10 shows that the Atom system consumes much less power than the other platforms. The increase in power per thread added is less than 1 W. Since the idle power of the entire system is about 27.5 W, the power increase due to having all four threads active is about 15% of the idle power.

In Figure 11, we observe that using more threads leads to less energy consumption, like in the other platforms. The power used by the Atom increases with an increase in the number of threads. However, the increase is not as rapid as in the Core i7 because of Atom's simpler architecture. At the same time, execution time reduces considerably with more threads, and hence the energy consumption decreases.

D. Nvidia ION2 Platform

We leverage the parallel computing power of the Nvidia ION2 platform using OpenCL for Jacobi, NQueens and Sort,

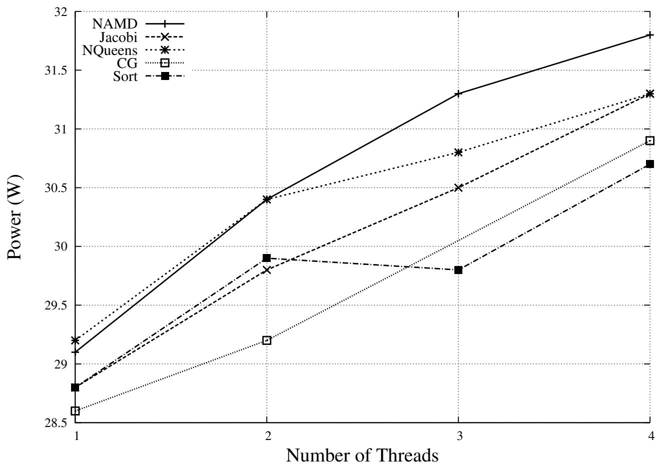


Fig. 10. Power consumption of the applications on different numbers of threads in the Atom processor.

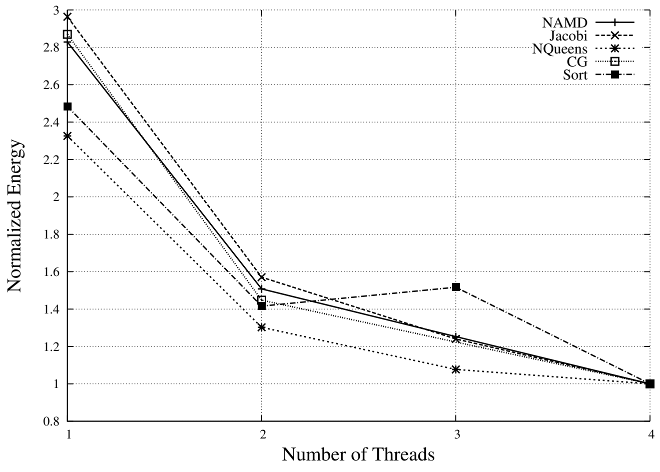


Fig. 11. Energy consumption of the applications on different numbers of threads in the Atom. The data is normalized to the energy consumed with all the threads.

and CUDA for NAMD. Unfortunately, we could not get a reasonably tuned version of CG. GPUs are very effective at data parallel applications due to the high number of simplistic SIMD compute units available in them. However, because the GPU is a coprocessor on a separate PCI-Express card, data must first be explicitly copied from the system memory to the memory on the GPU board. In general, applications which require a large amount of computation per data element and/or make full use of the wide memory interface are well suited to the GPU programming model.

Figure 12 shows data on the speed, power, and energy of the applications running on the ION2 platform. We use all 16 CUDA cores in the GPU, and do not change the parallelism of the applications because there would not be a significant difference in power. The speed bars show the speedup of the applications running on the ION2 platform relative to running on the Atom platform with the maximum number of threads. We see that the speedup is 22.65 for Jacobi and 16.68 for NQueens. While these are high speedups, given the number of compute cores available in the GPU, we would have expected higher speedups. The copying of device memory buffers at the end of each iteration forms the bottleneck in this computation. Note also that NAMD is no faster on ION2 than on Atom. NAMD has had scalable results on GPGPUs and we believe that, with careful tuning, it is possible to achieve better results. However, the effort was estimated to be high. Finally, Sort is 1.76 times slower than all the threads on the Atom. Thus, Sort is not suitable to run on GPGPUs.

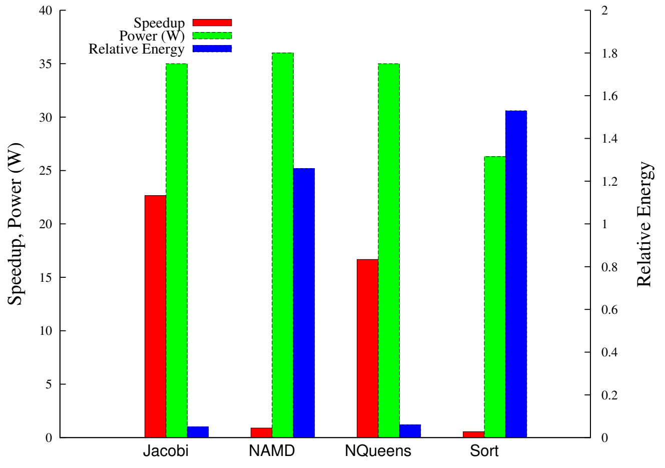


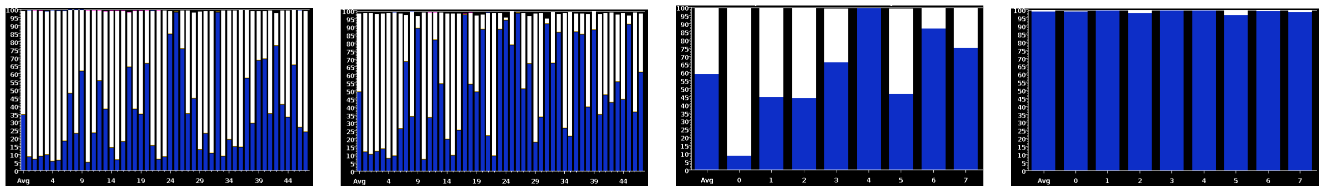
Fig. 12. Speed, power, and energy of the applicatons running on the ION2 platform.

The power bars show the absolute power consumption of the ION2 platform in W using all the 16 CUDA cores. We see that the power consumption ranges from 25 W to 35 W. These numbers are much lower than those of the Core i7. Note that we have not performed our experiments on more powerful GPUs, which are expected to provide better performance, albeit at the cost of some more power consumption. In any case, we do not expect the power consumption to rise as high as a heavy-weight processor like the Core i7.

The final bars show, for each application, the energy consumed by the ION2 normalized to the energy consumed by the Atom. We normalize the energies to the Atom numbers to be able to compare the energy of the different applications — otherwise, long-running applications would dwarf short-running ones. For these bars, we use the Y axis on the right side. Overall, from the figure, we see the ION2 is more energy-efficient than the Atom for Jacobi and NQueens; the opposite is true for NAMD and Sort.

E. Load Balancing

As the number of cores per chip increases, load balancing becomes more important (and challenging) for efficient use of the available processing power. Here, we investigate the effectiveness of dynamic load balancing on the SCC (with 48 threads) compared to the Core i7 (with 8 threads). We use *LBTest*, which is a benchmark in the CHARM++ distribution, with *RefineLB* as the balancer. LBTest creates a 3D mesh graph where the nodes have objects that perform random computation. Each object also sends a message (of a size that can be specified) to one of its neighbors randomly. In



(a) SCC Unbalanced (b) SCC Balanced (c) Core i7 Unbalanced (d) Core i7 Balanced

Fig. 13. Utilization of the threads before and after balancing the load on the SCC and Core i7 platforms.

our case, the size of the messages is very small compared to the computational load, so communication does not matter much. RefineLB is a simple load-balancing strategy that tries to balance the load by gradually removing objects from overloaded threads.

Figure 13 shows the utilization of each of the threads in the SCC and Core i7 platforms before and after applying the load balancer. The figure is obtained using Projections [7], which is a performance visualization tool in the CHARM++ infrastructure. From the figure, we can see that the load balancer perfectly balances the load on the Core i7 platform and somewhat improves the balance on the SCC platform. On average, it increases the average thread utilization from 35% to 50% in the SCC, and from 59% to 99% in the Core i7. Load balancing is more difficult in a platform like SCC, which has many cores. The load balancer has increased the average utilization of the SCC cores significantly, but at 50% average utilization, it is clear that more effective methods are needed in the future. Overall, it can be shown that the load balancer improves the performance of the benchmark (with the same input parameters) by 30% in the SCC and by 45% on the Core i7.

V. Comparison of Different Architectures

In this section, we use the data of previous sections to analyze and compare the architectures. For each of the three metrics (speedup, power and energy consumption), we run the applications on all the parallel threads of each platform, to use all the resources available. On all the platforms except the ION2, we use the same source codes, written either in CHARM++ (Jacobi, NAMD, NQueens, and Sort), or in MPI (CG). For the ION2, we use OpenCL for Jacobi, NQueens and Sort, and CUDA for NAMD. Unfortunately, we could not get a reasonably tuned version of CG for the ION2.

When comparing the SCC to the other platforms, one should keep in mind that the SCC is a research chip, whereas the other platforms are highly-optimized production machines. With this in mind, one main purpose of the comparison is to find the weaknesses of the SCC and propose improvements for the future.

Figure 14 shows the speed of the five applications on the different platforms relative to Atom. As can be seen, the ION2 shows significantly better performance than the other platforms for Jacobi and NQueens, but not for NAMD or Sort. Jacobi and NQueens are simple highly-parallel applications with regular memory access and communication patterns. They match this "SIMD-like" ION2 hardware nicely. In contrast, Sort has irregular memory accesses and communication patterns, which make it unsuitable for the ION2. Finally, we could not obtain good speedups for NAMD on ION2 with our minimal porting and tuning effort, even though NAMD has been shown to scale well on GPGPUs elsewhere. Porting applications to GPGPUs is one of the most important issues in these highly-parallel architectures. There are millions of lines of existing legacy parallel code, which cannot exploit GPGPUs easily (for example, scientific communities have a lot of parallel code mostly written in MPI). In addition, the effort for tuning and writing new code is high for GPGPUs. Generating highly-optimized codes on GPGPUs is not easy for an average programmer, and is not the subject of this work.

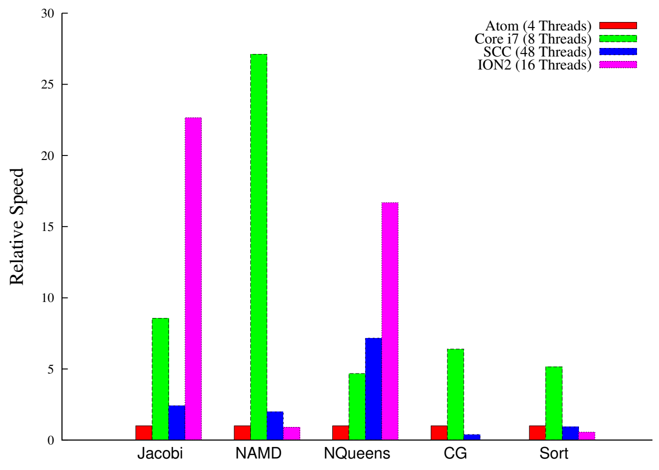


Fig. 14. Speed of the applications on the different platforms relative to Atom

Overall, GPGPUs (and other architectures that are similar to SIMDs in general) are attractive for applications with simple control flow and high parallelism. However, they fail to provide good performance in other classes of applications.

In Figure 14, the Intel Core i7 is much faster than the other platforms for NAMD, CG and Sort. This shows that heavy-weight multi-cores are attractive solutions for dynamic applications with complex execution flow such as NAMD. The higher performance is due to high floating-point performance, support for short-length vector instructions such as SSEs, support for complex control flow (through aggressive branch prediction and speculation), and support for a high degree of instruction-level parallelism (attained by out-of-order execution).

In addition, these multi-cores are suitable for applications with irregular accesses and fine-grained communications, such as CG and Sort. These traditional platforms have highly-optimized cache hierarchies, share memory, and need less thread-level parallelism for high performance. Thus, irregular accesses are handled properly by the cache hierarchy, fine-grained communications are less costly because of shared memory, and there is less communication because of less parallelism.

Focusing on the SCC, the figure shows that, in general, the SCC speedups are not that good. The SCC is faster than the Core i7 for NQueens, but slower for the other applications. For such applications, it is comparable to the Atom. The fine-grain communication in CG is especially hard to support well in the SCC.

According to table IV, NQueens is an integer application, with few floating-point operations. On the other hand, Jacobi, which has a similar scaling behavior on the SCC, has many floating point-operations. Hence, low floating-point performance is a weakness of the SCC and enhancing it can improve performance substantially. Also, since §IV showed that all the applications except CG have scalable performance on the SCC, we believe that by improving sequential performance, the SCC can be much faster. The SCC needs more sophisticated cores, which is easily to attain because CMOS scaling will bring more transistors on chip in the near future. In addition, network performance also needs to be improved along with the cores, to keep the system balanced, which is also possible. Thus, an upgraded SCC many-core architecture can become a very attractive alternative for the future.

The Atom is slower than the other processors in most of the cases, which is expected by its low-power design.

Figure 15 shows the power consumption of the applications on the different platforms. In the figure, the Atom and the ION2 platforms consume low power. The Core i7 consumes the most power, because of its higher frequency and its many architectural features such as out-of-order execution. As can be seen in the figure, the power consumption of the SCC is somewhere in between the low-power and the high-power platforms. The SCC decreases the power consumption compared to heavy-weight machines through more parallelism and lower frequency. It is an appealing platform because it moderates the high power consumption of conventional multi-cores, while still being general enough to avoid the usability issues of GPGPUs, and is generally faster than low-power designs like the Atom.

Figure 16 shows the energy consumption of the applications on the different platforms normalized to the energy consumed on the Atom platform. We use this nomalized-energy metric to be able to compare the different applications which, potentially, have a very different execution time. The figure shows that the ION2 platform is very energy-efficient for Jacobi and NQueens. For these regular applications, ION2 consumes low power and executes fast. However, ION2 is not so energy-efficient for the NAMD and Sort applications, because of their long execution time.

The Core i7 platform exhibits good energy efficiency across the board. The Atom platform is less energy-efficient. Although it uses low power, it has a relatively longer execution time and, therefore, the energy consumption is higher than in the Core i7 platform. Thus, low-power design does not necessarily lead to less energy consumption.

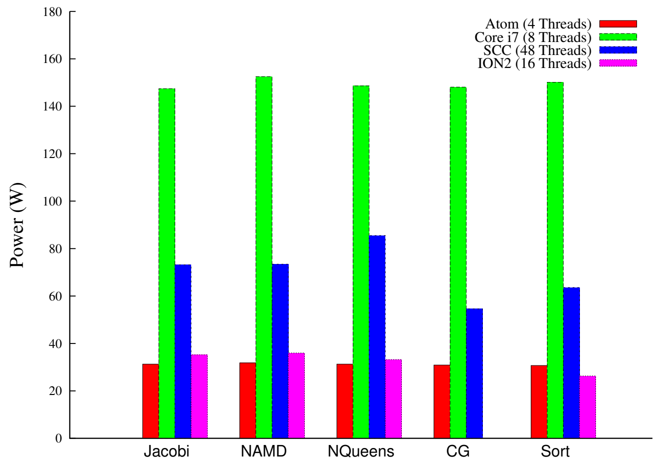


Fig. 15. Power consumption of the applications on the different platforms.

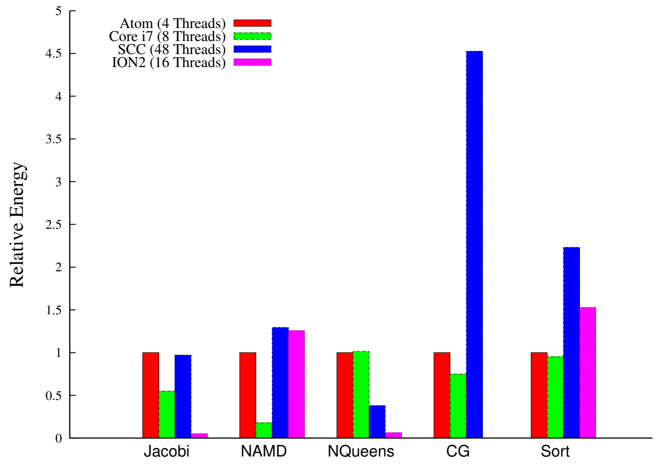


Fig. 16. Energy consumption of the applications on the different platforms normalized to the energy on the Atom platform.

The SCC platform is somewhat less energy-efficient than the Core i7 and Atom platforms for most applications, although it is still competitive. In the case of NQueens, the SCC is very energy-efficient. One exception is CG, where the SCC consumes substantially more energy. This is because CG has little computation and much communication. Overall, it is likely that an upgraded and refined SCC will have good energy efficiency for all of these applications.

VI. Related Work

Marker et al. [8] port a dense matrix computations library to the SCC. They use the RCCE communications library and replace the collective communications. Using a lower-level communications library may have performance advantages, but it causes porting difficulties, especially when the collective is not implemented in the library. Our approach of porting the runtime system made running the applications possible without any change to the source code.

Power management is another topic of research for the SCC [9], because of its extensive DVFS support. Different parts of the SCC design, such as the network [10] or communication libraries [4] are described elsewhere [1], [3].

Different communication libraries, including MPI, have been studied and implemented for the SCC [11], [12], [13]. Porting CHARM++ on top of them is a future study, which may result in performance improvements. There are also many other SCC-related studies going on in the Many-core Applications Research Community of Intel (http://communities.intel.com/community/marc).

Esmaeilzadeh et al. [14] provide an extensive report and analysis of the chip power and performance of five different generations of Intel processors with a vast amount of diverse benchmarks [14]. Such work, however, does not consider many-cores or GPUs, which are promising architectures for the future of parallel computing.

VII. Conclusion

Large increases in the number of transistors, accompanied by power and energy limitations, introduce new challenges for architectural design of new processors. There are several alternatives to consider, such as heavy-weight multi-cores, light-weight many-cores, low-power designs and SIMD-like (GPGPU) architectures. In choosing among them, several possibly conflicting goals must be kept in mind, such as speed, power, energy, programmability and portability. In this work, we evaluated platforms representing the above-mentioned design alternatives using five scalable CHARM++ and MPI applications: Jacobi, NAMD, NQueens, CG and Sort.

The Intel SCC is a research chip using a many-core architecture. Many-cores like the SCC offer an opportunity to build future machines that consume low power and can run CHARM++ and MPI code fast. They represent an intersting and balanced design point, as they consume lower power than heavy-weight multi-cores but are faster than low-power processors and do not have the generality or portability issues of GPGPU architectures. In our analysis of the SCC, we suggested improvements in sequential performance, especially in floating-point operation speed, and suggested adding a global collectives network.

We showed that heavy-weight multicores are still an effective solution for dynamic and complicated applications, as well as for those with irregular accesses and communications. In addition, GPGPUs are exceptionally powerful for many applications in speed, power and energy. However, they lack the sophisticated architecture to execute complex and irregular applications efficiently. They also require a high programming effort to write new code, and are unable to run legacy codes. Finally, as seen from the Intel Atom experiments, we observe that low-power designs do not necessarily result in low energy consumption, since they may increase the execution time significantly. Therefore, there is no single best solution to fit all the applications and goals.

Acknowledgments

We thank Intel for giving us an SCC system and the anonymous reviewers for their comments. We also thank Laxmikant Kale, Marc Snir and David Padua for their comments and support. This work was supported in part by Intel under the Illinois-Intel Parallelism Center (I2PC).

References

[1] T. G. Mattson, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe, "The 48-core SCC processor: The programmer's view," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[2] L. V. Kale and G. Zheng, "Charm++ and AMPI: Adaptive runtime strategies via migratable objects," in *Advanced Computational Infrastructures for Parallel and Distributed Applications*, M. Parashar, Ed. Wiley-Interscience, 2009, pp. 265–282.

[3] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, and T. Mattson, "A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Solid-State Circuits Conference Digest of Technical Papers*, 2010, pp. 108–109.

[4] R. F. van der Wijngaart, T. G. Mattson, and W. Haas, "Light-weight communications on Intel's single-chip cloud computer processor," *SIGOPS Oper. Syst. Rev.*, vol. 45, pp. 73–83, February 2011.

[5] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, "Scalable molecular dynamics with NAMD," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[6] D. Bailey, E. Barszcz, L. Dagum, and H. Simon, "NAS parallel benchmark results," in *Proc. Supercomputing*, Nov. 1992.

[7] L. V. Kale, G. Zheng, C. W. Lee, and S. Kumar, "Scaling applications to massively parallel machines using Projections performance analysis tool," in *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, vol. 22, no. 3, February 2006, pp. 347–358.

[8] B. Marker, E. Chan, J. Poulson, R. van de Geijn, R. F. Van der Wijngaart, T. G. Mattson, and T. E. Kubaska, "Programming many-core architectures - a case study: Dense matrix computations on the Intel Single-chip Cloud Computer processor," *Concurrency and Computation: Practice and Experience*, 2011.

[9] R. David, P. Bogdan, R. Marculescu, and U. Ogras, "Dynamic power management of voltage-frequency island partitioned networks-on-chip using Intel Single-chip Cloud Computer," in *International Symposium on Networks-on-Chip*, 2011, pp. 257–258.

[10] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar, "A 2 Tb/s 6×4 mesh network for a single-chip cloud computer with DVFS in 45 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 4, pp. 757 –766, April 2011.

[11] C. Clauss, S. Lankes, P. Reble, and T. Bemmerl, "Evaluation and improvements of programming models for the Intel SCC many-core processor," in *International Conference on High Performance Computing and Simulation (HPCS)*, 2011, pp. 525–532.

[12] I. Ureña, M. Riepen, and M. Konow, "RCKMPI–Lightweight MPI implementation for Intels Single-chip Cloud Computer (SCC)," in *Recent Advances in the Message Passing Interface: 18th European MPI Users Group Meeting*. Springer-Verlag New York Inc, 2011, p. 208.

[13] C. Clauss, S. Lankes, and T. Bemmerl, "Performance tuning of SCC-MPICH by means of the proposed MPI-3.0 tool interface," *Recent Advances in the Message Passing Interface*, pp. 318–320, 2011.

[14] H. Esmaeilzadeh, T. Cao, Y. Xi, S. M. Blackburn, and K. S. McKinley, "Looking back on the language and hardware revolutions: Measured power, performance, and scaling," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011, pp. 319–332.