2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)

Memory Performance and Bottlenecks in Multicore and GPU Architectures

Matheus S. Serpa, Francis B. Moreira
and Philippe O. A. Navaux
Informatics Institute
Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
{msserpa, fbmoreira, navaux}@inf.ufrgs.br

Eduardo H. M. Cruz
Federal Institute of Paraná (IFPR)
Paranavaí, Brazil
eduardo.cruz@ifpr.edu.br

Matthias Diener
University of Illinois at Urbana-Champaign
Urbana, USA
mdiener@illinois.edu

Dalvan Griebler and Luiz Gustavo Fernandes
School of Technology
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, Brazil
dalvan.griebler@acad.pucrs.br, luiz.fernandes@pucrs.br

***Abstract*—Nowadays, there are several different architectures available not only for the industry, but also for normal consumers. Traditional multicore processors, GPUs, accelerators such as the Sunway SW26010, or even energy efficiency-driven processors such as the ARM family, present very different architectural characteristics. This wide range of characteristics presents a challenge for the developers of applications. Developers must deal with different instruction sets, memory hierarchies, or even different programming paradigms when programming for these architectures. Therefore, the same application can perform well when executing on one architecture, but poorly on another architecture. To optimize an application, it is important to have a deep understanding of how it behaves on different architectures. The related work in this area mostly focuses on a limited analysis encompassing execution time and energy. In this paper, we perform a detailed investigation on the impact of the memory subsystem of different architectures, which is one of the most important aspects to be considered. For this study, we performed experiments in the Broadwell CPU and Pascal GPU, using applications from the Rodinia benchmark suite. In this way, we were able to understand why an application performs well on one architecture and poorly on others.**

***Index Terms*—Performance evaluation; Manycore systems; Memory subsystem; Cache memory; HPC**

I. Introduction

Traditional *multicore* architectures currently have dozens of cores, as well as a complex memory hierarchy to alleviate memory latency for all threads. These architectures rely on both instruction level parallelism (ILP) and thread level parallelism (TLP) to achieve high performance, as in the Intel Xeon Broadwell architecture [1]. Another type of architecture, which introduces different concepts, is the *GPU*. GPUs have thousands of simple cores, which alone are less powerful than the multicore ones, such that the GPU performance depends mostly on TLP. Although GPUs can achieve more performance than multicore architectures, they have some drawbacks: fewer types of applications can be executed efficiently on GPU, and the parallel programming APIs available for GPUs, such as CUDA [2], [3], present added complexity to the developer.

The usage of parallel and heterogeneous architectures poses several challenges for high-performance computing [4]. Applications need to be coded considering the particularities and constraints of each environment, as well as considering their distinct architectural characteristics [5]. For example, in the memory hierarchy, the presence of several cache memory levels, some shared and others private, introduces non-uniform access times, which impact applications' performance [6]. It is even more critical in heterogeneous architectures since each accelerator can have its own, distinct, memory hierarchy. Also, in heterogeneous architectures, the number of functional units may vary between different accelerators, and the instruction set itself may not be the same. In this context, it is important to analyze the performance and behavior of parallel and heterogeneous architectures, in order to provide better support for developers, so they can optimize their applications for the target system.

This work aims to perform a detailed analysis of the impact of the memory subsystem used in different architectures. We used hardware performance counters to gather accurate measurements of the actual impact of different factors that influence the memory access. We analyzed counters that measure the cache memory usage, main memory, interconnection traffic, among others. By doing so, we obtained a detailed understanding of how different aspects of the memory hierarchy impact the performance of applications. Such a study can serve as a basis for developers of parallel applications to optimize their applications.

This article is organized as follows. Section II contains the description of the experimental methodology. Section III shows the results and analysis. Section IV presents a summary of the related work in the area. Finally, Section V presents

This research received funding from Coordenao de Aperfeioamento de Pessoal de Nivel Superior - Brasil (CAPES) - Finance Code 001, FAPERGS 01/2017-ARD project ParaElastic (No. 17/2551-0000871-5), School of Technology from PUCRS and Petrobras under project 2016/00133-9.

2377-5750/19/$31.00 ©2019 IEEE
DOI 10.1109/PDP.2019.00041

concluding remarks and future work.

II. Methodology

The experiments were performed in the Broadwell and Pascal system environments. The Broadwell system is composed of two Intel Xeon E5-2699 v4 processors, where each processor consists of 22 physical cores, allowing execution of 88 threads with Hyper-Threading. The Pascal is a Tesla P100 GPU, where we used one of the GPUs with 3584 CUDA cores.

The benchmark suite Rodinia was chosen since it implements a set of applications with distinct parallel execution characteristics and contains implementations for multicore processors and GPU boards. For Broadwell we used the OpenMP implementation of Rodinia. On Pascal, we used the CUDA implementation.

The experiments shown in Section III present the average of 30 random executions. The standard deviation presented is given by the t-Student distribution with a 95% confidence interval. Moreover, we also investigate other metrics such as rate of core usage, bandwidth and cache hit ratio in the memory subsystem. The Intel PCM [7] and Intel VTune [8] tools were used to obtain data for the Broadwell executions, while for the Pascal GPU, we used the nvprof [9] tool.

III. Experiments

The performance of each application depends on the architecture. This motivates the study of applications and architectural characteristics, aimed to understand why a benchmark performs better in one architecture and how to improve its performance. Thus, the following subsections present performance analyses of *Rodinia* benchmarks. The benchmarks were executed in the Broadwell and Pascal architectures, in order to identify the distinct memory hierarchy characteristics that impact the performance of parallel applications.

A. Broadwell (CPU)

The Broadwell architecture obtains data by accessing the private L1 and L2 caches of each core, the L3 cache shared by all cores of the same processor, or the main memory. Figure 1 shows the L1 and L2 hit ratios. Figure 2 shows the L3 and combined hit rate, which is the amount of memory transactions serviced by any cache. The x-axis presents the name of each application ordered by the IPC (smaller shown first), and the y-axis indicates the hit ratio in percentage. The L1 cache is the fastest and smallest memory and is private to each core. The average L1 hit ratio was 96.1%. Even with an average close to 100%, small variations in the L1 hit ratio implies in significant variations in performance, and therefore it is not a good predictor of IPC by itself. Applications such as *lavaMD*, *nw*, *nn*, *hotspot3D* and *bfs* have L1 hit rates up to 99.9% due to their data accesses pattern. On the other hand, applications such as *backprop*, *srad*, and *particleFilter*, have lower hit rates (86.4% in *particleFilter*), affecting their performance because they access slower memories more frequently.

The L2 cache is also private to each core. *nn*, *leukocyte*, *heartwall* and *particleFilter* are applications that take advantage of the cache and thus have better performance results.

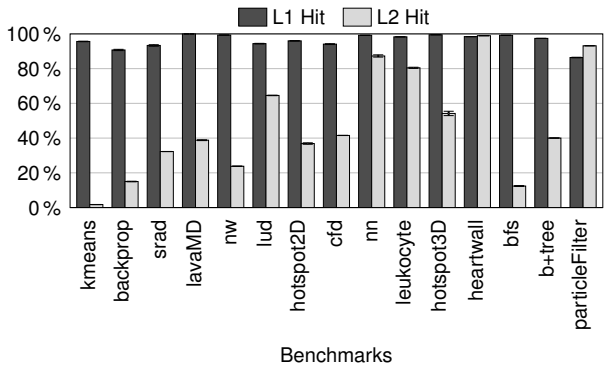

Fig. 1. L1 and L2 Cache hit ratio - Broadwell.

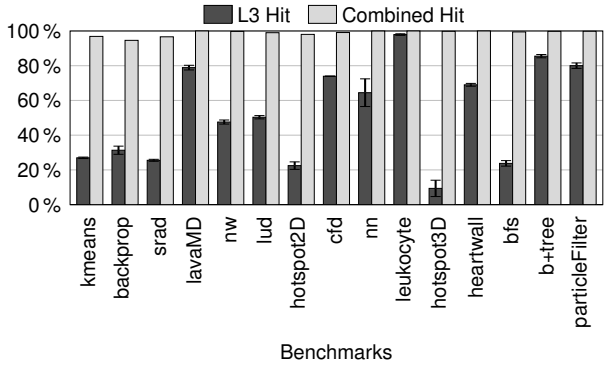

Fig. 2. L3 and Combined hit ratio - Broadwell.

The L2 hit ratios in these applications were up to 98.9%. The L3 cache, the last-level cache in the Broadwell architecture, is shared between all cores of the same processor. It helps several applications by reducing the accesses in the main memory. The applications *leukocyte*, *b+tree* and *particleFilter* have a L3 hit ratio greater than 80%, which means that almost their entire data fit in the L3. On the other hand, *hotspot3D* has a L3 hit equals to 9.4%, which is low compared to the average (52.5%), and yet the application performs above average when compared to the other applications. Most likely, the majority of its accesses were already filtered by the higher level caches.

We can observe, in the combined hit ratio, that the applications with the highest hit ratios are the ones with the highest IPCs, which clearly indicates that the cache memories have a big impact in the performance. It shows that any cache memory servicing a transaction is more important than simply having a hit in an upper level, lower latency cache. For instance, *particleFilter* has the best IPC even without the highest L1 hit ratio, but it has a very high combined hit ratio. To improve the performance, techniques such as loop interchange and loop tiling can be used. Using these techniques, more data is fetched to the cache memories, the data reuse in the caches increase, and cache line prefetchers can fetch data from the main memory more accurately.

Figures 3 and 4 show results of dynamic random-access

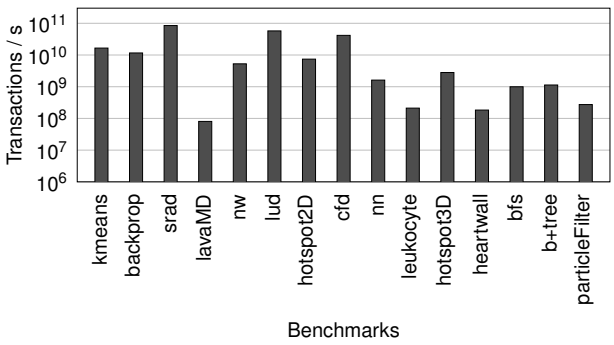

Fig. 3. DRAM transactions - Broadwell.

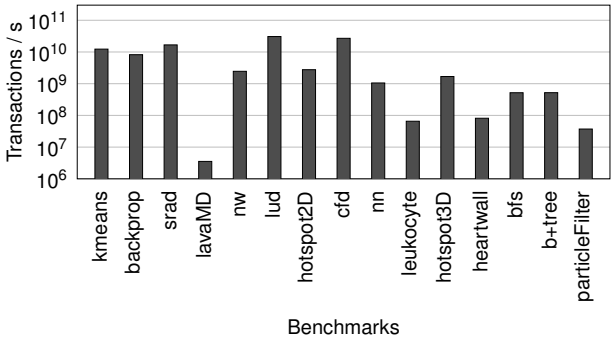

Fig. 4. Transactions across the QPI - Broadwell.

memory (DRAM) transactions per second and Quickpath Interconnect (QPI) transactions per second. Applications with low cache hit ratios have a large number of DRAM and QPI transactions, which reduce their performance by accesses to local and remote memories. The DRAM transactions per second was in average 15.5 GT/s, which is 5.5 times less than the maximum value of 84.9 GT/s. The applications *srad*, *lud* and *cfd* have the highest values of transactions per second, which limit their performance. Transactions across the QPI also reduce the performance of applications.

If we divide the applications in Figures 3 and 4 in the half, we can observe the applications on the left side have, on average, more DRAM and QPI transactions than the applications on the right side. Since the applications are shown ordered by their IPC, this indicates a tendency for applications that need less DRAM or QPI accesses to have better performance. We can also observe that applications with less cache hits (Figure 2) tend to have more DRAM and QPI traffic.

B. Pascal (GPU)

Section III-A discussed the performance of Broadwell memory subsystem. The GPU memory hierarchy has different design characteristics than this. Broadwell has large caches aimed at reducing the long latency of the main memory to smaller latencies of cache levels closer to the processor. GPUs target throughput, allowing the load of multiple operands per unit of time. Pascal architecture offer memories with different

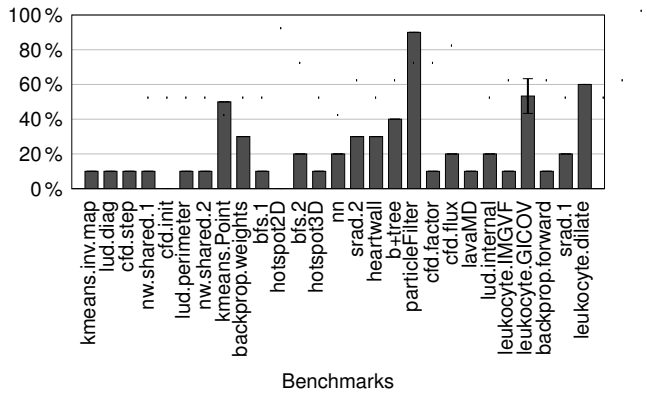

Fig. 5. The utilization level of the L1/texture cache relative to the theoretical peak utilization - Pascal.

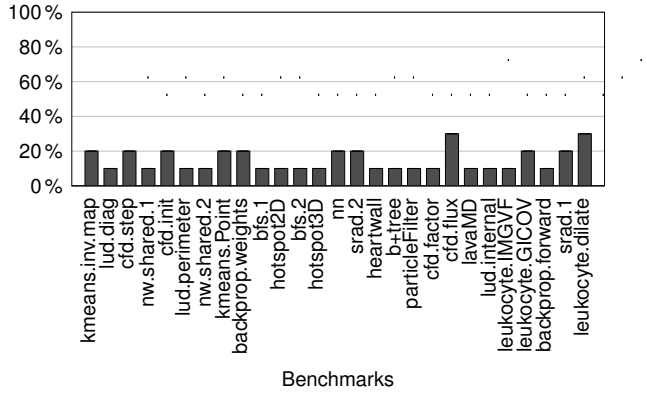

Fig. 6. The utilization level of the L2 cache relative to the theoretical peak utilization - Pascal.

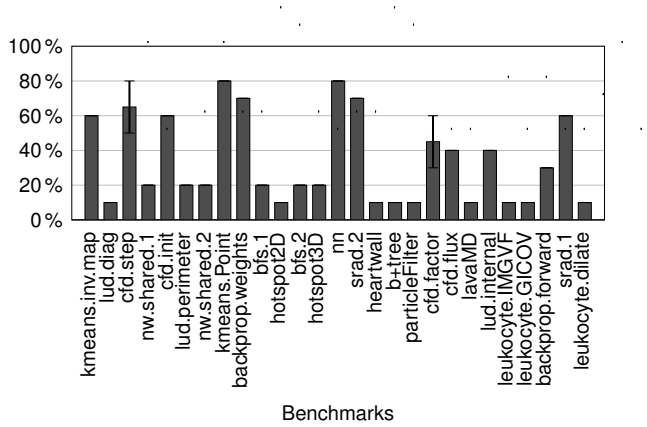

Fig. 7. The utilization level of the device memory relative to the theoretical peak utilization - Pascal.

characteristics. L1/texture (read-only) and shared memories are the fastest and smallest ones. Both can be used automatically or handled by the programmer. The texture memory is a read-only memory used to reuse data from the stack or the global memory. The L1 can be used to keep data of stack and

register spill.

Figure 5 shows the utilization of these memories relative to the theoretical peak usage. The utilization rate is low in both memories, with an of average 22.4% of the peak utilization. In the case of L1/texture, the max utilization level was 60.0% in the *leukocyte.dilate* kernel, which uses the L1/texture memory and also has the high IPC. There are ways to improve the usage of these memories and therefore applications performance, either by using the CUDA intrinsic *lgd()*, which indicates that the data should be stored in the texture memory.

The L2 cache and the device memory are larger but have lower speeds than shared and L1/texture. Figures 6 and 7 show the use of these memories. Another difference between Broadwell and Pascal memory subsystem is that, in Pascal, the global memory data accessed is first searched in L2 cache instead L1 cache. The average utilization of L2 was 14.8%. This means that most applications do not use the L2 cache. Applications such as *cfd* and *eukocyte* have a utilization of 30%, which reduces its accesses to the slow global memory. The average utilization of global memory was 33.7%, explaining why the L2 utilization is also low.

IV. Related Work

A memory model to analyze algorithms for many-core systems is presented in [10]. Analyzing their study, we observe that it is essential to have a thorough understanding of the behavior of the applications because the performance alone does not allow us to comprehend the bottlenecks of an application or architecture. Mei et al. [11] analyzed the characteristics of the memory subsystem in 3 different GPU architectures: Fermi, Kepler, and Maxwell. The authors conclude that the Kepler architecture planning was aggressive in its memory bandwidth, which has often been underused, and that, in the Maxwell architecture, more resources were invested in shared memory, generating a more efficient and balanced system.

Serpa et al. [12] study the impact of thread and data strategies, revealing that, with smart mapping policies, one can indeed significantly improve memory performance on multicore architectures. Satish et al. [13] analyze the performance gap between simple and highly optimized code in modern multicore and manycore architectures. They propose optimizations to the source code that can make better use of the memory hierarchy and the vectorization instructions.

The related work demonstrates that, by taking into account the different features of each architecture, developers can vastly improve an application's performance. Our work goes beyond a scalability analysis and looks for a greater understanding of the impact of different applications on different systems, measured from hardware counters. The analysis of the related work demonstrates how in-depth knowledge of the application behavior at the architectural level allows developing techniques to gain performance. In this context, the focus of this article is to study the effects of different memory subsystems of CPU and GPU architectures in parallel applications, and thus to analyze the impact that each memory subsystem characteristic has on the applications.

V. Conclusions and Future Work

The wide range of architectures presents a challenge for developers when optimizing the applications due to their different characteristics. This work analyzed the impact of one of the most important characteristics: the memory subsystem. The Broadwell and Pascal architectures were evaluated. Related work in this area focus solely in the performance or energy consumption, while our work perform a deeper analysis of the behavior of the memory hierarchy. For this, hardware counters were employed to measure statistics such as cache misses, main memory transactions, prefetch accuracy, among others.

The results show that, in the architectures, the main performance limit are the accesses to main memory. In Pascal architecture, the overall memory utilization rate is directly linked to application performance. The best case is that there is little use of global memory and much utilization of the *cache* memories. In the Broadwell architecture, applications with high hit rates in the cache have the best performance. More specifically, results show that it is more important to have a high cache hit ratio in any cache level than simply having a high cache hit rate in upper levels such as L1.

Future work includes a study of the energy efficiency of different architectures.

References

[1] A. Nalamalpu, N. Kurd, A. Deval, C. Mozak, J. Douglas, A. Khanna, F. Paillet, G. Schrom, and B. Phelps, "Broadwell: A family of ia 14nm processors," in *VLSI Circuits (VLSI Circuits), 2015 Symposium on*. IEEE, 2015, pp. C314–C315.

[2] S. Cook, *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes, 2012.

[3] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[4] S. Mittal and J. S. Vetter, "A survey of cpu-gpu heterogeneous computing techniques," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 69:1–69:35, Jul. 2015.

[5] W. Gropp and M. Snir, "Programming for exascale computers," *Computing in Science Engineering*, vol. 15, no. 6, pp. 27–35, 2013.

[6] E. H. Cruz, M. Diener, M. A. Alves, L. L. Pilla, and P. O. Navaux, "Lapt: A locality-aware page table for thread and data mapping," *Parallel Computing (PARCO)*, vol. 54, pp. 59 – 71, 2016.

[7] Intel, "Intel Performance Counter Monitor - A better way to measure CPU utilization," 2012. [Online]. Available: http://www.intel.com/software/pcm

[8] Intel, "Intel VTune Amplifier XE 2016," 2016.

[9] Nvidia, "Developer Zone - CUDA Toolkit Documentation," 2016. [Online]. Available: http://docs.nvidia.com/cuda/profiler-users-guide/

[10] L. Ma, K. Agrawal, and R. D. Chamberlain, "A memory access model for highly-threaded many-core architectures," *Future Generation Computer Systems*, vol. 30, pp. 202 – 215, 2014, special Issue on Extreme Scale Parallel Architectures and Systems, Cryptography in Cloud Computing and Recent Advances in Parallel and Distributed Systems, {ICPADS} 2012 Selected Papers.

[11] X. Mei and X. Chu, "Dissecting GPU memory hierarchy through microbenchmarking," *CoRR*, vol. abs/1509.02308, 2015.

[12] M. S. Serpa, A. M. Krause, E. H. M. Cruz, P. O. A. Navaux, M. Pasin, and P. Felber, "Optimizing machine learning algorithms on multi-core and many-core architectures using thread and data mapping," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018, pp. 329–333.

[13] N. Satish, C. Kim, J. Chhugani, H. Saito, R. Krishnaiyer, M. Smelyanskiy, M. Girkar, and P. Dubey, "Can traditional programming bridge the ninja performance gap for parallel computing applications?" in *International Symposium on Computer Architecture (ISCA)*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 440–451.