Evaluating Concurrency Throttling and Thread Packing on SMT Multicores

Marco Danelutto, Tiziano De Matteis, Daniele De Sensi and Massimo Torquati
Department of Computer Science, University of Pisa
Largo B. Pontecorvo 3, I-56127, Pisa, Italy
Email: {marcod, dematteis, desensi, torquati}@di.unipi.it

Abstract—**Power-aware computing is gaining an increasing attention both in academic and industrial settings. The problem of guaranteeing a given QoS requirement (either in terms of performance or power consumption) can be faced by selecting and dynamically adapting the amount of physical and logical resources used by the application. In this study, we considered standard multicore platforms by taking as a reference approaches for power-aware computing two well-known dynamic reconfiguration techniques: CONCURRENCY THROTTLING and THREAD PACKING. Furthermore, we also studied the impact of using simultaneous multithreading (e.g., Intel's HyperThreading) in both techniques. In this work, leveraging on the applications of the PARSEC benchmark suite, we evaluate these techniques by considering performance-power trade-offs, resource efficiency, predictability and required programming effort. The results show that, according to the comparison criteria, these techniques complement each other.**

Keywords: concurrency throttling, thread packing, power-aware, green computing, PARSEC benchmarks.

I. INTRODUCTION

In recent years power consumption management has become a major concern for data centers due to economic cost, reliability problems and environmental reasons [1]. *Power-awareness* is the ability of a system to be aware of its power consumption and to change its operating behaviour in order to meet a given requirement on this important metric.

A typical approach to reduce the power consumptions of applications consists in slowing them down, while still guaranteeing a performance level imposed by the user [2], [3]. In other cases, explicit power consumption requirements may be specified in order to do not exceed the available thermal or power budget (the so called *power capping*) [4], [5], [6]. Similar requisites are also supported by hardware vendors, which provide mechanisms and tools to specify power capping values (e.g. Intel RAPL [7]).

Different factors play a crucial role in achieving the desired goals in terms of performance and/or power budget. For example, it is particularly important to choose the proper number of threads/processes that compose the parallel application, how they are mapped on the available resources and the operational configuration of the machine used (e.g., frequency and voltage). Due to different application workloads and architecture features, the intercourse among these factors is often unclear and the problem of selecting their appropriate combination is a challenging task.

In this work we try to entangle this complexity by focusing on *Concurrency Throttling* (CT in the following), i.e. change the number of threads of the parallel application, and *Thread Packing* (TP), i.e. select the right mapping of threads over the available cores. The study has been conducted considering the PARSEC 3.0 [8] benchmark suite comprising a number of multi-threaded applications covering a wide range of workload characteristics. Applications are executed on multicore CPUs, which represents a common scenario in modern high-performance clusters, data centers and commodity devices. In addition to this, for both techniques, we will analyse the impact of *Simultaneous Multithreading* (SMT in the following), an hardware facility that is widely diffused in modern processors and allows multiple independent threads to be executed simultaneously on the same physical core, to better utilise the available resources. In this case, the physical core is composed by a fixed number of *contexts*, one for each thread. The evaluation will take into account several aspects: performance/power trade-offs, efficiency and required programming effort.

To the best of our knowledge, this is the first systematic and comprehensive comparison of these techniques in terms of optimality and coverage of the desired performance/power requirements. In addition to this, as a main contribution of this work, we provide to the application programmers the insights on how these two techniques can help in achieving the desired levels of performance and power consumption, and what to expect when using them in real application scenarios. The results obtained, clearly show that there is no clear winner between CT and TP techniques, none of them is always better than the other, instead, they complement each other for different aspects. However, if we mostly take into account the programming effort and the application overhead introduced by the technique, the TP-based mechanisms are the best choice. This is also an important message for run-time developers that want to provide efficient abstractions for dynamic reconfiguration of parallel applications for multicore platforms.

The rest of this paper is structured as follows. In Section II we describe some related works that use CT and TP mechanisms. Then, starting from a motivating example, we clarify the main objectives of this work (Section III). In Section IV we present the results of our evaluation. Eventually, in Section V we draw the conclusions and we discuss some possible future directions.

II. Related Work

In the past few years, power efficient computing has drawn the attention of research and industrial communities. Considering a parallel program in execution over a multicore CPU, the programmer or the run-time system may act on various "*knobs*" for controlling performance and power consumption, as for example: the level of concurrency, threads-to-cores affinity, and the operating status of the CPU (i.e. frequencies and voltages). Turning these knobs properly allows to obtain different results in terms of performance and power consumptions. Among the different mechanisms and techniques that can be exploited in this regard, Concurrency Throttling (CT), Thread Packing (TP) represent the most studied ones.

Most of the works proposed in the literature consider only one of these mechanisms at a time. Authors in [9], [10] used CT to decide the best amount of threads to be used in running the parallel application in order to achieve the maximum performance. Indeed, due to contention on shared resources, such as caches and floating points units, using all the available cores does not always correspond to the most effective choice. Another common approach is to use CT to provide specific guarantees on performance [2], [11]. In this cases, the application programmer (or user) sets a minimum performance requirement for its parallel application, either in terms of maximum execution time or minimum rate of elements to be processed per time unit. Then, the runtime system will select the best amount of threads and cores to use during the application run to satisfy such requirement. A number of of recent works combine CT with other techniques (e.g. *Dynamic Voltage and Frequency Scaling*, DVFS) to guarantee a given level of performance while minimising, at the same time, energy or power consumption [12], [13], [14].

Conversely, TP has been mainly used in [15], [4] to provide explicit guarantee on the maximum power consumption and in [16], [17] to minimise the energy consumption.

Differently than the previous cited works, [18] uses both CT and TP mechanisms. In this scenario, *Hardware Simultaneous Multithreading* (SMT) represents a feature of modern multicore CPUs that has not been completely explored in the context of power aware computing. Few works such as [19], [10] studied this facility, considering the different contexts present in each core of the CPU while using the CT mechanism.

Despite the wide literature describing and using CT and TP techniques, to the best of our knowledge, is still missing a thorough study on how these techniques can be exploited and combined when considering also SMT. We believe that such comparison would allow researchers and developers of parallel application to better understand the impact of each technique on both performances as well as power consumption.

III. Motivating Example

In this section we consider a simple motivating example used to support the discussion and to justify the main objectives of this study. We consider four different cases: Concurrency Throttling with and without SMT (CT-HT and CT, respectively), Thread Packing with and without SMT (TP-HT and TP, respectively).

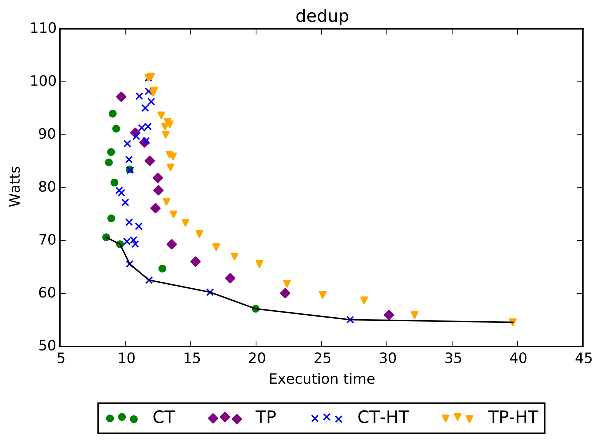

Fig. 1: Comparison of power-performance trade-offs for the Dedup application. For the sake of readability, we considered only configurations with even numbers of cores/threads.

Figure 1 shows the results obtained running the DEDUP application (one of the benchmark of the PARSEC suite) on a 24 cores 2-way Hyper-threaded machine (HyperThreading is the Intel's implementation of SMT). Each point in the figure represents a specific configuration that uses a given number of resources (threads and cores) for each of the four cases considered. On the x-axis we have the execution time (in seconds) of the application, and on the y-axis its average power consumption over the program execution (in watts). In the figure, we have also reported the Pareto frontier (the continuous line), i.e. the set of configurations such that does not exist any configuration having both lower execution time as well as lower power consumption. The rationale is that, when reconfiguring an application, only the points on the Pareto frontier should be considered. On the other hand, the points on the Pareto frontier belong to different reconfiguration techniques, therefore there is no mechanism (among those considered) which is always better than the others for all possible configurations. If we consider the CT case, we can see that there is no configuration that is able to consume less than 58 Watts, conversely if we consider the TP-HT there is no configuration that is able to finish its execution in less than 13 seconds. Furthermore, increasing the number of resources (thread and/or cores) above a given threshold does not provide any benefit on the execution time, instead, it does increase the power consumed. In general, if we had considered only some particular techniques (e.g., TP and TP-HT), we would have always selected non-optimal configurations.

Despite DEDUP represents a notable case, other PARSEC benchmarks exhibit different behaviours. To mention, in FACESIM almost all the possible configurations lie on the Pareto frontier. In such cases, the reconfiguration mechanisms complement each other and the best choice depends on the specific constraints set by the user. Finally, there are cases where all the points on the Pareto frontier belong to a single reconfiguration technique. We will analyse each PARSEC benchmark in detail in Sect. IV.

IV. EVALUATION

The evaluation of the different mechanisms is performed on the set of applications provided by the PARSEC benchmark suite [8]. PARSEC parallel benchmarks differ one each other in terms of: application domain, programming model (pipeline, data-parallel and unstructured), granularity, working set size, data sharing and data exchange patterns. This heterogeneity allowed us to study the reconfiguration techniques on a wide range of real world applications. Among the different input sizes provided by PARSEC, we choose the *native* one, in order to have a real world behaviour of the applications. We studied all PARSEC applications except X264, which we were not able to run on our system. For all the benchmarks we tested the reference *Pthreads* version, except for FREQMINE that is shipped with only the *OpenMP* version.

For each PARSEC benchmark, the number of threads (also referred as *concurrency degree* in the following) to activate has been selected by using the `-n` parameter provided by the `parsecmgmt` tool. All experiments were conducted on an Intel workstation with 2 Xeon E5-2695 v2 @2.40GHz CPUs, each with 12 2-way HyperThreaded cores, running a Linux based operating system. The frequency has been set to the maximum for all the experiments. To measure the power consumption, we used the open source C++ MAMMUT library[1]. Threads-to-cores affinity has been enforced by using the TASKSET Linux utility. In all the presented results, we only considered the power consumption and the time spent in the so-called *region of interest* (ROI), i.e. the parallel sections of the applications, without considering initialisation and cleanup phases, to avoid distortions of the measurements.

The different techniques are compared considering a different concurrency degree, a different number of cores as well as the number of SMT contexts. Each of these configurations is uniquely identified by:

1) the technique used, i.e. CT, CT-HT, TP or TP-HT;
2) n, the concurrency degree with which the PARSEC application is executed;
3) m, the number of computational resources used.

Concerning the point 3), the type of computational resources used changes according to the technique selected (i.e., point 1) above): for CT and TP we will consider distinct cores, i.e. we will not use the HyperThreading facility available in the target platform. On the contrary, for CT-HT and TP-HT we will take into account also the core contexts.

When testing the CT-based techniques, we change the level of concurrency degree n and the number of used resources accordingly (i.e., $m = n$). For the case in which the HyperThreading capability is not used, we launch the application with at most 24 threads, that is $1 \leq n, m \leq 24$; for CT-HT we execute it with maximum 48 threads exploiting also all the available contexts ($1 \leq n, m \leq 48$). For TP-based techniques we fix the concurrency degree to the maximum number of available resources (i.e. $n = 24$ and $n = 48$ for TP and TP-HT, respectively) while we change the number m according to the resources effectively used (i.e. $1 \leq m \leq 24$ for TP and $1 \leq m \leq 48$ for TP-HT).

For each of the PARSEC benchmark, we executed all the available configurations multiple times. In the following, we will consider the averaged values of the execution time and power consumption. As shown in Fig. 1, for each benchmark we obtain in this way different set of points, each one referring to the different techniques.

In the rest of this section we evaluate *Concurrency Throttling* and *Thread Packing* in terms of power-performance tradeoffs, their efficiency as well as the programming effort required to use them.

A. Power-Performance Tradeoff

In this section, we evaluate how CT and TP techniques can be used for selecting the best configuration. The user imposes a requirement on one of the two metrics, either performance by expressing the maximum execution time or power consumption by setting the maximum number of watts to consume. Among all the configuration that satisfy the given constraint, the *optimal* one is the solution that is able to minimise the other metric. That is, the lowest power consuming configuration which guarantees a given execution time or the most performing configuration that does not exceed a specific power budget. The optimal configurations are the ones that fall on the Pareto frontier (see Fig. 1). To clarify this aspect, consider the case in Fig. 1. Suppose that we want an execution time of maximum 25 seconds. To analyse the quality of the TP mechanism we compare the power consumed by the configuration on the Pareto frontier that is able to satisfy the requirement (58 Watts) with the power consumed by considering only TP mechanism (62 Watts). Then, we state that the TP mechanism has a *relative loss* of 6.89% with respect to the optimal solution.

To avoid biases due to specific constraint choices, we compare the different mechanisms on multiple constraints, similar to what is done in other works [3]. For example, if an application has a minimum execution time of 20 seconds and a maximum of 220, then we will use as constraints $20s, 22s, 24s, \ldots, 220s$. Basically, we slice the range of execution times in 100 intervals of the same length. By doing so, we are able to test a wide range of scenarios and to cover the entire performance spectrum. The same approach has also been adopted in the evaluation of the power constraint.

In Table I we report the percentage of constraints which cannot be satisfied by each of the analysed techniques. For performance constraints, TP-based solutions are usually less performing, thus they cannot satisfy requirements close to the performance peak of the application. On the other hand, they are able to better exploit the available resources and to reach low power consumption configurations which are not present in the CT-based mechanisms.

[1]http://danieledesensi.github.io/mammut/

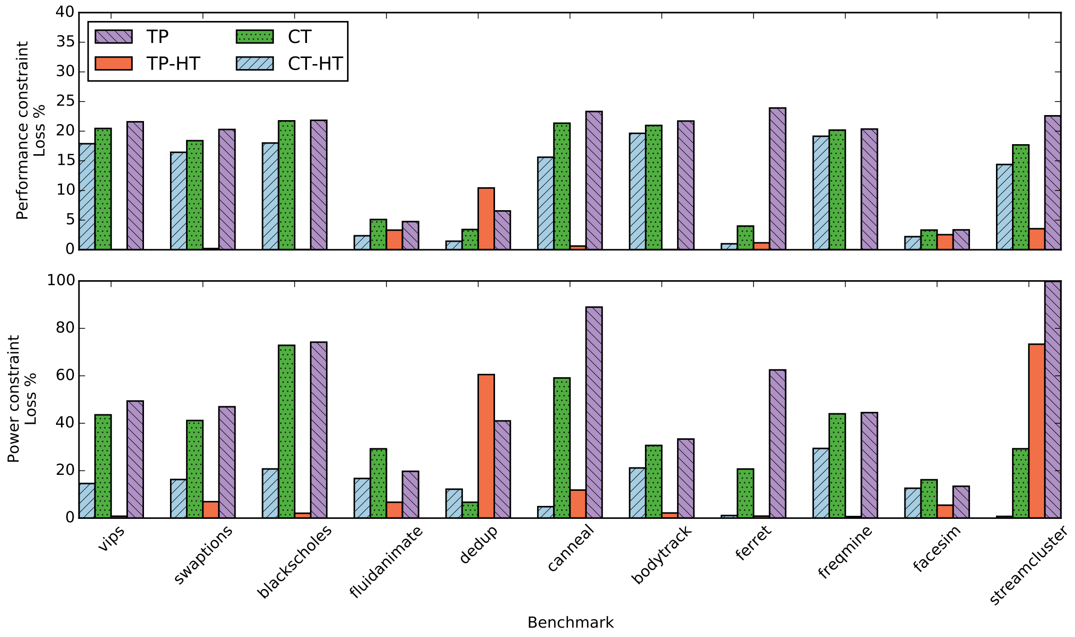

Fig. 2: Comparison of the loss of each mechanism with respect to the best one.

TABLE I: Percentage of constraint which cannot be satisfied by each mechanism (averaged over all the benchmarks).

SCENARIO	CT	TP	CT-HT	TP-HT
PERFORMANCE CONSTRAINT	0.45	0.95	0.72	1.54
POWER CONSTRAINT	10.86	6.63	9.36	0.0

In Figure 2 we report the loss percentage of each mechanism with respect to the optimal solution. For a given application, each bar represents the average value computed over all the possible requirements. TP-HT exhibits the lowest loss in almost all the benchmarks both for power consumption and performance constrained scenarios.

B. *Efficiency*

Another interesting evaluation concerns the average number of core contexts used by each reconfiguration techniques. A common assumption when execution a parallel program on a multicore platform is that the assigned cores may be exclusively used by the application for the entire execution. In such scenario, using as few cores as possible is a very desirable property.

In Table II, it is shown, for the performance and the power consumption requirements, the average number of additional core contexts used by each technique to minimise the other requirement. For each PARSEC applications, we considered only the set of requirements that can be reached by using the given mechanism. Then, we compare the number of used core contexts against the ones employed by the optimal configuration.

TABLE II: Efficiency comparison: number of additional core contexts used with respect to the optimal solution.

SCENARIO	CT	TP	CT-HT	TP-HT
PERFORMANCE CONSTRAINT (AVG)	0.25	0.89	2.34	4.39
POWER CONSTRAINT (AVG)	0.65	1.39	15.05	20.07

As it can be noticed, due to less contention, the techniques that do not use HyperThreading are able to consume less resources. In particular, CT-based techniques are able to reduce the number of core contexts used with respect to TP-based mechanisms.

C. *Programming effort*

An important aspect that we would like to analyse concerns the implications of the chosen mechanism in terms of programming effort required to implement and use it.

Thread Packing is usually implemented by changing threads-to-core affinity (e.g., by using the `taskset` Linux utility) to let threads run on specific cores or contexts of the CPU. The application structure is not changed, thus this approach does not require any additional effort to the application developer.

On the other hand, *Concurrency Throttling* may require more effort since the application must be able to change, while it is running, the number of threads it uses. In pure functional computations (e.g. *task-farms* parallel pattern), the effort required to change the number of threads is limited. However, in the most general case where the application has a state shared or partitioned among its threads, if the number of threads is changed at runtime, there may be the necessity to reorganise or redistribute the state among them in order to preserve the correctness of the computation. Since this depends on the specific structure and characteristics of the application, it must be done manually by the application programmer. This process is time-consuming, error-prone and in some cases it may even be unfeasible.

D. Summary

In Table III, we provide a qualitative summary of the analysis done throughout this paper. For each evaluation we assigned a score between one and three (the higher the better).

TABLE III: Summary of the performed evaluation.

Analysis	CT	TP	CT-HT	TP-HT
Power-Performance Tradeoff	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
Efficiency	⋆⋆⋆	⋆⋆	⋆	⋆
Programming effort	⋆	⋆⋆⋆	⋆	⋆⋆⋆

As it can be noticed there is not a clear winner and the compared techniques complement each other. Thread Packing allows to reach a slightly better power-performance trade-off with respect to Concurrency Throttling and requires a limited programming effort to be implemented. On the contrary, Concurrency Throttling exhibits the best efficiency in terms of resources used. By using HyperThreading, it is possible to achieve better power-performance trade-offs while increasing the amount of used resources.

V. Conclusions and future work

This work evaluates the *Concurrency Throttling* and the *Thread Packing* techniques considering SMT multicore platforms. The evaluation has been performed on Parsec benchmarks, showing that *Thread Packing*-based mechanisms have a lower impact on the programming effort and are characterised by better power-performance tradeoffs, despite using in general more resources thatn *Concurrency Throttling*-based solutions.

As a future work, we will study others low-level mechanisms used for controlling power and performance such as the *Dynamic Voltage Frequency Scaling* (DVFS). Moreover, additional platforms such as the Intel Knights Landing (4-way SMT cores) and the IBM Power server (8-way SMT cores) will be used to validate the results presented in this work.

Acknowledgements

This work has been partially supported by EU H2020-ICT-2014-1 project RePhrase (No. 644235) "*REfactoring Parallel Heterogeneous Resource-Aware Applications - a Software Engineering Approach*"

References

[1] J. G. Koomey, "Worldwide electricity used in data centers," *Environmental Research Letters*, vol. 3, no. 3, p. 034008, 2008.

[2] J. Li and J. F. Martínez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," *Proceedings - International Symposium on High-Performance Computer Architecture*, vol. 2006, pp. 77–87, 2006.

[3] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann, "A Probabilistic Graphical Model-based Approach for Minimizing Energy Under Performance Constraints," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 267–281, Mar. 2015.

[4] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: adaptive DVFS and thread packing under power caps," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 '11*. New York, New York, USA: ACM Press, Dec. 2011, p. 175.

[5] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic Knobs for Responsive Power-aware Computing," *SIGPLAN Not.*, vol. 46, no. 3, pp. 199–212, 2011.

[6] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar, "The need for speed and stability in data center power capping," in *Proc. of IGCC 2012*. IEEE Computer Society, pp. 1–10.

[7] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz, "Beyond dvfs: A first look at performance under a hardware-enforced power bound," in *Proc. of the 2012 IEEE 26th Intl. Parallel and Distributed Processing Symposium Workshops & PhD Forum*, ser. IPDPSW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 947–953.

[8] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[9] M. Danelutto, D. De Sensi, and M. Torquati, "Energy driven adaptivity in stream parallel computations," in *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on*, March 2015, pp. 103–110.

[10] M. Curtis-maury, F. Blagojevic, C. D. Antonopoulos, and D. S. Nikolopoulos, "Prediction-based Power-Performance Adaptation of Multithreaded Scientific Codes," *IEEE Transactions on Parallel and Distributed Systems*, 2008.

[11] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, and A. Leva, "Comparison of Decision-Making Strategies for Self-Optimization in Autonomic Computing Systems," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 7, no. 4, pp. 1–32, Dec. 2012.

[12] D. De Sensi, "Predicting performance and power consumption of parallel applications," in *24th Euromicro Intl. Conf. on Parallel, Distributed, and Network-Based Processing (PDP)*, Feb 2016, pp. 200–207.

[13] D. De Sensi, M. Torquati, and M. Danelutto, "A reconfiguration algorithm for power-aware parallel applications," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, Oct. 2016.

[14] T. De Matteis and G. Mencagli, "Keep calm and react with foresight: Strategies for low-latency and energy-efficient elastic data stream processing," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2016, pp. 13:1–13:12.

[15] P. Petoumenos, L. Mukhanov, Z. Zheng Wang, H. Leather, and D. S. Nikolopoulos, "Power Capping: What Works, What Does Not," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, Dec. 2015, pp. 525–534.

[16] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz, "Prediction models for multi-dimensional power-performance optimization on many cores," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 250–259.

[17] S. Holmbacka, E. Nogues, M. Pelcat, S. Lafond, D. Menard, and J. Lilius, "Energy-Awareness and Performance Management with Parallel Dataflow Applications," *Jour. of Signal Processing Systems*, Nov. 2015.

[18] M. J. I. Yang Ding, Mahmut K, Padma Raghavan, "A Helper Thread Based EDP Reduction Scheme for Adapting Application Execution in CMPs."

[19] S. Sridharan, G. Gupta, and G. S. Sohi, "Holistic run-time parallelism management for time and energy efficiency," in *Proceedings of the 27th international ACM conference on International conference on supercomputing - ICS '13*. New York, New York, USA: ACM Press, Jun. 2013, p. 337.