Green software development for multi-core architectures

Fábio Diniz Rossi, Miguel Gomes Xavier, Endrigo D'Agostini Conte, Tiago Ferreto, César A. F. De Rose
Pontiphical Catholic University of Rio Grande do Sul
Porto Alegre – Brazil
fabio.diniz@acad.pucrs.br

***Abstract*—Advances in computer architecture to provide higher parallelism (e.g. hyper threading and multi-core) usually incur in higher complexity in software development. Applications should be designed to use efficiently the additional resources in order to improve its performance. However, the popularity of mobile devices and recent studies in IT-related energy consumption have driven software developers to focus also on energy efficiency. Besides improving applications' performance, software developers should aim at minimizing the amount of energy consumed by the applications. Energy saving becomes an important non-functional requirement for new applications. This paper evaluates the behavior of applications on multi-core architectures and proposes energy-saving alternatives for software development.**

***Keywords*-multi-core; hyper-threading; energy efficiency; software development;**

I. Introduction

For decades, major computer processor manufacturers accompanied Moore's Law by adding more features to their processors and increasing significantly processor's frequency. In the early 2000s, physical limitations, such as heat dissipation and data synchronization problems, led companies to find other alternatives to keep improving processor's performance. These issues tend to the development of new alternatives providing higher parallelism, such as: hyper-threading, multi-core CPUs, accelerated processing units, among others.

Currently, processors with multiple processing units are available on different devices, such as: computers, smartphones, tablets, and network devices (e.g. routers and switches). This increase in computer parallelism required programmers to employ parallel programming techniques when developing applications. The applications should present better performance on newer devices with more execution cores, as traditionally happened before with the increase in processor frequency.

Recently, developers started also to take into consideration the energy consumption of their applications, specially due to the growth of the mobile market, with battery-based devices. The power consumption of a chip is given by the equation $P = C.V^2F$ [1], where P is power, C is the capacitance being switched per clock cycle (proportional to the number of transistors whose inputs change), V is voltage, and F is the processor frequency (cycles per second). Frequency increasing means increasing the amount of energy used by a processor.

Consequently, there is a trade-off between resource use and energy savings. If more resources are used, it will result in a better performance, but more energy will be consumed. If energy-saving techniques are used, there will be a performance loss due to the decrease in processor frequency. Some questions are raised about saving energy in multi-core processors, and these would serve as hypotheses evaluated in this paper. The hypotheses are:

- In a processor with two physical cores, if all the threads migrate to one processing unit and the other unit is turned off, will there be energy savings?
- The use of hyper-threading increases by 30% the applications performance, but do the energy consumption follows the same increase?
- If the threads are evenly distributed among all available cores, can it save energy?

This paper focus on evaluating these hypotheses through the experiments executions in a real testbed. The paper is organized as follows: Section II presents an overview on multi-core and hyper-threading technologies and the importance in reducing energy consumption; Section III presents related work; Section IV describes the testbed used in the experiments; Section V evaluates the results obtained in the experiments regarding performance and power consumption; Section VI concludes the paper and presents future work.

II. Background

Due to the large amount of resources that current devices offer, software development generally does not consider issues related to energy consumption. Even in mobile development, in which there is a greater concern for the lifetime of the batteries, it is not common to find oriented programming models for this purpose. In this section, we present the Hyper-Threading technology and discuss the energy savings as a non-functional requirement for software development. We also present some studies on design patterns focused on green computing.

A. Hyper-Threading

Hyper-Threading (HT) [2] is Intel's implementation for simultaneous multithreading (SMT). It simulates several logical processors in a single core processor allowing the operating system to schedule many threads or processes simultaneously through each processor. HT excels by taking advantage of superscalar architectures, decreasing the number of dependent instructions on the pipeline. This feature can increase up to 30% of the overall application performance.

Each logic processor has its own advanced programmable interrupt controller (APIC) and a set of registers. Other resources of the physical processor, such as cache memory, buses, execution unity, etc., are shared between logical processors.

Despite the fact that Hyper-Threading's energy consumption is slightly higher, the overall amount of energy needed to run a task is lower than the same execution on a single core processor. This happens because the task finishes in fewer cycles, therefore apparently consuming less.

HT processors are present in smartphones, tablets and in several other embedded devices. In the context of smart cities, power-aware devices can reduce the environmental impact caused by heat and harmful gases emission to the atmosphere. This can happen in several ways, from software development focused on sustainability, to techniques that allow reducing processor's frequency. This paper proposes that energy consumption should be treated as a non-functional requirement for software development. It suggests a paradigm shift, that requires from the developer, a greater knowledge of the architecture to be used. On the other hand, such paradigm shift can bring significant advantages in terms of energy consumption.

B. Energy Savings as a Software Requirement

Software development involves several stages and activities that regardless of the method chosen, occur to allows delivering the software working properly, within budget and deadlines for its development. To achieve the project goals, all development activities have to be carefully worked out and developed, either using a development approach as UP (Unified Process) or Agile methodologies (XP, SCRUM, etc.). In any one of them, we will find activities of requirements analysis, design, architecture definition, codification and others. A consistent analysis of the requirements, i.e., identification, definition, prioritization and classification of the main problems that the future software must solve is the basis of a software project success.

Among these, we focus on the requirements analysis. Requirements [3] express characteristics or properties that a system must have or do, as well as its operating restrictions. Requirements can be classified in functional requirements or non-functional requirements. Non-functional requirements relate to the characteristics of the application in terms of performance, usability, reliability, security, availability, maintainability and technologies involved.

As the physical limits of silicon were achieved, concerns about energy consumption becomes a non-functional requirement that must be taken into account for future software development.

More than ever, the developer must know the characteristics of the hardware device that will serve as the base for the software. However, when software development is more specific to the hardware, it is less general for other architectures. This is the motivation to set up a design pattern for green computing.

C. Design Patterns for Green Computing

Design patterns are formalization of best practices to be implemented in application. Just as there are design patterns that describe the best way to explore the characteristics of the application on the hardware architecture, there should be a pattern that takes into account energy consumption. Litke et al. [4] analyzes three design patterns: Factory method, observer and adapter. The paper presents that the factory and observer patterns increase energy consumption. This is expected, since these patterns have focus on other metrics, not energy consumption.

We can note that the natural way to develop power-aware applications lies in the behavioral profile characterization of the application on the available resources, and through this information, deciding which power saving techniques or technologies can be used in this application.

III. Related Work

Tan et al. [5] discusses the motivation for conducting characterization of energy in an embedded operating system, proposing a new method to perform a systematic characterization. The method proposed consists of two parts: mapping the sets of components that combine the power consumption of the device, and obtaining quantitative models for these characteristics. This enables the models creation that can act on the operating system to save energy.

Using machine learning, the work of Kan et al. [6] maintain a record of the application behavior and modifies the frequency of the processor through Dynamic Voltage and Frequency Scaling (DVFS), aiming to adjust the power consumption. The paper shows an average energy-saving of 9.1%. In the same way, the work of Schuster and Brinkschulte [7] proposes to use model-driven development based on state machines. Applications are defined on a high level of abstraction and efficient implementations are adapted to the target platform.

The work of Siegmund et al. [8] presents a comparison of techniques for energy-saving on mobile devices, based on

aspect-oriented and feature-oriented programming. In this same context, the work of Siebra [9] argues that requirements facing the energy consumption should be created, and it can result in significant power savings. As a preliminary work, Chatzigeorgiou and Stephanides [10] propose energy efficiency metrics for the development of software requirements. The paper considers the sources of power consumption in every digital circuit and measures are calculated using hierarchical information defined in software, forming a basis for the energy metric definition. This measure can be used to decide the level of energy consumption of the entire software system.

This paper is focused on multicore architectures and many of computer microarchitecture research on reducing energy consumption in processors. Some works propose to dynamically adjust processor resources as shown by Albonesi et al. [11], other works presented by Li et al. [12] and by Sasanka et al. [13] present comparisons of energy consumption between SMT (Simultaneous Multithreading) and CMP (Chip Multiprocessing) or multicore as presented by Kumar et al. [14] and Kumar et al. [15]. Most of these studies were performed by analytical methods. Grant and Ahmad [16] follow a work that explores energy efficiency in an asymmetric architecture using hyper-threading, and proposes a new scheduling algorithm that maintains a global energy consumption with minimal impact on application performance. Focused on multithreaded applications in OpenMP, the study showed that the most used applications maintained a lower power consumption when using non-hyper-threading. The purpose of the paper was to show that with some changes in scheduling of tasks it would be possible save energy while maintaining the same performance.

IV. Experimental Setup

For the evaluations of this paper, we used the testbed shown in Figure 1. Some features of this NUMA [17] architecture (Intel(R) Xeon(R) 2.27GHz) are important for the tests, as the existence of two physical processors with four cores using HT and Linux kernel version supporting the CFS (Completely Fair Scheduler) [18].

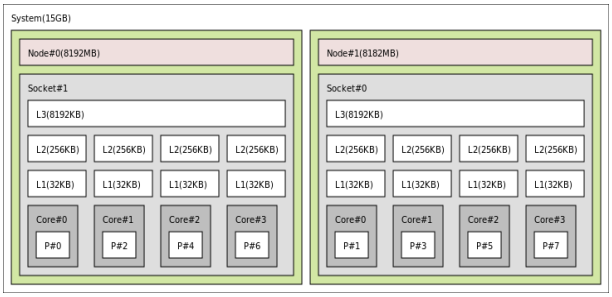


Figure 1. Test Architecture

Our experiments were conducted using the NPB benchmark suite, as presented in Kale [19], one of the most well-known and widely used benchmarks for HPC. NPB is derived from computational fluid dynamics (CFD) applications and consist of five kernels and three pseudo-applications. We chose the EP and LU to represent the behavior of two different types of threads, CPU-intensive (EP) and IO-intensive (LU).

- **EP (Embarrassingly Parallel):** It is a benchmark that provides an estimation of most reachable performance for analysis of floating point calculation. It is normally used to evaluate performance without the significant cost of interprocess communication, therefore an ideal CPU-intensive benchmark.
- **LU (Lower-Upper Symmetric Gauss-Seidel):** It is a benchmark that performs matrix factorization being a product of a lower triangular matrix and upper triangular matrix. It performs complex interprocess communication. It is a very complete HPC benchmark and is similar to the one used to build the Top500 list [1], ranking the world's most powerful supercomputers.

Aiming to answer the questions presented in this paper, three different configurations were tested:

- 1 physical processor (called 1P)
- 2 physical processors + Affinity (called 2P-Af)
- 2 physical processors + CFS (called 2P-CFS)

The first scenario (1P) can be seen in the Figure 2, and consists of only one physical processor available to the operating system, and the second physical processor disabled.

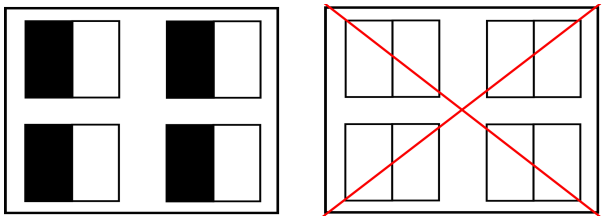


Figure 2. 1 Socket Test Scenario

The second scenario (2P-Af) presented in the Figure 3 is similar to the first scenario, but presents the second physical processor available, although idle. This is possible by scheduling all threads to the first physical processor, performing affinity. The second physical processor being in idle state still consumes energy, though with a minimum of frequency.

The third and last scenario (2P-CFS) (Figure 4) shows the two physical processors available, and the threads are distributed according to the rules of the CFS scheduler.

[1]http://www.top500.org

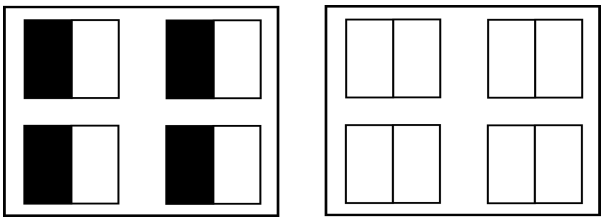


Figure 3. 2 Sockets + Thread Affinity Test Scenario

In multi-core architectures, CFS splits threads between the available processors to optimize resource usage.

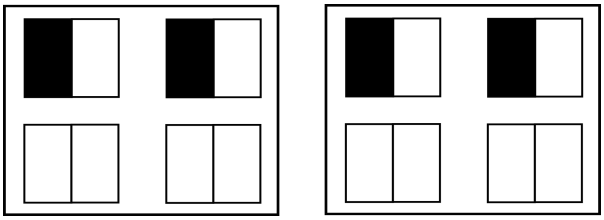


Figure 4. 2 Sockets + CFS Test Scenario

For each scenario and for each group of benchmarks tested, all tests were performed fifteen times, showing no significant differences between runs.

V. Experimental Results

The graphs show results of benchmarks executions running 4, 8 and 16 threads, which are sets of executions that represent values when compared to the numbers of cores: a number less, equal and greater about them. We believe that this set describes the real use of multithreaded applications on multi-core environments. In our tests, presented in Figure 5, the x-axis represents each specific execution time and the y-axis represents the power consumption in watts.

In Figure 5 5(a) we can see energy consumption measurement of executions of EP benchmark using 4 threads. In this scenario, when compared to 2P-CFS, the 2P-Af showed a reduction in energy consumption by 3%, while the 1P showed a reduction in power consumption of 13%. This is the only set of execution in the CFS scheduler that loses in energy consumption for other models, and this is due to the low load imposed by 4 threads in architecture.

Executions of the LU benchmark with 4 threads shown in Figure 5 5(b) did not present the same behavior, in which the 2P-Af and 1P consumed more energy (55% and 51% respectively) than the model with only CFS.

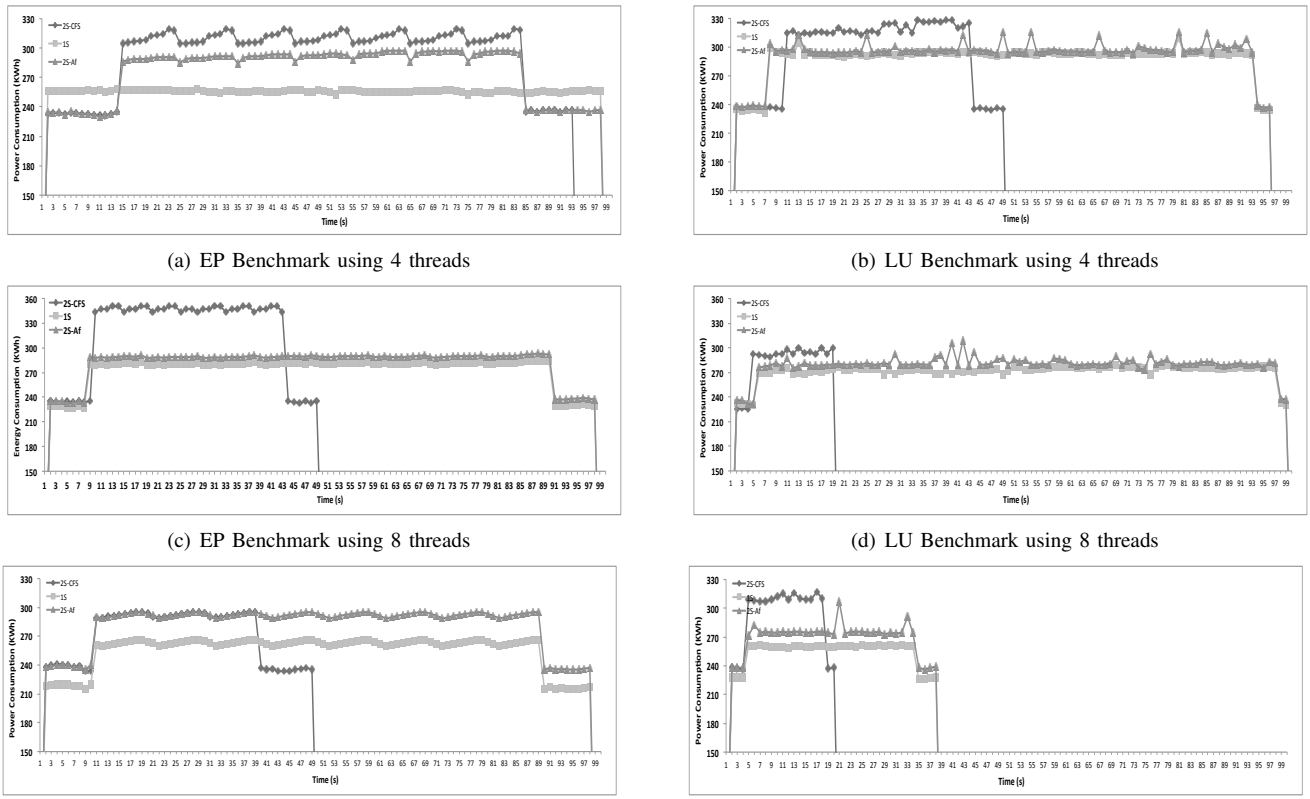
Running 8 threads, the 2P-CFS is energetically efficient when compared to other models. Figures 5 and 5(c) show the benchmark EP with an increase of 44% in the consumption by the use of 2P-Af and 33% by using 1P. The LU benchmark shows an increase of power consumption at the level of 150% by 2P-Af and 137% by 1P.

The behavior exhibited by executions with 16 threads is the same as 8 threads, but with different values of consumption. In the EP benchmark, 2P-Af had an increase in energy consumption of 69%, while 1P had an increase in power consumption by 44%, compared to 2P-CFS. The LU benchmark showed an increase of 24% when running 2P-Af and 16% when running on 1P, when compared with 2P-CFS.

Based on the results presented, we can perform some analysis on the trade-off between resource usage and power consumption, as follows:

- It is possible to make adjustments in power consumption based on the threads location. Depending on the threads position and their characteristics, it can have energy efficiency. Although the gain in energy cost seems small, in various architectures with multiple cores and threads this may represent a great gain.
- Benchmarks using two physical processors certainly consume more energy each time interval. However, in most cases it executes on a smaller time execution, which allows a reduction in total power consumption. This shows that the CFS scheduler allocation of threads among the available resources is energetically efficient.
- In all cases presented, the LU benchmark consumes more energy than the EP benchmark. It means that the application that perform memory access consumes more power than CPU-intensive applications. Running LU benchmark presents a performance gain when running on two physical processors and the CFS scheduler, and this is due to the increased size of the available cache memory. EP uses a set of only 20 megabytes of data, while the LU benchmark uses 650 megabytes of data.
- The use of two physical processors, scheduling threads only to the first processor and turning the second processor off, show almost the same time execution, however the affinity option consumes more energy because it keeps the second physical processor consuming energy while idle.
- When using only one physical processor holding the second physical processor off, consumption values are always lower, but the run time is always higher.

These analysis of the trade-offs between resource usage and energy savings enables fine adjustment in software development, since the concern with energy consumption is taken into account as a non-functional requirement. However, there is a clear dependence on the knowledge of processor architecture, which makes the development of energy-aware software less general. This is not really a limitation, given the software development for embedded platforms is quite specific to the hardware.



(a) EP Benchmark using 4 threads

(b) LU Benchmark using 4 threads

(c) EP Benchmark using 8 threads

(d) LU Benchmark using 8 threads

(e) EP Benchmark using 16 threads

(f) LU Benchmark using 16 threads

Figure 5. Execution Time vs. Power Consumption Evaluations

VI. Conclusion and Future Work

Computers with multiple processing units have been used for several years already, mainly in high performance computing, but recently the interest in the topic grew because of physical limitations of the increase in frequency processing. This increase in frequency and consequent increase in performance has a cost in energy consumption.

Therefore, there is an increasing concern about energy consumption of computers, and several studies are being performed trying to reduce energy consumption and its environmental impact, keeping the performance gains.

Most studies aimed at energy savings in computing are focused on infrastructure, using techniques to change processor frequency, perform more efficient forms of data communication, operating systems using sleep states as standby and hibernate, new policies for scheduling and use of the devices. However, there have been few studies on how to develop software with a focus on energy-saving, with the intention of using all available resources, without consuming more energy to it.

This study attempts to answer some questions that can direct the software development targeting an energy-aware environment. To this end, we used in our evaluations two benchmarks representing the two main types of resource demanding applications: CPU-intensive and IO-intensive. This paper answers some questions about the use of multi-core processors in an energetically efficient manner. This study was guided by three questions presented in the introduction.

The first hypothesis is about the threads migration to only one core, with the intention to turn the other cores off to save energy. This hypothesis is true, since in all tested cases we have observed that there is indeed energy-saving while maintaining idle cores turned off.

The second hypothesis checks if hyper-threading impacts on energy consumption, as well as impacts the execution time of jobs. We can see an increase in energy consumption that accompanies the percentage of performance gain. This is due to a larger area of the processor components that must maintain to support hyper-threading technology, which consumes more energy.

The third hypothesis verifies that a fair distribution of threads on all available cores is energy efficient. This hypothesis is true, because the CFS scheduler is present in most current kernels that support other technologies such as DVFS. Thus, the scheduler distributes the threads on the processor cores, and the kernel can adjust the frequency of these, saving energy by fitting the frequency and voltage.

During all assessments in this paper, we have moni-

tored metrics including number of cores, speedup, context switches, total CPU used, CPU idle, time of CPU idle and temperature. With a correlation analysis, through which we identified most influence metrics on energy consumption. Because of that, it will be possible to develop a power consumption equation for multi-core applications. As future work, the same set of experiments could be run to test energy efficiency in embedded processors such as PowerPc. Another path to follow is to consider some primitives at design phase of the application, such as page, stack, heap and other kinds of memory data structures in order to evaluate the trade-off between performance and power consumption.

REFERENCES

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[2] X. Tian, Y.-K. Chen, M. Girkar, S. Ge, R. Lienhart, and S. Shah, "Exploring the use of hyper-threading technology for multimedia applications with intel® openmp* compiler," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 36.1–.

[3] J. Nicolás and A. Toval, "On the generation of requirements specifications from software engineering models: A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 9, pp. 1291–1307, Sep. 2009.

[4] A. Litke, K. Zotos, E. Chatzigeorgiou, and G. Stephanides, "Energy consumption analysis of design patterns," in *Proceedings of the International Conference on Machine Learning and Software Engineering*, 2005, pp. 86–90.

[5] T. K. Tan, A. Raghunathan, and N. K. Jha, "Energy macromodeling of embedded operating systems," *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 1, pp. 231–254, Feb. 2005.

[6] E. Y. Y. Kan, W. K. Chan, and T. H. Tse, "Eclass: An execution classification approach to improving the energy-efficiency of software via machine learning," *J. Syst. Softw.*, vol. 85, no. 4, pp. 960–973, Apr. 2012.

[7] S. Schuster and U. Brinkschulte, "Model-driven development of ubiquitous applications for sensor-actuator-networks with abstract state machines," in *Proceedings of the 5th IFIP WG 10.2 international conference on Software technologies for embedded and ubiquitous systems*, ser. SEUS'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 527–536.

[8] N. Siegmund, M. Rosenmüller, and S. Apel, "Automating energy optimization with features," in *Proceedings of the 2nd International Workshop on Feature-Oriented Software Development*, ser. FOSD '10. New York, NY, USA: ACM, 2010, pp. 2–9.

[9] C. Siebra, P. Costa, R. Miranda, F. Q. B. Silva, and A. Santos, "The software perspective for energy-efficient mobile applications development," in *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*, ser. MoMM '12. New York, NY, USA: ACM, 2012, pp. 143–150.

[10] A. Chatzigeorgiou and G. Stephanides, "Energy metric for software systems," *Software Quality Control*, vol. 10, no. 4, pp. 355–371, Dec. 2002.

[11] D. H. Albonesi, R. Balasubramonian, S. G. Dropsho, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster, "Dynamically tuning processor resources with adaptive processing," *Computer*, vol. 36, no. 12, pp. 49–58, Dec. 2003.

[12] Y. Li, D. Brooks, Z. Hu, and K. Skadron, "Performance, energy, and thermal considerations for smt and cmp architectures," in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, ser. HPCA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 71–82.

[13] R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes, "The energy efficiency of cmp vs. smt for multimedia workloads," in *Proceedings of the 18th annual international conference on Supercomputing*, ser. ICS '04. New York, NY, USA: ACM, 2004, pp. 196–206.

[14] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, pp. 32–38, 2005.

[15] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," 2003, pp. 81–92.

[16] R. E. Grant and A. Afsahi, "Power-performance efficiency of asymmetric multiprocessors for multithreaded scientific applications," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, pp. 25–29.

[17] F. Hoza and V. Radulescu, "A numa architecture for parallel structures," in *Proceedings of the 26th IASTED International Conference on Modelling, Identification, and Control*, ser. MIC '07. Anaheim, CA, USA: ACTA Press, 2007, pp. 302–307.

[18] C. S. Pabla, "Completely fair scheduler," *Linux J.*, vol. 2009, no. 184, Aug. 2009.

[19] V. Kale, "Towards using and improving the nas parallel benchmarks: a parallel patterns approach," in *Proceedings of the 2010 Workshop on Parallel Programming Patterns*, ser. ParaPLoP '10. New York, NY, USA: ACM, 2010, pp. 12:1–12:6.