Power-Efficient Schemes via Workload Characterization on the Intel's Single-Chip Cloud Computer

Gustavo A. Chaparro-Baquero, Qi Zhou and Chen Liu
Florida International University (FIU)
Miami, FL
{gchap002, qzhou003, chen.liu}@fiu.edu

Jie Tang
Beijing Institute of Technology
Beijing, China
tangjie.bit@gmail.com

Shaoshan Liu
Microsoft
Redmond, WA
shaoliu@microsoft.com

***Abstract*—The objective of this work is to evaluate the viability of implementing workload-aware dynamic power management schemes on a many-core platform, aiming at reducing power consumption for high performance computing (HPC) application. Two approaches were proposed to achieve the desired target. First approach is an off-line scheduling scheme where core voltage and frequency are set up beforehand based on the workload characterization of the application. The second approach is an on-line scheduling scheme, where core voltage and frequency are controlled based on a workload detection algorithm. Experiments were conducted using the 48-core Intel Single-chip Cloud Computer (SCC), running a parallelized Fire Spread Monte Carlo Simulation program. Both schemes were compared against a performance-driven, but non-power-aware management scheme. The results indicate that our schemes are able to reduce the power consumption up to 29% with mild impact on the system performance.**

I. Introduction

Multi-core and many-core platforms have gained much interest and are under great development in recent years. The basic idea of these platforms is to increase the number of cores on a single chip in an effort to continue increase the performance. Power consumption has been a great concern over High Performance Computing (HPC) platforms, because the budget for operational cost due to power consumption and cooling of HPC systems has increased considerably for the last several decades. Hence power-aware resource management is essential for current and future HPC systems. Recent years have witnessed extensive researches on power-saving schemes to relief the constraints of huge power consumption [1].

The Single-chip Cloud Computer (SCC) experimental processor is a 48-core "concept vehicle" created by Intel Labs as a platform for many-core software research [2]. One important feature of this experimental chip is its hardware and software support for Dynamic Voltage and Frequency Scaling (DVFS), which is very important for developing power-saving schemes for different applications. DVFS is a method to provide variable amount of energy for a task by scaling the operating voltage and frequency of the system dynamically. With this function, the operating frequency and voltage level can be changed according to the application needs with different goals in mind. A power reduction scheme implemented on the SCC is presented and tested with a real-life computation application in [3], showing that it is possible to reduce the power consumption on the SCC efficiently by monitoring the traffic of messages among the cores of the system, making the power management algorithm sensitive to the dynamic workload variation of the application. However, for our approach, we wanted to test the power and energy variations of the SCC platform implementing DVFS schemes using a simple parallelizable real-life application which does not change the workload of the application dynamically, but assigns an uneven distributed workload to each core from the beginning of the execution instead.

To demonstrate the efficiency of the proposed schemes, a fire spread simulation program, which is a real-life application using Monte Carlo method [4], is tested on the SCC platform using different power-saving schemes. The application itself has an uneven workload distribution, which means that some cores take longer time than other cores to finish the assigned task. Because the SCC platform is DVFS enabled, it allows us to schedule the power level of the system based on the workload of the application itself. Both off-line scheduling and on-line scheduling schemes are implemented to contrast their efficiency in power saving against a performance-driven, but non-power-aware scheduling scheme. The results show that our schemes are able to incur mild performance loss, while gaining considerable power saving.

The remainder of this paper is structured as follows. Section II goes deeper into the three main components of our design, which are the power saving schemes for HPC, the SCC platform and the fire spread Monte Carlo simulation algorithm. Section III describes the methodology used to implement the experiment on the SCC platform. Section IV summarizes the procedures to implement the experiment step by step. Section V presents the overall power consumption of the SCC platform, the time elapsed for computing each task and the total energy consumed by each measurement using each of the different power management schemes, and points out the main findings achieved. Finally, section VI concludes the paper with a brief comment of our future work.

II. Background

A. Power saving schemes for HPC

Power and energy consumption has become a hot research topic for HPC platform, because the budget for power and cooling is significant, sometimes even overwhelming. To address this growing problem, it is necessary to improve the power and energy efficiency of the system. There are many different approaches: hardware, system integration, system software, middleware, and application software [1].

Significant research in the area of power-aware computing has been done in the past few years, and there are some algorithms which have shown considerable energy saving. Many of them have been developed for systems which implemented DVFS modules. In [5] the authors divided DVFS scheduling strategies into two categories: (a) off-line trace-based schedulers, and (b) on-line/run-time profiling-based schedulers. For off-line trace-based scheduling, it is a requirement to have *a priori* knowledge of the application workload. Once the application workload is characterized, the application can be decomposed into phases and an appropriate gear is selected for the execution of each phase. It is noteworthy to mention that the term *"Gear"* refers to a specific voltage and frequency configuration. On the other hand, on-line profiling-based schedulers can dynamically set up the appropriate gears during the computation time by measuring either software or hardware variables and applying the scheduling algorithm. The main benefit of using the run-time schedulers is their ability to change gears during the execution of an application that has not been profiled yet [1].

One example of on-line scheduling algorithm, *ECOD*, is proposed in [5]. It saves on average 31.5% energy tested on different NAS benchmarks. According to the authors, their algorithm use a system-software approach that gives an accurate workload characterization using a unique synthesis of hardware performance counters in order to determine when and how to use dynamic voltage and frequency scaling. The DVFS algorithm reduces the frequency and voltage of a processor only when the processor is not needed to do useful work. *ECOD* manages the application performance and the power consumption in real time, based on an accurate measurement of CPU stall cycles due to off-chip activities.

Another example of on-line scheduling algorithm is proposed by [6]. The authors proposed a power aware algorithm that automatically and transparently adapts its voltage and frequency settings to achieve significant power reduction and energy savings with low impact in performance. The algorithm schedules the frequency and voltage in such a way that the system does not exceed a certain percentage in performance loss pre-established by the user. The whole model is based on measuring the instruction execution rate to correlate the execution-time impact with the CPU-frequency changes.

One example of off-line scheduling algorithm is proposed in [7]. The authors proposed and evaluated two algorithms that reduce power consumption of tree-based parallel sparse computations through voltage and frequency scaling. Their objective was to reduce the overall energy consumption by scaling the voltages and frequencies of those processors that are not in the critical path, which means that their algorithm is oriented towards saving power without increasing the computation time.

B. The SCC Platform

The Single-chip Cloud Computer (SCC) was introduced as an experimental chip by Intel in December 2009. It integrates 48 superscalar Pentium P54C cores. The SCC employs higher core counts instead of increasing the clock frequency to avoid excessive power demand and associated heat dissipation. The SCC platform is composed of 24 dual-core tiles as shown in Figure 1. Each of the two cores in a tile is separately supported by its own 32 KB of L1 cache and 256KB of unified L2 cache.

Both cores within a tile share a router, which controls SCC's 2D mesh network. Among the functions of the router, it generates the sidebands for error detection and routing and enables communications via virtual channels to eliminate deadlock. The most important characteristic and architectural innovation is the inclusion of a shared memory on each tile refereed as the Message Passing Buffer (MPB), which provides a fast, on-die shared SRAM. Each tile contains 16KB of MPB, which can be split evenly between the two cores. Due to the fact that conventional cache coherency protocols are hardware implemented and not easily feasible in a many-core architectures, the inclusion of this MPB is one of the most explicit supports of coherency on SCC, because it gives support to message-passing.

Message-passing is a technique that relies on software to provide cache coherency among cooperating cores without the support offered in traditional symmetric multiprocessing (SMP) configurations such as snooping protocols. Basically, the SCC processor is a message-passing chip. The cores may interact through shared memory, but with the total lack of cache coherence between cores. Therefore, the most convenient programming model to implement for this chip is built on its ability to send and receive messages between cores. The designed message passing library developed for this chip is called RCCE, which is at a certain level similar to the Message Passing Interface (MPI) library for HPC [2].

SCC cores are divided into seven voltage domains. Six of those are 2x2 arrays of tiles, as shown in Figure 2. When you change the voltage, you choose a voltage domain and change the voltage for all the cores in that domain. The SCC has one frequency domain for each of the 24 tiles. The frequency of each tile can be changed individually. However, the power management calls in the RCCE API do not currently support the control of all frequency domains independently. Instead RCCE only allows stepping of the frequency on the cores within a voltage domain. Effectively for RCCE, frequency and voltage domains coincide and are jointly called power domains [8].

In addition, the SCC implements Dynamic Power Management (DPM), which enables the SCC to reduce power consumption in multiple ways. As it will be explained in detail

later, the power level of a voltage domain and a frequency domain in SCC can be specifically manipulated based on the algorithm used. Therefore, various approaches can be implemented to save power consumption [9].

C. Fire Spread Monte Carlo Simulation (FSMCS) Algorithm

In general terms, Monte Carlo methods are a class of very well known computational algorithms that use random selections in calculations that lead to the solution to numerical and physical problems. They have been widely used in scientific researches, especially useful for simulating systems with many coupled degrees of freedom, such as fluids, disordered materials and strongly coupled solids. Relied on repeated random sampling to compute their results, simulation using Monte Carlo method is not deterministic. Namely, the output of the simulation may not always be the same for the same probability [10], as we will see in our model description later. Each Monte Carlo model involves random sampling, which means that if it is just simulated one time, the result may deviate away from its desired value to a certain level. One intuitive and obvious solution is to perform multiple simulations of the same model with the same parameters and then average all the results. The more repetition, the more accurate the results will be.

Monte Carlo method simulation applications require a high computational effort to generate a convergent result and are inherently parallel. Therefore, the SCC architecture provides an advantage for executing this kind of applications. The more number of independent execution units the better in terms of performance. We wanted to implement an easy parallelizable real life application, in order to simplify the evaluation of applying DVFS schemes.

The numerical simulation of fire spread over forest is of great importance to the society [11], [12]. There are many different kinds of models to simulate fire spread, but one thing in common among the models is that most of them are using Monte Carlo methods. It is noteworthy to mention that the fire spread model employed in this study simulates over a square forest size which is determined by a finite number of trees. The researcher may find that if he wants to get a more accurate data, he would need to repeat the simulation many times and increase the size of the simulation model to a large number. If the forest size is increased, e.g., double the number of trees on each side, we normally would observe an increase in the computation time by 4 to 5 folds. In addition, for the model used in this paper, the simulation is repeated 10000 and 50000 times, respectively. Therefore the computation demand increases tremendously, especially coupled with a large size of the model. On the other hand, in Monte Carlo methods each calculation is independent of the others and therefore suitable to be parallelized with very high performance expectations. This is one of the main reasons that we refer to many-core computing, in this case to the SCC platform specifically.

As stated above, the FSMCS is implemented to simulate the fire spread among a forest. The general procedures are explained as following:

1) Initialize the variables, such as forest size and probability of catching fire, etc.
2) Every tree has a status: “unburn”, “smoldering”, “burning”, “burnt”. Light a tree by marking its status as “smoldering”, becoming the source of fire.
3) The four trees (North, South, West, East) next to it have the same probability of catching fire.
4) To determine if a tree catches fire, a random number between 0 and 1 is generated. The tree catches fire if following condition is satisfied: random number generated is smaller than the probability of catching fire. For example, the random number generated is 0.34, and the probability of catching fire is 0.40, then the condition is satisfied, so the status of the tree changes to “smoldering”; otherwise, the tree remains “unburnt”.
5) Once a tree status has been set to “smoldering”, after one time step it is set to “burning”, and after another time step it is set to “burnt”. A “burning” tree may change the status of “unburnt” adjacent trees to “smoldering”. In other words, all the trees are checked to see whether there are “burning” trees. If so, the status of the neighboring trees (N, S, E, W) next to the “burning” tree is evaluated.
6) The fire continues until all the trees are either “burnt” or “unburnt”.

III. Methodology

In this paper we present two power management approaches: an on-line scheduling scheme and an off-line scheduling scheme. The idea for the on-line scheme is deduced

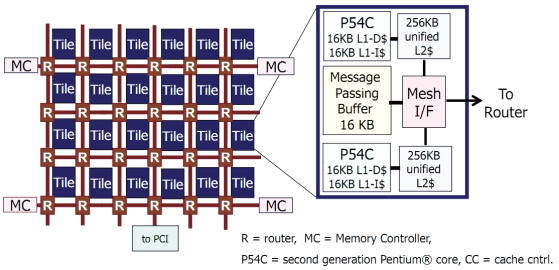

Fig. 1. Overview of the SCC platform architecture [2]

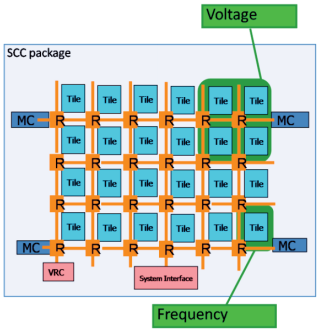

Fig. 2. Voltage and frequency domains of the SCC platform [8]

by analyzing the scheduling algorithm proposed in [5]. We followed the experimental arrangement used in [13] in order to set up the experimental environment and measure the average power consumed by the system.

The first step is to characterize the workload on the SCC platform by running the Monte Carlo method simulation program, in order to see how the power-saving schemes are applicable. It is noteworthy to mention that the SCC platform allows to change the voltage and frequency levels by groups of four dual-core tiles, dividing all the 48 cores into 6 power domains. Based on the characterization of the application, we observed that the workload of the FSMCS program is predictable and unevenly spread among the 48 cores of the SCC. Hence, it is possible to determine which gear is best suitable for each power domain for each workload. Afterwards, we continue to check if it is possible to manage the gears for each power domain dynamically or statically. We also noticed that some cores are finishing their task much faster compared with other cores. According to this, we developed two variations for the on-line scheme to adjust the power levels of each of the six power domain in the SCC.

A. Workload Characterization

The main motivation of characterizing the workload is to interpret the difference of computation time among each core, based on which we are able to schedule the frequency and voltage to a proper level to save power and energy.

In this experiment, the benchmark simulates 192 different probabilities of assessing fire risk, and each probability is repeated 10000 and 50000 times. It is easy to understand that for the lower probabilities (corresponding to the case that the fire may die out soon and not be able to spread away), the computation time is less than that of higher probabilities. Thus, generally it should follow the trend of increasing computation demand as the probability increases.

Being aware of the difference in computation time for each core, we propose two different approaches, both of which are supposed to reduce power consumption. One of them is to schedule a lower power level for those cores that finish their tasks first, and keeps the high power level for those that finish last. To maintain zero performance loss, we have to make sure the cores running at lower frequency takes time no longer than the one with highest power level. The other scheme is to keep the cores at higher power level first, and lower the frequency when the core finishes its task. By doing so, we are supposed to save energy during the idle time. These two schemes will be explained in details in the following sections.

B. Power Level Scheduling

1) Off-line scheduling: Off-line scheduling requires us to perform the workload characterization as a prerequisite. Taking into account the unbalanced workload distribution seen on the characterization of the application, we can schedule a lower frequency for those power domains which finish earlier than the rest, and schedule a higher frequency for those using more time. If the frequency is properly chosen, then the computation time of any lower-frequency cores should not exceed the longest computation time with higher frequency. As long as this condition is satisfied, our approach does not incur any performance loss, while saving energy on the cores with lower frequency. Three different power levels are scheduled in the approach as shown in Table I. Previous characterizations and measures about power consumption on the SCC platform shows that the power Configuration I corresponds to a high power mode; Configuration III corresponds to a low power mode; and Configuration II corresponds to a low power mode running at its highest frequency, which gives a good relation of energy savings and performance decrease. Due to the system constraint, we changed the frequency and voltage level of the entire power domain at the same time. Table II identifies each of the six power domains and also indicates the number of each of the eight cores within a specific power domain, highlighting the Master Core (MC). In our off-line scheduling scheme, we scheduled Configuration II for the power domains 1, 2 and 3 and Configuration I for the power domains 4, 5 and 6.

2) On-line scheduling: Realizing the uneven distribution of the workload among the cores from the workload characterization, we are able to schedule the frequency and voltage level at a proper manner. The main idea of this approach is to lower the power level of the cores which finish before the others. But in order for the power level change to be in effect, it has to be called by the master core of each power domain, so we let the master core changes the power domain configuration as long as it finishes its job, regardless of whether other cores in the same power domain have finished their jobs or not, which is the first variation of the scheme; in the second variation, we make the master core of each power domain wait until the job completion of all the cores within that power domain to change the gear of the domain.

This on-line approach differentiates from the off-line approach in the following aspects. In the first place, this approach changes frequency and voltage level dynamically during the execution time, i.e., after finishing its task. In addition, different power levels are implemented in on-line scheme. In this approach, the program runs at Configuration I (1.1V /

TABLE I
THREE POWER CONFIGURATIONS

Configuration/Gear	I	II	III
Voltage Frequency	1.1V 800MHz	0.8V 533MHz	0.8V 400MHz

TABLE II
GEAR MAPPING OF EACH POWER DOMAIN FOR THE OFF-LINE SCHEME. THE MASTER CORE (MC) OF EACH POWER DOMAIN IS HIGHLIGHTED

Power Domain	MC	Cores							Voltage Frequency	Configuration
1	**0**	1	2	3	12	13	14	15	0.8 v 533 MHz	II
2	**4**	5	6	7	16	17	18	19	0.8 v 533 MHz	II
3	**8**	9	10	11	20	21	22	23	0.8 V 533 MHz	II
4	**24**	25	26	27	36	37	38	39	1.1 V 800 MHz	I
5	**28**	29	30	31	40	41	42	43	1.1 V 800 MHz	I
6	**32**	33	34	35	44	45	46	47	1.1 V 800 MHz	I

800MHz) at the beginning, and we dynamically make it call the power level change to Configuration II (0.8V/533MHz) later on.

IV. Data Gathering

In this section, we explain the process we conduct to prepare and settle the experimental environment. We started from an MPI-based program for the FSMCS [4], the algorithm for which has been explained in previous sections. Then we have to modify the MPI functions using special functions designed for SCC platform only.

The new parallelized code for the fire spread Monte Carlo simulation has the following parameters as inputs:

- *Size of Forest*. With this parameter we are able to specify the number of trees at each side of the square forest.
- *Number of Probabilities*. This parameter allows us to specify the number of different probabilities to be simulated. The fire spread probabilities are evenly distributed to cores executing the program. For example, using 192 probabilities with 48 cores, each core will be simulating 4 different probabilities.
- *Number of Iterations*. Once each core is assigned with an equal number of probabilities, each core must simulate its assigned probabilities by a specific number of times, which is the Number of Iterations. For instance, specifying a 50000 means each core will perform the simulation 50000 times, calculating an average of the results at the end of the computation.
- *Power Domain Configuration*. As stated before, this parameter sets the voltage and frequency configuration of the power domain for running the simulation.

A. Procedure for Power Measurement

In order to measure the overall power of the system, we followed the procedure suggested in [14] to extract the voltage and current used to calculate the overall power consumption. The SCC platform uses a lightweight embedded controller as the Board Management Controller (BMC). The BMC is used for hardware control and monitoring. Different variables can be monitored by the BMC, including the voltage and current of the power supply feeding the entire SCC chip. These two variables can be read in order to calculate the power consumption of the 48 cores at a given time. The FSMCS program invokes the procedure a finite number of times, in this case 100. This allows us to measure the power values along the completion of the computation by specifically making Core 0 call the procedure repeatedly. Since each power measure implies accessing the Management Console PC (MCPC) OS file system (Ubuntu 10.04 64-bits) [13] several times as well as the Board Management Controller (BMC) [15] of the SCC in order to read the registers containing the status of the machine, the latency of each measure is a little less than one second. We have excluded such latency from the execution time measures.

B. Procedure for changing frequency and voltage

In order to add the capability of changing the frequency and voltage to the FSMCS program, we used the corresponding system functions specified in [16], which allow us to specify the lowest voltage level that is consistent with the input value of the frequency and initiate the voltage change and to change the frequency in the power domain without affecting the voltage.

C. Experimental Scenario

In order to test the power-reduction schemes previously explained, the original FSMCS program was modified in order to comply with the specifications of each scheme. Each scheme was run four times, combining the values of size of the forest as 10 and 20 and the number of iterations as 10000 and 50000. The completion time of each core was recorded, as well as the overall system power values measured during the computation. Following is described the set up for each scheme.

1) *Configuration I static:* The simulation program was run with all the power domains configured to 1.1 V and 800 MHz, which corresponds to Configuration I in Table I. The voltage and frequency values were set up at the beginning of the computation and the configuration was not changed during the whole computation.
2) *On-line Asynchronous:* The simulation program calls the voltage and frequency change of each power domain once the master core in each domain finishes its computation. Namely, it calls a change in frequency from 800 MHz to 400 MHz and a change in voltage from 1.1 V to 0.8 V (from Configuration I to III). The reason is that only the master core within each power domain is actually able to execute the voltage and frequency changes.
3) *On-line Synchronous:* The simulation program calls the voltage and frequency change of each power domain once every core within a power domain finishes its computation. Namely, the master core of the power domain calls a change in frequency from 800 MHz to 400 MHz and a change in voltage from 1.1 V to 0.8 V (from Configuration I to III) only when every core in the power domain finishes its task.
4) *Off-line Scheme:* The simulation program sets the voltage and frequency for each power domain statically according to Table II, which means half of the power domains are running Configuration I and the other half running Configuration II.
5) *Configuration III static:* The simulation program was run with all the power domains configured to 0.8 V and 400 MHz, which corresponds to Configuration III. The voltage and frequency values were set up at the beginning of the computation and the configuration was not changed during the whole computation.

V. Analysis of Results

The workload characterization for the FSMCS program measured by the completion time of the task assigned to each core is shown in Figure 3, varying the size of the forest and the number of iterations. It was evaluated using a power configuration of 1.1V and 800MHz. From Figure 3 we can see an uneven distribution of the workload for each core in the system. The lower cores finish the task earlier than the upper ones for all the four cases.

Of the four cases shown in Figure 3, Figure 4 shows the completion time for simulating a forest size of 20 with 50000 iterations per probability, sorted by the power domain number instead. The master core of the power domain, which is the one responsible for power configuration, is specially identified. It can be seen that the behavior of the previous figure remains almost constant i.e, lower cores finishing first than upper cores. This fact proves the suitability of the application for exploring the response of the SCC platform applying different power-management schemes via workload characterization, because every simulation will have a known uneven workload distribution.

The data provided by this two figures also serve as the baseline configuration for comparing with the results from our proposed schemes. For example, from Figure 4 we see that power domains 1, 2 and 3 do not affect the critical path for finishing the computation of the simulation, whereas power domains 4, 5 and 6 contain cores with maximum workload, so in this case our critical path is composed by these power domains.

The overall system power consumption comparison across different schemes with various simulation sizes is shown in Figure 5. The corresponding normalized values for the overall system power consumption using the *Configuration I static* scheme as reference are shown in Figure 6. It can be seen that the *Configuration I static Scheme* serving as our reference shows an average overall power consumption of 94.5 watts. The *On-line Synchronous Scheme*, the *Off-line Scheme*, the *On-line Asynchronous Scheme*, and the *Configuration III static Scheme* show an average overall power consumption of 82 watts, 67.8 watts, 67.3 watts and 35.6 watts respectively.

The overall computation time comparison across different schemes with various simulation sizes is shown in Figure 7. The corresponding normalized values for the overall computation time of the system using the *Configuration I static* scheme as reference are shown in Figure 8.

The overall energy consumption comparison across different schemes with various simulation sizes are shown in Figure 9. These values are calculated by multiplying the overall power consumption with the total completion time for each scheme. The corresponding normalized values for the energy consumption of the system using the *Configuration I static*

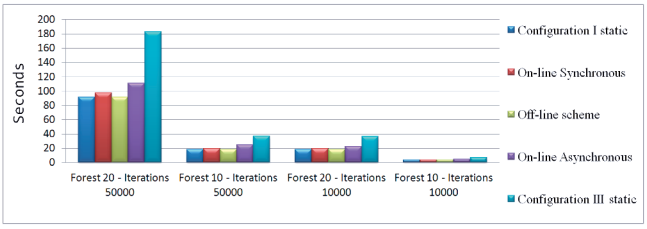

Fig. 7. Computing time spent by running the program on 48 cores

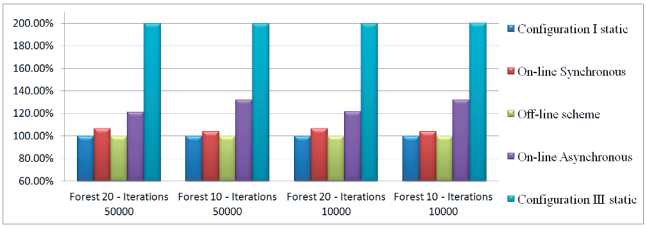

Fig. 8. Computing time normalized to *Configuration I static* scheme

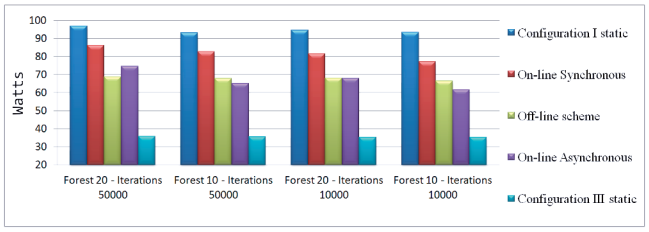

Fig. 5. Average power consumed by running the program on 48 cores

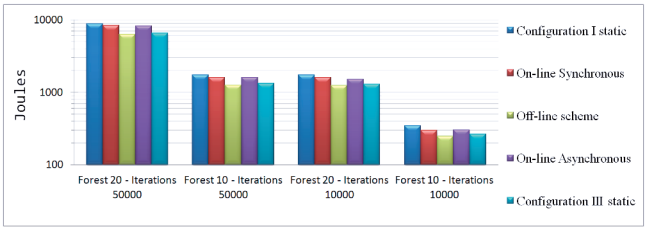

Fig. 9. Average energy consumed by running the program on 48 cores

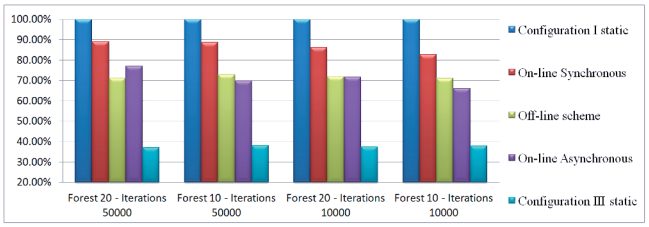

Fig. 6. Power consumption normalized to *Configuration I static* scheme

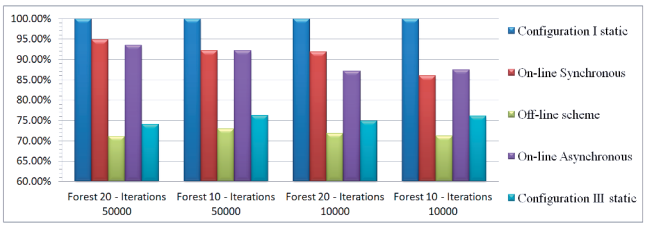

Fig. 10. Energy consumed normalized to *Configuration I static* scheme

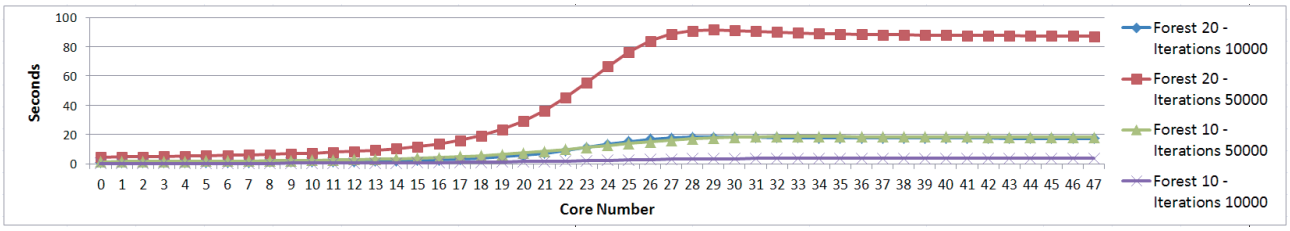

Fig. 3. Workload Characterization for the FSMCS program measured by the completion time of the task assigned to each core.

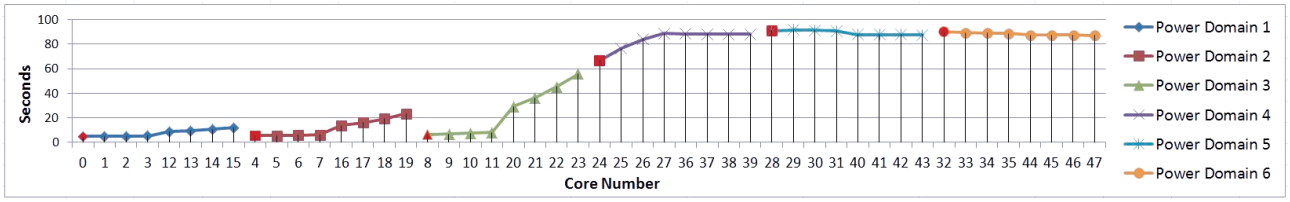

Fig. 4. Workload Characterization for the FSMCS sorted by power domain number. The first core of each domain is the master core and is specially identified.

scheme as reference is shown in Figure 10.

Figure 11 shows a general comparison of the schemes across average power consumption, average energy consumption and average completion time, which are calculated by averaging the results for running each configuration with each scheme. Figure 12 also shows a general comparison among schemes but sorted by the scheme used. Looking at the normalized values, it can be seen that the *On-line Synchronous Scheme* uses 87% of the power used by the reference scheme, which is *Configuration I Static*. The *On-line Asynchronous Scheme* and *Off-line Scheme* both use 71% of the power used by the reference scheme. The *Configuration III static Scheme* uses 38%.

In terms of computation time, the *Off-line Scheme* has no increase when compared with the baseline scheme, while the *On-line Synchronous Scheme* and the *On-line Asynchronous Scheme* have an increase of 5% and 27%, respectively. The *Configuration III static Scheme* doubles the computation time. The off-line scheme can keep the same performance as *Configuration I static Scheme* because it has *a priori* knowledge about the application, which the on-line schemes do not have and that is why they incur some loss in performance.

In terms of energy consumption, the *Off-line Scheme* spent 72% of the energy of the reference scheme, which is even lower than the *Configuration III static Scheme* with a normalized value of 75%. The *On-line Synchronous Scheme* and the *On-line Asynchronous Scheme* spent 91% and 90% respectively. Clearly the *Off-line Scheme* is the best energy-friendly scheme when compared with others.

Figure 13 shows the comparison of the reduction in energy provided by each scheme versus the increase in time caused by each scheme. The values are calculated by applying $\Delta Energy/\Delta Execution\_time$, contrasting to the reference scheme *Configuration I static Scheme*. It is measured in Joules per second. The figure shows that the *Off-line Scheme* is the optimal scheme for the testing application since it reduces the

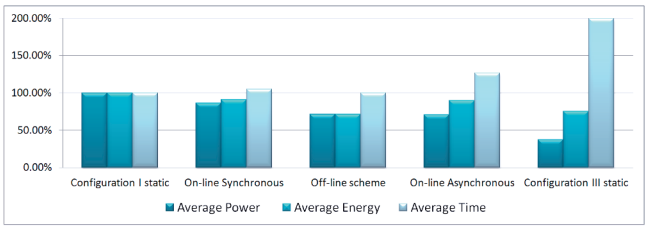

Fig. 11. Comparison of schemes sorted by power, energy and comp. time

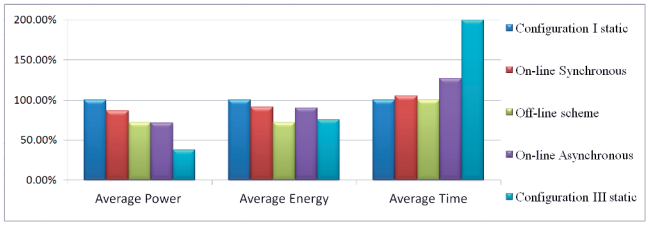

Fig. 12. Comparison of schemes sorted by schemes

overall power consumed while keeping the completion time equal to the reference scheme, which is the *Configuration I static Scheme*. Nevertheless, the *On-line Synchronous Scheme* shows an improvement of 178.2 J/s, which is 4.9 times larger than the improvement shown by the *On-line Asynchronous Scheme* with an improvement of only 36.2 J/s. These results suggest that on-line schemes could behave better if the scheduled change of gear takes into consideration the idle condition of every core within the power domain.

It is noteworthy to mention that these results lead us to think that for an overall power-sensitive application, of which the critical timing path can be effectively characterized, an off-line scheme could work more efficiently than an on-line scheme on the SCC platform. Nevertheless, the off-line scheme performs better because we have previously characterized the application and that gives us some knowledge about it beforehand.

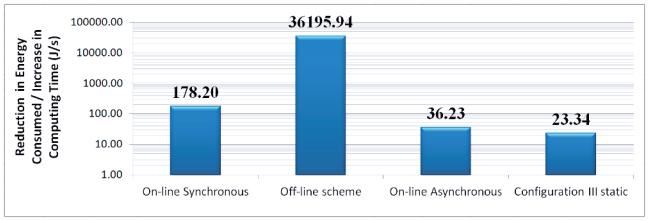

Fig. 13. Comparison of the reduction in energy versus the increase in time cross schemes, contrasting to the reference scheme *Configuration I static Scheme*

Certainly, we can argue that if such *a priori* information is not available, which is the case for runtime decision making for real-life applications, then an on-line scheme could be effective as well. The on-line scheme with its two variations we proposed showed a significant reduction in power and energy consumption with mild impact on performance. The results also show that the on-line schemes implemented on the SCC are able to provide efficient power management, taking advantage of the main benefit of using the run-time scheduler which has the ability to change gears dynamically during the execution of an application.

VI. Conclusion and Future Work

This paper presents a behavioral quantification of the performance, overall power and energy consumption of the new Intel Single-Chip Cloud Computer research platform, running a workload characterizable application, which is a Fire Spread Monte Carlo Simulation program. The simulation was performed using two different power saving schemes: an on-line scheme with two variations, and an off-line scheme. We presented the different results contrasting the overall system power consumption, the total completion time as well as the total energy consumption during the entire computation for each of the schemes used. Results show that it is possible to implement a common parallelized scientific program over the SCC platform and apply a power reduction scheme efficiently with a low performance reduction, taking advantage of the dynamic voltage and frequency scaling capability the SCC platform provides. Our data also shows that it is possible to achieve up to 29% power savings with our power reduction scheme executing a workload characterized application and when compared with a high-power-consuming scheme.

As far as future work is concerned, we can further evaluate the power saving schemes using more fine-grain power configurations. We have shown that it is possible to reduce the overall power consumption of the SCC platform by applying workload-based power reduction schemes. We would explore the power management approaches on a wide variety of other benchmarks and applications, as well as associated load balancing issue. The idea of allocating idle cores to other applications would result in rescheduling of tasks from heavy-loaded cores to idle ones, which we are actively considering to adopt as a future feature for this study.

Acknowledgment

We would like to thank the anonymous reviewers for their insights and comments that help us improve the quality of this paper. We would like to thank Intel Labs for providing us with SCC system in material transfer to conduct this research. This work is partly supported by the National Science Foundation under grant number ECCS-1125762. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] G. Valentini, W. Lassonde, S. Khan, N. Min-Allah, S. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. Pecero, D. Kliazovich, and P. Bouvry, "An overview of energy efficiency techniques in cluster computing systems," *Cluster Computing*, pp. 1–13, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10586-011-0171-x

[2] 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 *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.53

[3] R. David, P. Bogdan, R. Marculescu, and U. Ogras, "Dynamic power management of voltage-frequency island partitioned networks-on-chip using intel's single-chip cloud computer," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, July 2011, pp. 257–258.

[4] D. Joiner, *Basic MPI Tutorial*, The Shodor Education Foundation, Inc, www.shodor.org/refdesk/Resources/Tutorials/BasicMPI/index.php, 2011.

[5] S. Huang and W. Feng, "Energy-efficient cluster computing via accurate workload characterization," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, may 2009, pp. 68 –75.

[6] C.-h. Hsu and W.-c. Feng, "A power-aware run-time system for high-performance computing," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1–. [Online]. Available: http://dx.doi.org/10.1109/SC.2005.3

[7] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan, "Reducing power with performance constraints for parallel sparse applications," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, april 2005, p. 8 pp.

[8] T. Kubaska, *The SCC Programmers Guide*, 0th ed., Intel Labs, http://communities.intel.com/docs/DOC-5684, May 2010.

[9] M. Baron, "The single-chip cloud computer: Intel networks 48 pentiums on a chip," *Microprocessor Report: The Insider Guide to Mi- croprocessor Hardware*, April 2010.

[10] G. Karniadakis and R. Kirby, *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and their Implementation*. Cambridge University Press, 2003.

[11] Y. Carmel, S. Paz, F. Jahashan, and M. Shoshany, "Assessing fire risk using monte carlo simulations of fire spread," *Forest Ecology and Management*, vol. 257, no. 1, pp. 370–377, 2009.

[12] W. Song, F. Weicheng, W. Binghong, and Z. Jianjun, "Self-organized criticality of forest fire in china," *Ecological Modelling*, vol. 145, no. 1, pp. 61–68(8), November 2001.

[13] P. Thanarungroj and C. Liu, "Power and energy consumption analysis on intel scc many-core system," in *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*, nov. 2011, pp. 1 –2.

[14] T. Kubaska, "How to read scc voltages with a rcce program," Intel Labs, http://communities.intel.com/docs/DOC-5463, Tech. Rep., August 2010.

[15] ——, *The SccKit 1.4.x Users Guide*, part 8 ed., Intel Labs, http://communities.intel.com/docs/DOC-6240, September 2011.

[16] T. Mattson and R. van der Wijngaart, *RCCE: a Small Library for Many-Core Communication*, 0th ed., Intel Corporation, May 2010.