Memory Hierarchy Performance Characterization of SPEC CPU2017

Agustín Navarro-Torres*,1,
Pablo Ibáñez-Marín*,1,
Jesús Alastruey-Benedé*,1,
Víctor Viñals-Yúfera*,1

* *Universidad de Zaragoza, I3A, Spain*

ABSTRACT

SPEC CPU is one of the most common suite of benchmarks used in computer architecture research. On June 2017, a new version was released to replace the old version, CPU2006, which remained state-of-the-art for 11 years. We present a detailed memory hierarchy performance analysis of all single-thread benchmarks that made up the new suite.

KEYWORDS: SPEC CPU2017; characterization; memory hierarchy; performance analysis; benchmarks; hardware counters

1 Introduction

SPEC CPU is one of the most widely used benchmark suites for high performance computing research on academia and industry. The new version, CPU2017 [1], released on June 2017 is called to replace the 2006 version. Thus, new characterization is necessary in order to help researchers to select the benchmarks with particular characteristics or pick simulation points.

In this work we analyze the memory hierarchy performance of SPEC CPU2017. Namely, we study all the single-thread benchmarks, identify the memory-intensive ones, and analyze the sensitivity of them to the last-level cache size and the different hardware prefetchers.

2 Experimental methodology

We execute all the SPEC CPU2017 single-thread benchmarks in an Intel Xeon Skylake-SP Gold 5120 (Xeon-SP in short). Several hardware performance counters were collected with the perf profiler[4] to measure the following metrics: cycles per instruction (CPI), misses per kilo instruction (MPKI) in all cache levels and bytes read from main memory per kilo instruction (BPKI).

[1]E-mail: {agusnt, imarin, jalastru, victor}@unizar.es

Our Xeon-SP has 14 cores, two levels of private cache (32 KB for instructions and 32 KB for data in the first level, and 1 MB in the second level) and a shared last-level cache (LLC, 19.25 MB, 11-way set-associative). The processor disposes of four hardware prefetchers [3]: *L1 Data cache unit prefetcher* (DCUI), *L1 Data cache instruction pointer stride prefetcher* (DCUP), *L2 Data cache spatial prefetcher* (L2A), and *L2 Data cache streamer* (L2P).

We use the *Intel Cache Allocation Technology* (CAT) feature to modify the LLC capacity that a program has available during its execution. Hardware prefetchers are selectively enabled or disabled by writing their corresponding *Model Specific Registers* (MSRs). This way we can evaluate the application performance with multiples LLC configurations on a real system.

3 Evaluation

3.1 Identification of Memory Intensive Benchmarks

We executed all CPU2017 single-thread applications with all the inputs provided by SPEC in a limited resources context: all hardware prefetchers disabled and the smallest LLC size that can be assigned to an application (1.75 MB). For each application/input pair we measured misses per kilo instructions for the three cache levels (MPKI1, MPKI2 and MPKI3). These metrics are shown in Figure 1. We plot in red the bars associated to applications that have very low MPKI2 and MPKI3 ratio, even in a limited resources context. These application/input pairs have little interest for memory hierarchy studies. Among the remaining application/input pairs, we have selected an input for each application, plotted in green, that will be used in the following experiments.

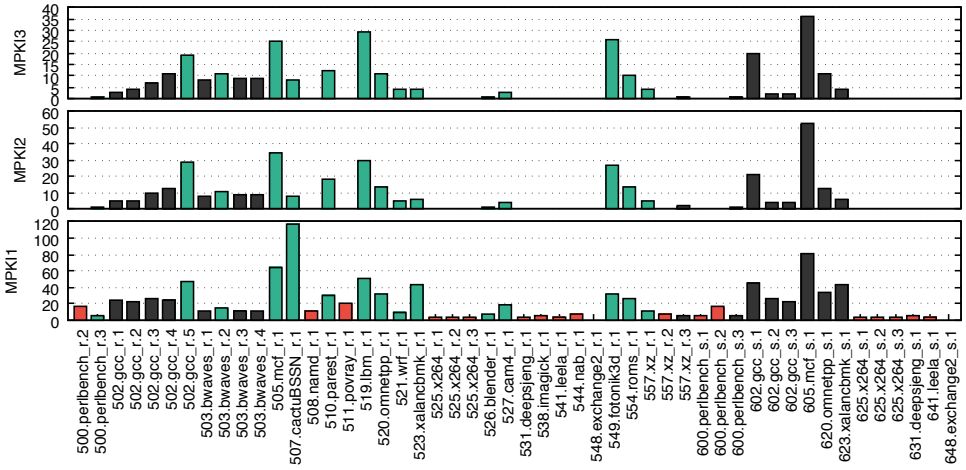

Figure 1: MPKI1, MPKI2 and MPKI3 for all SPEC CPU2017 single-thread benchmarks.

3.2 Sensitivity to the LLC Size and to Hardware Prefetching

In this experiment we study the sensitivity of the memory-intensive benchmarks to the LLC size and to the hardware prefetcher. The 15 workloads selected in the previous subsection were executed with different LLC sizes. All these runs were performed in two different ways: one with all the prefetchers enabled and the other with all disabled.

Five LLC sizes, 19.25 MB, 14 MB, 7 MB, 3.5 MB and 1.75 MB, were configured by limiting the number of ways available for the program to 11, 8, 4, 2 and 1, respectively. This resource allocation was performed by CAT. Figure 2 shows the MPKI3 for the selected LLC sizes and benchmarks.

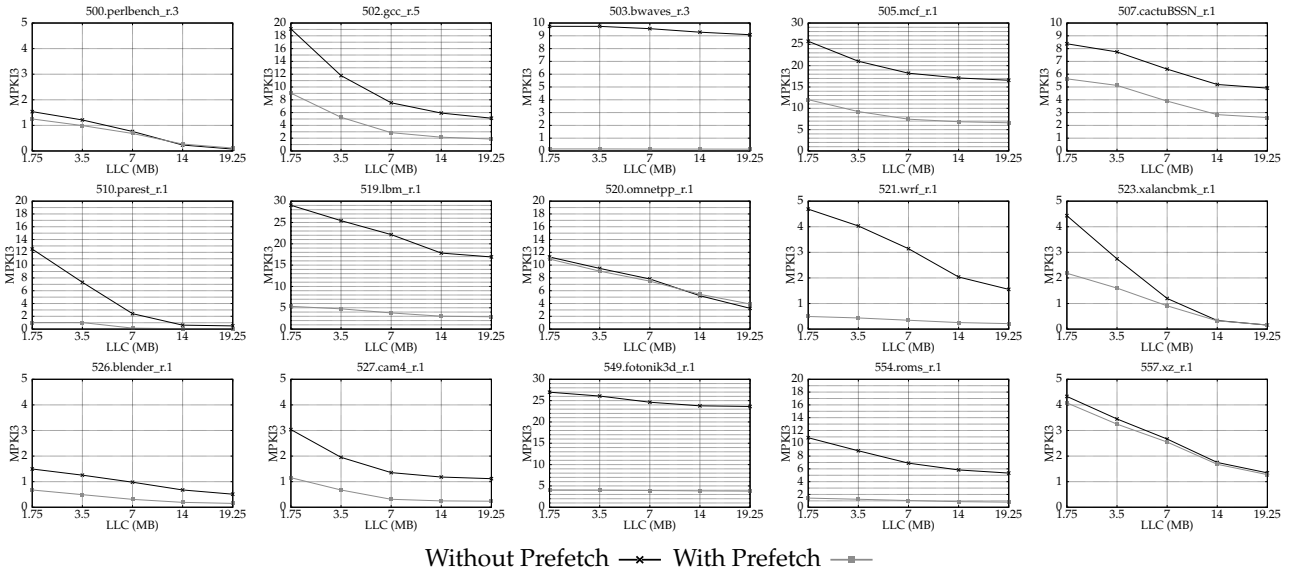

Figure 2: LLC cache misses (MPKI3) of the selected application for different LLC sizes.

Without prefetching, an increase in the LLC size translates to a significant MPKI3 reduction in all applications except `503.bwaves`. This improvement decreases considerably when prefetching is enabled for several applications: `510.parest`, `519.lbm`, `521.wrf`, `549.fotonik3d` and `554.roms`.

Prefetching is very effective in terms of MPKI3 reduction for 12 benchmarks: it reduces MPKI3 in all cache sizes, specially in the smaller ones. For two benhmarks (`500.perlbench` and `557.xz`), it gets minor improvements only for small LLC sizes. And finally, for one application (`520.omnetpp`), prefetching does not reduce LLC cache misses in any case, and even slightly increases the MPKI3 with the largest LLC.

3.3 Performance of the Hardware Prefetchers

In this section, we analyze the impact of the different hardware prefetchers on the applications performance and on the applications memory bandwidth consumption. All the selected workloads were executed with different configurations: with all prefetchers enabled, with all of them disabled, and with each one of the prefetchers enabled individually (one at a time). The experiment was accomplished with the maximum LLC size. Figure 3 shows the performance in terms of cycles per instruction (CPI, left axis), and memory bandwidth in terms of bytes read from main memory (BPKI, right axis) for the selected benchmarks.

In general, L2P is the best prefetcher. For the 12 prefetch-friendly applications, L2P alone achieves more than 82% of the CPI reduction, and more than 90% for 7 of them. The second best prefetcher is DCUI followed by DCUP. L2A gets the least improvement. It only reduces CPI more than 5% for 6 applications, with a maximum of 15% for `554.roms`.

Hardware prefetching is bandwidth-efficient since it causes significant extra bandwidth consumption in only three benchmarks (`520.omnetpp`, `549.fotonik3d` and `554.roms`).

Considering that LLC MPKI is improved for two of these three benchmarks (except `520.omnetpp`), the overall result is positive.

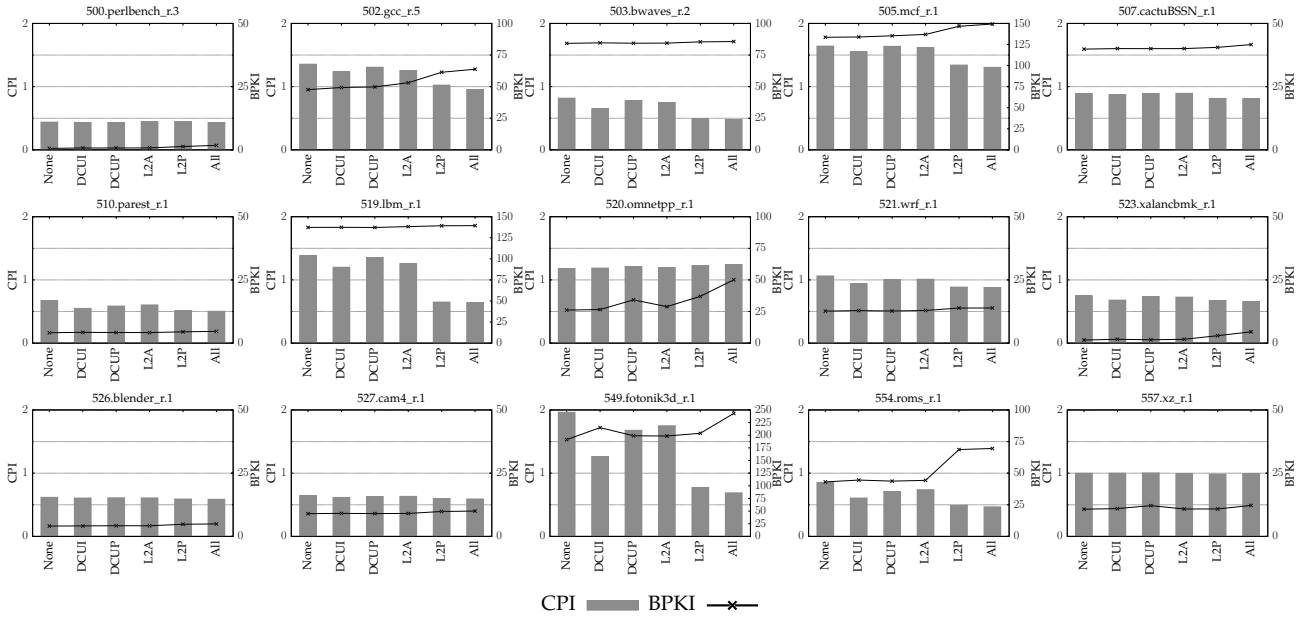

Figure 3: Impact of the hardware prefetchers on the application performance (CPI) and on the memory bandwidth (BPKI).

4 Conclusions

In this work we analyzed the memory hierarchy performance of the SPEC CPU2017 single-thread applications. We identified 19 out of 50 application/input pairs, from 9 applications, with little use of the memory hierarchy. They have very low miss ratios in the second and third level caches, even with small LLC sizes and without hardware prefetching.

We analyzed the sensitivity to the LLC size and to the different hardware prefetchers of the 15 memory-intensive applications. Prefetching is very effective in 12 of them, and the L2P prefetcher is responsible of a large fraction of the improvement. 9 applications show important MPKI reduction when increasing the LLC size even with prefetching.

References

[1] SPEC, SPEC CPU® 2017, `https://www.spec.org/cpu2017/`

[2] Nguyen, Khang T, *Introduction to Cache Allocation Technology in the Intel® Xeon® Processor E5 v4 Family*, Intel software developer zone 2016, `https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology`

[3] Intel, *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Intel 2016, update April, 2018.

[4] Arnaldo Carvalho de Melo. The new linux 'perf' tools. 2010.