# Detecting Memory-Bound Parallel Regions to Improve the Energy-Efficiency of Applications

Gabriel Bronzatti Moro*, Lucas Mello Schnorr*
* Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil

***Abstract*—Performance and energy consumption are fundamental requirements in some areas. It is a challenge to maintain the same performance using less energy. In this article, we analyze the behavior of the Lulesh application using a pre-run that provides the necessary information to identify its memory-bound regions. Our results present the detection of the Memory-Bound behavior of such application. Among these results, it is possible to visualize each region covered by the thread and its respective location in the source code of the application, as well as its misses rate using hardware counters provided by the PAPI tool. As future work, we intend to implement a library that will use this knowledge base to reduce the processor frequency in the Memory-Bound regions. Our assumption is that in these regions it will be possible to reduce the energy consumption of the application.**

## I. Introduction

Energy consumption is an important issue in large data centers as well as in battery-dependent devices. There is a trade-off between performance and energy consumption, enabling one to find a best configuration. Scientific applications, image and video processing are widely used in High-Performance Computing, they can pottentially be considered as Memory-Bound applications because they are limited by access time to memory [1].

DVFS (Dynamic Voltage and Frequency Scaling) approach is used to reduce the power consumption in Memory-Bound Applications, using a low processor frequency when the application is waiting for memory. Some studies investigate the appropriated frequency adjustment using information about applications behavior, platform characteristics, and load balancing. Freeh et al. [2] define an approach for distributed applications, Laurenzano et al. [3] present a technique for sequential and parallel applications, Spiliopoulos et al. [4] create a tool for sequential applications (using the loop level) and Millani and Schnorr [5] use a granularity by parallel regions for the frequency adjustment. Among those studies, it is possible to understand that for OpenMP parallel applications, it is necessary more efforts to optimize Memory-Bound regions in parallel applications using information about the applications behavior. In this context, our main objective is to identify precisely the Memory-Bound regions of an application, so that it is possible as future work to create an automatic approach with the variation of processor frequency by parallel region.

This paper is organized as follows. Section II shows some concepts about Application Behavior and Energy Consumption. In Section III, the related works regarding automatic phase detection for HPC applications. After it, the Section IV is presented our proposal and its corresponding methodology. The Section V describes the results. Section VI concludes the paper with the main contributions and future works.

## II. Application Behavior and Energy Consumption

Mapping, Stencil, and Reduction are parallel application models widely used for shared memory programming. Mapping uses a process that launches several threads (fork), after this division, a specific processing occurs in each thread or a similar processing on different data [6]. Stencil applications perform the convolution approach to obtain a new result. In this case, it is used "n" iterations to get the result for a particular matrix's element, using its neighboring cells [7]. Reduce (Division-and-Conquer) applications have a combination of small solutions obtained from the processing of parts of a collection of input elements. The output of this type of implementation is an unique result, which was calculated by consecutive reduction steps.

For example, a Vector Multiplication Algorithm is different than a Search Algorithm in Graphs. The Algorithm of Search in Graphs has a more Memory-Bound behavior than the Vector Multiplication Algorithm because when a search is performed on graphs, the nodes are far apart from each other whereas in Vector Multiplication Algorithm the elements are better located, allowing good Cache spatial location.

In Memory-Bound applications, the performance increases when the rate of misses in the L2 Cache is reduced [8]. Otherwise, Non-Memory-Bound applications, the performance does not improve by reducing cache misses rate. Hardware counters are used to trace such metrics, enabling one to collect the number of accesses to different Cache Levels, Main Memory, the number of instructions executed, and other data about the platform and application performance behavior.

In addition to the misses rate for the Cache L2 to define the behavior of an application, it is possible also use the Instructions by Cycle (IPC) metric. This metric allows to check how many instructions the program performs per processor cycle. That way, it enables to verify if reducing the number of instructions of an application it will occur the increase of its performance. If this happens it is possible to classify the application such as Non-Memory-Bound or CPU-Bound application [8].

The power consumption can be measured by energy sensors or estimated from models (theoretical estimate) [9]. Some tools

provide the access to energy sensors. Intel PCM is an example of a tool that allows obtaining the total energy consumption spent by the application, it provides the CPU (with Cache consumption), and Main Memory energy [10].

Several works use Dynamic Voltage and Frequency Scaling (DVFS) to save energy by reducing the processor frequency. This approach allows reducing the processor frequency in certain parts of the program execution, which have a more Memory-Bound behavior [11]. The CPUfreq framework allows changing the frequency used by the processor, this framework provides Performance, Powersave, OnDemand, Userspace, and Consertative mode [12]. Each of the modes defines a priority to be achieved. Performance uses the highest frequency available, unlike the Powersave mode that always use the lowest frequency to obtain the lowest possible energy consumption [13]. Still, OnDemand mode performs the frequency adjustment according to processor usage. Generally, CPUFreq is used with Userspace mode to adjust appropriately the processor frequency.

## III. Related Work

There is no definitive solution to detect if a code region is more memory or CPU-bound. It is possible to find investigations about phase detection to sequential [4], [3], distributed [2] and shared-memory parallel applications [3], [5].

Spiliopoulos et al. [4] show a tool called Power-Sleuth that is able to provide a detailed description of the behavior of an application when executed at a respective frequency. They use three techniques in their work: phase detection, Dynamic Voltage and Frequency Scaling (DVFS), and correlation models. Their approach identifies the application's regions using ScarPhase library that uses an execution history. The program's functions have a similar behavior defined by misses rate, number of access to memory, and others metrics. This article uses only sequential applications, the investigation of the Memory-Bound regions can be obtained in a coarser granularity in the timestamps between samples. On the other hand, parallel applications are executed on different threads, each thread may have a different behavior according to load balancing, application model, or communication type [4]. In addition to Spiliopoulos et al. [4] approach, Poellabauer et al. [1] describe a technique called Feed-back loop, this approach uses the metric MAR (data cache misses divided by instructions executed). From the MAR equation, it is possible to analyze the percentage of misses in the instructions executed by the application as a whole. This metric allows investigating the application's behavior in a particular architecture. The results show an energy saving of up to 27% for the six applications executed.

Laurenzano et al. [3] define an automated approach that allows selecting the most appropriate processor frequency for a given program loop. The processor frequency is chosen based on a static analysis (performed before execution) and another analysis performed during the execution time of the application, using the obtained traces. The authors use several benchmarks, based on the Pcubed (PMaC's Performance and Power benchmark) framework, which allows the exploration of different behaviors of interaction loops, in order to define a characterization for the target platform. Characterization of the platform defines values such as power consumption, performance, execution patterns and processor frequencies. The results obtained in the experiment can be used later as a knowledge base, so it is possible to visualize the behavior of the energy consumption when adjusting the characterization factors of the platform. Among the results obtained by the work, the best is the reduction of up to 10.6% in energy consumption

Unlike Laurenzano et al. [3], Freeh et al.[2] present an approach to distributed memory for MPI applications. This approach uses a frequency for each node, the frequency is defined by a heuristic called `Gear` that defines the gain between energy consumption and performance. With the trace obtained from a pre-execution, the approach defines blocks (like Basic Blocks) that perform operations. For each block the desired gain is set. The gain is the best configuration found between power consumption and performance for a certain phase of the application. Their results show an advantage for more than half of the applications performed. The best result obtained is the reduction of energy consumption by 9% and the execution time by 1%. On the other hand, the technique of Freeh et al. [2], Ge et al. [14] present an approach that also uses the DVFS technique for parallel applications in clusters, but in this approach rather than using the more Memory-Bound regions of the application to apply the frequency reduction, the authors perform the reduction of processor frequency when CPU performance is not required, for example when it occurs communication between MPI processes.

For parallel applications that use the OpenMP library, Millani and Schnorr [5] present an approach which analyzes the parallel regions of a program using a detailed analysis with the technique of Design of Experiments and Screening Design. The authors use seven benchmarks. Through the executions they conclude that it is possible to achieve considerable energy and performance gains from the use of their approach, depending on the behavioral characteristics of the application. The technique consists in manual code instrumentation to trace the parallel regions in the source code. Otherwise, the focus of our work is on the automatic identification of these parallel regions, based on specific hardware counters values to find L2 misses rate and IPC by parallel region. Our approach use L2 misses rate because it allows for compiled results that accentuate the application's most Memory-Bound points, different from the L3 misses rate.

## IV. Methodology

Our approach allows investigating the program's behavior using Score-p instrumentation to collect the hardware counters using PAPI (Performance Application Programming Interface). We focus on OpenMP parallel programs. Figure 1 shows all the steps of our methodology.

The first step is to use the Score-p tool (with the OTF2 trace file format) to trace the hardware counters in a per-

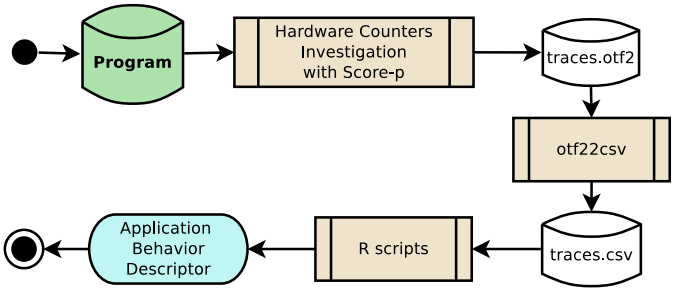


Fig. 1. The main steps of our methodology.

region fashion, where a region may be a parallel region with begin and end timestamps and the set of HW counters per thread. The hardware counters are those provided by PAPI tool, but we use only four hardware counters: PAPI_L2_TCA (Level 2 total cache accesses), PAPI_L2_TCM (Level 2 cache misses), PAPI_TOT_CYC (total cycles), and PAPI_TOT_INS (instructions completed). From this, it is possible to calculate the IPC (PAPI_TOT_INS divided by PAPI_TOT_CYC) and L2 misses rate (PAPI_L2_TCM divided by PAPI_L2_TCA).

The Akypuera `otf22csv` tool is used in a second step to convert the OFT2 file generated by execution with Score-p instrumentation to a CSV format. Because we use the Score-P tracing features, and not the profilling part, the granularity of the trace is a very detailed yet voluminous trace data per-thread and per-code region, enriched with HW counters metrics. To use the CSV information it is necessary to perform the grouping of all the metrics: thread, code region, timestamp and value of hardware counter. This grouping was performed using some statistical libraries of the R language. The CSV generated by the script accurately informs the behavior of the application, so it is possible to understand the most Memory-Bound points of the application, as well as Non-Memory-Bound.

From these previous steps, we intend to implement a library that uses the POMP2 library (from Opari2), so that this library automatically applies in Memory-Bound regions the use of an appropriated processor frequency according to previous steps. This library reads a configuration file, created by the analyst, to set the frequency appropriated to each code region in runtime.

In this work, we employ such methodology using the Lulesh application (using the params "-s 15 -i 100"). In the experiment we executed twice, the first to collect the amount of access and misses to l2 cache memory, the second to collect the number of CPU cycles and the total instructions. We use two executions to avoid the hardware counters saturation, so the first execution allows calculating the L2 misses rate and the other the calculate IPC in each region by thread. The execution platform used was the hype2, a Workstation with 2 processors Intel(R) Xeon(R) CPU E5-2650 2.30GHz, with 40 physical cores in total, and 126G of RAM memory.

## V. Results

The most important investigation is not the timestamp where the misses occur, but the code region that occurs the highest misses rate. In Figure 2 shows a subset of the parallel regions executed along time, in a case with 28 threads, for a very small time frame of ≈600 microseconds.

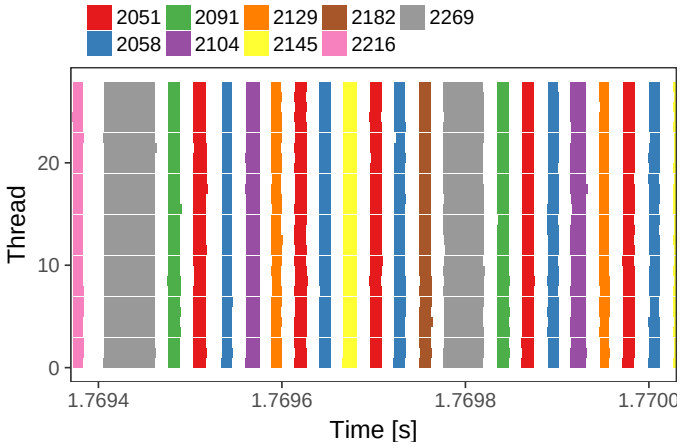


Fig. 2. Lulesh Behavior with a subset of nine parallel regions, identified by their starting line number in the source code.

Figure 2 shows nine regions (colors) per thread (on Y) as a function of time (on the X axis). These regions are identified by the starting line numbers that the tracer got from the unmodified source code of the application. We depict the behavior of 28 threads (represented by each row) and a temporal interval from 1.7694s to 1.7700s, just to show the richness of information we are able to collect through Score-P. One region might be executed multiple times depending on the application, as it is the case here for region identified by `2051` (the red color), which has been executed five times in this time frame.

For each of the 28 parallel regions of Lulesh, we evaluate their Memory-Bound behavior using per-region and per-thread HW-derived metrics. Figure 3 shows each of the regions (on X axis) for the two metrics: IPC (Instructions per Cycle, on top) and L2MR (L2 Miss Rate, on the bottom). These values are calculated by first taking the mean of the four measured metrics, then calculating the ratios that lead to IPC and L2MR. Only values for Thread 0 (the main thread) are considered, but all threads share the same HW counter behavior as we have observed. The highest L2MR is seem in regions `549` and `1171`, but only the latter has a low IPC. Further investigation is required to properly identify the true Memory-Bound regions, for example, by taking into account the average duration of the regions and eventually, other HW counters and derived metrics.

## VI. Conclusion

The main objective of this work is to analyze the behavior of the Lulesh application, analyzing all its parallel regions, specifically the regions that present Memory-Bound behavior. From the work it is possible to map each region, collecting hardware counters to calculate the L2MR (L2 Miss Rate) and the IPC (Instructions per Cycle) of each region during runtime.

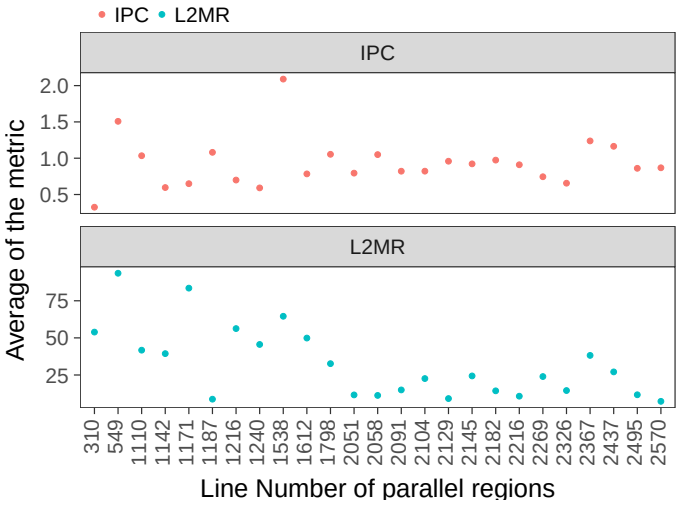


Fig. 3. IPC (Instructions per Cycle) and L2MR (L2 Miss Rate) for each of the 28 parallel regions of Lulesh that have been identified by Score-P.

The next step in this work is to implement a library that uses the results of the first analysis to properly set the appropriate processor frequencies in its Memory-Bound regions.

## VII. Acknowledgments

We thank FAPERGS/Inria ExaSE, FAPERGS GreenCloud, FAPERGS 16/2551-0000 354-8 (PPP 2014), CNPq 447311/2014-0, CNRS/LICIA Intl. Lab, the EU H2020 Programme and MCTI/RNP-Brazil under the HPC4E Project, grant 689772. We are also grateful for the masters scholarship and the HW resources provided by HPE under the HPCELO project. Some experiments were carried out at the Grid'5000 platform (https://www.grid5000.fr), with support from Inria, CNRS, RENATER and several other organizations.

## References

[1] C. Poellabauer, L. Singleton, and K. Schwan, "Feedback-Based Dynamic Voltage and Frequency Scaling for Memory-Bound Real-Time Applications," *In Proc. IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 234–243, 2005.

[2] V. W. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer, "Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster," *Proceedings - 19th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2005*, vol. 2005, pp. 164–173, 2005.

[3] "Reducing energy usage with memory and computation-aware dynamic frequency scaling," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6852 LNCS, no. PART 1, pp. 79–90, 2011.

[4] V. Spiliopoulos, A. Sembrant, and S. Kaxiras, "Power-sleuth: A tool for investigating your program's power behavior," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*. IEEE, 2012, pp. 241–250.

[5] L. F. Millani and L. M. Schnorr, "Computation-aware dynamic frequency scaling: Parsimonious evaluation of the time-energy trade-off using design of experiments," in *European Conference on Parallel Processing*. Springer, 2016, pp. 583–595.

[6] P. Pacheco, *An introduction to parallel programming*. Elsevier, 2011.

[7] G. Roth, J. Mellor-crummey, K. Kennedy, and R. G. Brickner, "Compiling Stencils in High Performance Fortran," no. November, 1997.

[8] C. Jesshope and C. Egan, *Advances in Computer Systems Architecture: 11th Asia-Pacific Conference, ACSAC 2006, Shanghai, China, September 6-8*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006.

[9] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 47, 2014.

[10] D. S. Silveira, G. B. Moro, E. H. da Cruz, P. O. Navaux, L. M. Schnorr, and S. Bampi, "Energy consumption estimation in parallel applications: an analysis in real and theoretical models," 2016.

[11] E. L. Seur and G. Heiser, "Dynamic Voltage and Frequency Scaling: the laws of diminishing returns," *Proceedings of the 2010 international conference on Power aware computing and systems*, pp. 1–8, 2010.

[12] T. I. Wiki, "DVFS User Guide," 2012. [Online]. Available: http://processors.wiki.ti.com/index.php/DVFS_User_Guide

[13] D. Brodowski and N. Golde, "CPU frequency and voltage scaling code in the Linux(TM) kernel," 2016. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[14] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed dvfs scheduling for scientific applications on power-aware clusters," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*. IEEE, 2005, pp. 34–34.