Measuring Hardware Counters for HPC Application Phase Detection

Gabriel Bronzatti Moro, Lucas Mello Schnorr
Institute of Informatics, Federal University of Rio Grande do Sul
Caixa Postal 15064 — CEP 91501-970 Porto Alegre – RS – Brazil

***Abstract*—Besides reducing the execution time of parallel applications, the power consumption is an increasingly addressed problem in High-Performance Computing. A parallel program can be divided into regions, which can have particular characteristics, for instance, a behavior more CPU or memory bound. This paper presents a preliminary effort to measure performance counters along time to enable the automatic detection of memory-bound parallel regions. Once attained, the automatic detection differs from the previous efforts since it does not require any code instrumentation, making it less intrusive. Three NAS benchmark parallel applications are used in the experiments: the Discrete 3D fast Fourier Transform (FT), the Lower-Upper Gauss-Seidel solver (LU), and the Conjugate Gradient (CG). As hardware counters, we measure the L2 and L3 cache miss rate along time using the likwid's timeline mode, a tool to measure hardware counters from the user space. The experiments enable us to identify possible memory-bound code regions by correlating with changes in cache miss rates. We intend to configure a suitable processor frequency for each memory-bound parallel region of the application, reducing the energy consumption of the application with minimal performance loss.**

I. Introduction

Large HPC applications are composed of many parallel regions that are executed by different threads. For example, in an application that simulates the heat exchange in a metal plate, we could define two parallel regions: the first to define the initial state of the plate and another part would be responsible for calculating the heat exchange in different points of the plate for a number of timesteps. Each parallel region has its own characteristic, some may be considered memory-bound, where there is a high rate of the cache miss, while others may be considered more CPU-bound, with a high instruction execution rate, others more IO-bound when the thread is limited by waiting for input or output operations. In the previous example of the heat plate, one could measure the hardware counters at a given frequency to define the main characteristic of each parallel region. An automatic detection of memory-bound parallel regions enable one to adjust the processor frequency, possibly reducing energy consumption with minor performance penalties in execution time.

The main objective of this study is to measure hardware counters at every given time interval to discover memory-bound code regions. Once these regions have been detected, we intend to apply Design of Experiments screening techniques [1] to find the best processor frequency configuration for each region, pretty similar to what has been done already [2], but automatically. This paper presents our preliminary results by showing a time-oriented method to collect hardware counters, using likwid's timeline mode [3]. The preliminary results indicate that the collected data can possibly enable the automated identification of memory-bound code regions.

This paper has the following organization. Section II presents related work regarding automatic phase detection for HPC applications. It also motivates our work. Section III details our proposal and its corresponding methodology to fullfill our goal. Section IV describes the platform we have been using to conduct experiments and the preliminary results we have obtained so far. Section V concludes the paper listing the main contributions and future work towards automatic detection.

II. Related Work

There is no definitive solution to detect if a code region is more memory or CPU bound. Some works focus more on phase detection to sequential applications [4][5]. Spiliopoulos et al.[4] present in his work a tool that analyzes the behavior of a sequential application by detailed analysis of its execution phases, based on cache misses of the different levels of cache. The identified phase may be comprised of a set of program functions, which are grouped by having a similar behavior. The identification of the behavior of these functions is performed in accordance with a prior history of execution, based on the trace generated by the application. The tool generated identifies the best processor frequency to be used in each phase to best performance and reduce energy consumption. This paper analyzes only sequential applications, in this perspective the identification of the memory-bounds regions areas can be obtained in a coarser granularity in the interval between samples timesteps. As for parallel applications running different flows may have different behaviors which may vary according to the application of load balancing, which causes the granularity of the samples is thinner for better understanding its behavior. In addition to this approach, Laurenzano et al.[5] present an approach finer granularity for identifying the most appropriate processor frequency for each application loop. Through multiple executions is set a model of various sizes, it allows to find for example the most appropriate setting for the given bond program, varying aspects that can influence the energy reduction and performance improvement for the application.

There is also investigation to consider MPI or OpenMP parallel applications. Freeh et al.[6] present an approach to define the most suitable frequency for each phase of an MPI

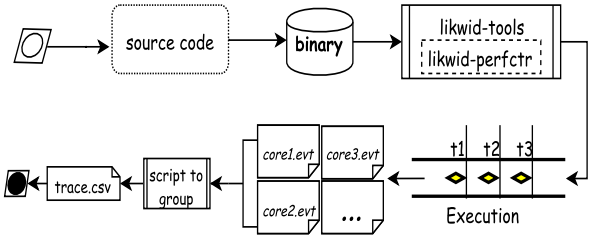
application. Among the available frequencies, the approach looks at what is the best frequency for a given node operate during the execution of the application. Another approach [2], focused in OpenMP applications, analyzes the parallel regions of a program using a rigorous evaluation using design of experiments and screening designs. According to their analysis based on seven benchmarks, it is possible to reach a considerable gain in energy reduction, eventually with no performance loss, depending on the characteristics of the benchmark. Their approach use manual instrumentation to identify the code regions that are going to be analzed. We focus instead in the automatic detection of such regions based on hardware counters. In this paper we show our investigation in how these counters can be measured along the application execution.

The next section describes our measurement and evaluation methodology to collect hardware counters at a given frequency.

III. Measurement and Evaluation Methodology

The methodology used in the work first defines the compilation a source code into binary. The program is run under the likwid-perfctr tool that allows you to collect hardware counters for each processing core. The data is processed by a script tailored to generate a detailed application trace to carry out the data analysis. The Figure III shows an overview of the methodology with all such steps.

Fig. 1. Overview of the methodology.



We employ such methodology in three NAS Parallel applications: the 3D Discrete Fast Fourier Transform (FT), the Lower-Upper Gauss-Seidel Solver (LU) and the Conjugate Gradient (CG). The applications are executed with 32 threads using the input size (class B) of the NAS benchmark. The execution platform used was the beagle1, a Workstation with 2 processors Intel (R) Xeon (R) E5-2650 CPU 2.00 GHz, each with 8 physical cores and Hyper-Threading technology.

The hardware counters are collected using likwid's timeline mode [3], configured to measure L2 and L3 cache miss rate at a given time interval. Such interval between each metric recording is defined according to the total execution time of each application. For example, in the FT application, the measurement period is defined as 30 milliseconds, generating about 172 samples (for each of the 32 threads). For the LU application a 100 milliseconds is adopted, for about 363 samples. For CG, a period of 50 milliseconds for about 384 samples. According to the likwid authors, adopting a period less than 100 milliseconds might generate non-valid results. Even so, the global behavior is still valid if one aggregate such information along time. We report FT and CG results under this limitation in order to investigate how frequent measurements impact our analysis of phase detection.

IV. Preliminary Results

We present the L2 and L3 cache miss rate considering the aggregated metrics for all cores of the two processors where we conducted experiments. Points in the plots represented such aggregated values, while lines are there only to show the metric trend along time. Despite the fact that we aggregated values, we have looked to each core cache level miss and they are all similar and homogeneous (because of the regular nature of the applications we used), justifying such aggregation to simplify the analysis.

A. Discrete 3D Fast Fourier Transform (NAS-FT, B Class)

Figure 2 depicts the L2 and L3 cache miss rate of the Discrete 3D Fast Fourier Transform (NAS-FT, B Class) when measuring metrics every 100 milliseconds. It clearly shows phases – represented by the peaks at regular intervals – when taking into account the L2 cache miss rate. For the L3 cache miss we observe that after the initialization phase (where a peak of 37% L3 miss rate is measured), the rate decreases towards zero with minor outliers. The highest L2 cache miss rate found for FT is about 30% between 7.5 to 10 seconds late time execution, while generally we can see that the observed phases reach a level of 30% in L2 misses. The lowest L2 miss rate is measured at 10% during the initialization phase.

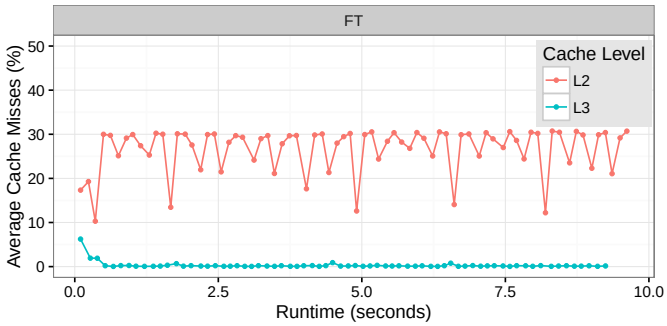


Fig. 2. The L2 and L3 cache miss rate of the Discrete 3D Fast Fourier Transform (NAS-FT, B Class) when measuring metrics every 100 milliseconds.

B. Lower-Upper Gauss-Seidel Solver (NAS-LU, B Class)

Figure 3 shows the L2 and L3 cache miss rate of the Lower-Upper Gauss-Seidel Solver (NAS-LU, B Class). Behavior is clearly different from the FT application, seen in Figure 2. We observe that the L2 cache miss rate fluctuates around 20%, while the L3 cache miss rate is about zero the whole execution,

except during the initialization phase, where a 13% rate is observed.

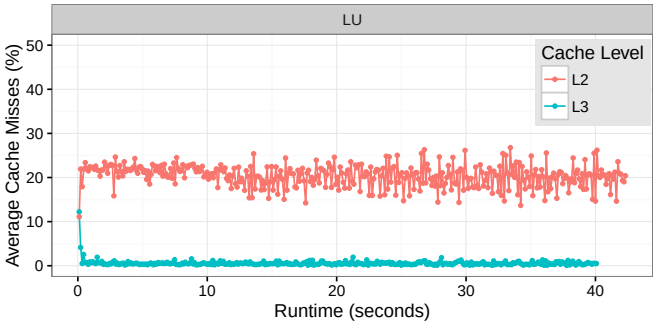


Fig. 3. The L2 and L3 cache miss rate of the Lower-Upper Gauss-Seidel Solver (NAS-LU, B Class) when measuring metrics every 100 milliseconds.

C. Conjugate Gradient (NAS-CG, B Class)

Figure 4 shows the L2 and L3 caches miss rate for Conjugate Gradient (NAS-CG, B Class) application. After the initialization phase, behavior of both metrics becomes stable at about 38% of misses for L2, and around zero for L3. Comparing against the previous applications, we see that the L2 miss rate for CG is greater than the others, suggesting that it is more memory-bound. This could potentially lead to a more gains in energy reduction if a proper processor frequency is selected.

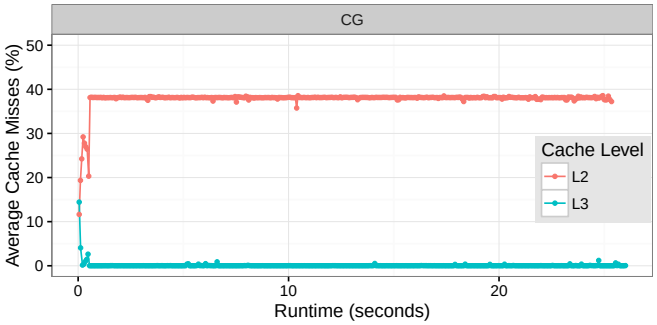


Fig. 4. The L2 and L3 cache miss rate of the Conjugate Gradient (NAS-CG, B Class) when measuring metrics every 50 milliseconds.

V. Conclusion

A parallel program has regions that depend on more on memory than others, one of the alternatives used to save energy is to use the technique Dynamic Voltage and Frequency Scaling (DVFS) to set up an appropriate frequency at a certain time of implementation of the program that can be run at low frequency, which can contribute to a reduction in power consumption of parallel application as a whole.

The main goal of this study is to measure parts more memory-bound of a parallel program, these regions can be defined as program parts that have a similar behavior in the cache misses rate for cache levels l3 and l2. The main contribution of this work is its methodology, which defines the steps and tools necessary that should be used to identify memory-bound portions of a parallel application.

As future work, we identify set the memory-bound parts of a parallel program based on the identified metrics, this work was an important step because was possible identified the fragments of the parallel application that have a high rate of cache misses for the cache L2 and L3. The automated detection of memory-bound regions would involve an analysis of the metrics of cache, thus could be defined the specific timestamp where starts the memory-bound region of the program and the time at which it ends.

Acknowledgements

This investigation receives funds from the HPC-ELO project under the HPE/UFRGS agreement, the H2020 program EU and MCTI / RNP-Brazil through HPC4E project with code 689772, the FAPERGS / Inria ExaSE design, universal design CNPq 447311 / 2014-0, and international CNRS / LICIA laboratory.

References

[1] R. Jain, *Art of Computer Systems Performance Analysis: Techniques For Experimental Design Measurements Simulation and Modeling*. Wiley, 1991.

[2] 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 *3rd International Workshop on Reproducibility in Parallel Computing (REPPAR)*, 2016.

[3] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *2010 39th International Conference on Parallel Processing Workshops*. IEEE, 2010, pp. 207–216.

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

[5] M. A. Laurenzano, M. Meswani, L. Carrington, A. Snavely, M. M. Tikir, and S. Poole, "Reducing energy usage with memory and computation-aware dynamic frequency scaling," in *European Conference on Parallel Processing*. Springer, 2011, pp. 79–90.

[6] 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," in *19th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2005, pp. 4a–4a.