Integrating Memory Perspective into the BSC Performance Tools

Harald Servat*, Jesús Labarta†‡, Hans-Christian Hoppe*, Judit Giménez†‡ and Antonio J. Peña†
*Intel Corporation
†Barcelona Supercomputing Center (BSC)
‡Universitat Politècnica de Catalunya (UPC)

***Abstract*—The growing gap between processor and memory speeds results in complex memory hierarchies as processors evolve to mitigate such differences by taking advantage of locality of reference. In this direction, the BSC performance analysis tools have been recently extended to provide insight relative the application memory accesses depicting their temporal and spatial characteristics, correlating with the source-code and the achieved performance simultaneously. These extensions rely on the Precise Event-Based Sampling (PEBS) mechanism available in recent Intel processors to capture information relative to the application memory accesses. The sampled information is processed with the Folding mechanism to provide a detailed temporal evolution of the memory accesses and in conjunction with the achieved performance and the source-code counterpart. The results obtained from the combination of these tools help application developers to understand better how the application behaves and how the system performs. We demonstrate the value of the complete work-flow by exploring an already optimized state-of-the-art benchmark, providing detailed insight of their memory access behavior.**

***Index Terms*—performance analysis, memory references, sampling, instrumentation**

I. Introduction

The growing gap between processor and memory speeds leads to complex memory hierarchies in current processors. The memory hierarchy is organized in different *strata* to better exploit the applications' temporal and spatial localities of reference. On one end of the hierarchy lie extremely fast, tiny and power-hungry registers while on the other end there is the slow, huge and less energy-consuming DRAM. In between these two extremes, there are multiple cache levels that mitigate the expense of bringing data from the DRAM when the application exposes either spatial or temporal locality.

Some performance tools, such as HPCToolkit [1], dmem_advisor [2] and Intel® VTune™Amplifier [3] provide the most referenced variables or the highest latency accesses to help the user focusing when shortening the time—to—solution. However, understanding the memory access patterns not only help on this direction but also may offer additional insights to improve the execution behavior by helping prefetch mechanisms, calculating reuse distances, tuning cache organization and envision the usage of hybrid memory systems.

Detailed memory-access analysis has typically relied on low-level instrumentation [4], [5], [6] with the consequent performance overhead. The Folding mechanism [7] belongs to the BSC performance analysis tool suite and demonstrates that this analysis can rely on coarse-grain sampling and minimal instrumentation and allows users exploring in-production binaries. We have integrated tightly memory sampling mechanisms into BSC tools to help on the exploration of the application performance, its progression on code regions and their access to the address space.

The organization of this paper is as follows. Section II summarizes the extensions applied to the BSC performance tool-suite to cover the memory accesses. Section III follows with an exhaustive performance and memory access analyses of a well-known benchmark. Finally, Section IV draws some conclusions.

II. Extensions to the BSC performance tools

We have extended the BSC performance tool suite in two directions. First, in the context of the monitoring tool (Extrae), we take advantage of the PEBS infrastructure built in recent Intel® Xeon® and Intel® Xeon Phi™processors to sample memory operations and capture information such as the address referenced, its access cost and which part of the memory hierarchy provided the data. To help the analyst correlating addresses with the application, Extrae also captures the application data objects by instrumenting dynamic memory allocations (such as `malloc` and `realloc`) in addition to exploring the binary for static data objects. This information is used to match the sampled references into data objects which are identified by their call-stack (dynamically-allocated) or by their given name (statically-allocated). The integration also allows capturing load and store references (if hardware permits) by using Extrae's multiplexing capabilities, and thus avoiding the need to run the application twice.

Second, in the context of the Folding mechanism, the tool provides a report where applications are explored in three orthogonal directions: source code, memory accesses and performance. This allows understanding the application data locality exposed on different code regions while knowing which is the achieved performance. The multiplexing capabilities on the monitoring side also avoids having to explore two independent reports with randomized address spaces[1].

[1]Due to ASLR - Address Space Layout Randomization

III. Application evaluation

We have evaluated HPCG (High Performance Conjugate Gradient) on the Jureca system [8]. The code benchmarks computer systems based on an additive Schwarz, symmetric Gauss-Seidel preconditioned conjugate gradient solver [9]. We have compiled the application using the Intel compiler suite with the `-O3 -xavx -g` compilation flags. We have executed the benchmark using the 24 cores of a node using a problem size $nx{=}ny{=}nz{=}104$. The analysis focuses on the execution phase, ignoring the initialization and finalization.

In a preliminary analysis of the application, most of the PEBS references were not associated to a memory object. This occurs because the application allocates its data using many consecutive allocations below the threshold (100s of bytes). The data objects are allocated using two different mechanisms in lines 108-110 and 143 within the file `GenerateProblem_ref.cpp`, respectively. The first set of objects are allocated through the `new` C++ language construct while the second set are allocated through the `[]`-operator of the C++ `map` structures. To avoid creating huge event trace-files, we grouped these allocations in two groups by manually wrapping the first and last addresses of each group of allocations using instrumentation capabilities.

Figure 1 shows the result obtained when applied to the modified version of HPCG. We notice that each iteration consists of two calls to `ComputeSYMGS_ref` (labels A and D) and `ComputeSPMV_ref` (B and E) and in between there is a call to `ComputeMG_ref` (C). We identify linear accesses in the higher and lower part of the address space. More precisely, regions A and D present a phase (`a1` and `d1` in blue) that accesses the address space from lower to upper addresses (forward sweep) followed by a phase (`a2` and `d2` also in blue) that accesses the address space from upper to lower addresses (backward sweep). It is worth to note that there are no stores (*i.e.* black points) in the lower part of the address space in the execution phase, because data has been written in the setup phase.

From the performance perspective, the code does not exceed 1500 MIPS representing an IPC of 0.6 considering the nominal frequency except for the transitions between phases where the performance shows a slight increase due to a reduction of the cache misses. Although the instruction rate does not significantly increase when the application progresses from forward sweep to backward sweep there are performance differences when executing these regions. Since the results shown in Figure 1 indicate that `a1` and `a2` traverses the whole data structure, the approximations for the memory bandwidth while traversing the structure are 4197 MB/s and 4315 MB/s, respectively. In comparison, the observed bandwidth while traversing the same structure in region `B` achieves 6427 MB/s.

IV. Conclusions

The PEBS hardware infrastructure assists with sampling memory-related instructions and gathering valuable details about the application behavior. The usage of PEBS in the BSC tools results in thorough memory access patterns exploration

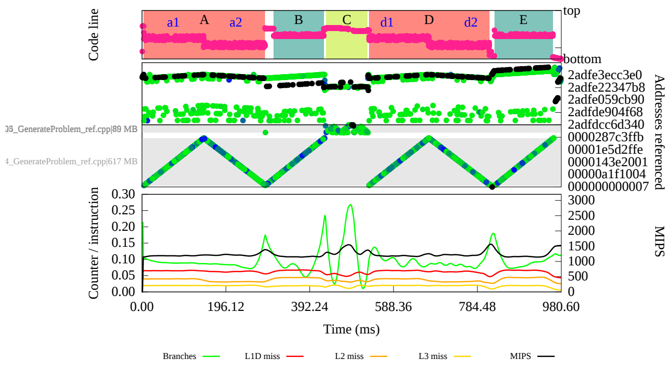

Fig. 1: Analysis of HPCG 3.0.

on a state-of-the-art benchmark without having to use high-frequency sampling and thus not incurring on large overheads. The exploration included scan of the memory access patterns from a time perspective and the identification of the most dominant data streams and their temporal evolution along computing regions.

In particular, the HPCG results show that the main routine traverses the address space two times (first forward then backward); also a part of the address space is not modified. HPCG also shows difference performance values for forward and backward sweeps not only in cache miss ratios but also in the cost of providing data from memory. Additionally, the fact that a portion of the address space is only read during the execution phase means that this region might benefit from memory technologies where loads are faster than stores.

Acknowledgments

This work has been performed in the Intel-BSC Exascale Lab. We would like to thank Forschungszentrum Jülich for the compute time on the Jureca system. Antonio J. Peña is cofinanced by the Spanish Ministry of Economy and Competitiveness under Juan de la Cierva fellowship number IJCI-2015-23266.

References

[1] X. Liu and J. M. Mellor-Crummey, "A data-centric profiler for parallel programs," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13*, 2,013, pp. 28:1–28:12.

[2] A. J. Peña and P. Balaji, "Toward the efficient use of multiple explicitly managed memory subsystems," in *IEEE International Conference on Cluster Computing (CLUSTER)*, 2014, pp. 123–131.

[3] "Intel VTune Amplifier," last accessed May 2017. [Online]. Available: https://software.intel.com/en-us/intel-vtune-amplifier-xe

[4] K. Beyls and E. D'Hollander, "Refactoring for data locality," *Computer*, vol. 42, no. 2, pp. 62–71, 2,009.

[5] V. Subotic *et al.*, "Quantifying the potential task-based dataflow parallelism in MPI applications," in *Euro-Par*, 2011, pp. 39–51.

[6] A. J. Peña and P. Balaji, "A framework for tracking memory accesses in scientific applications," in *43rd International Conference on Parallel Processing Workshops (ICCPW)*. IEEE, 2014, pp. 235–244.

[7] H. Servat *et al.*, "Unveiling internal evolution of parallel application computation phases," in *Int. Conf. on Parallel Processing (ICPP)*, 2011.

[8] "Jureca system architecture," http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html - last accessed, May 2017.

[9] J. Dongarra, M. A. Heroux, and P. Luszczek, "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems," *IJHPCA*, vol. 30, no. 1, pp. 3–10, 2016.