Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 2557 – 2560

Procedia
Computer Science

International Conference on Computational Science, ICCS 2013

Using huge pages and performance counters to determine the LLC architecture

Josué Feliu[a,*], Julio Sahuquillo[a], Salvador Petit[a], José Duato[a]

[a]*Department of Computer Engineering (DISCA)*
Universitat Politècnica de València
Camí de Vera s/n, 46022, València, Spain

Abstract

Performance of current chip multiprocessors (CMPs) is strongly connected with the performance of their last level caches (LLCs), which mainly depends on the cache requirements of the processes as well as their interference. To effectively address such issues, researchers should be aware of the features of LLCs when performing research on real systems. Consequently, some research works have focused on experimentally determining such features, although most existing proposals take assumptions that are not met in current LLCs. To achieve this goal in real machines, we devised three tests that make use of *huge pages* to control the accessed cache sets, and *performance counters* to monitor the LLC behavior. The presented tests can be used in many experimental cache-aware research works; for instance in the design of thread scheduling policies.

Keywords: cache architecture; cache geometry; huge pages; performance counters; LLC

1. Introduction

Many recent research work has focused on evaluating the performance when running multiprogrammed workloads on a chip multiprocessor (CMP) system. Authors have focused on a better management of bandwidth utilization [1] and on reducing the interference in the last level cache (LLC) [2] [3] among the processes that are running together (from now on, co-runners) in the different cores of the processor.

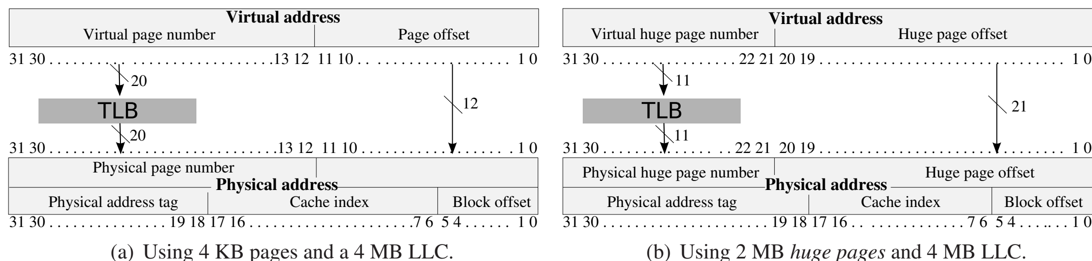
For a correct performance analysis concerning LLC caches, as well as for experiment design purposes, researchers require precise information about the LLC characteristics, mainly the cache geometry (cache size, associativity and line size) and the hierarchy organization (number of LLCs and cores sharing each LLC).

Some prior research works have also focused on determining cache parameters relevant for the system performance [4] [5] [6] [7]. Unfortunately, most of those works use time-based metrics, which often require complex timing analysis to deduce the LLC characteristics. Moreover, they do not report other important memory hierarchy information like the cache associativity or the cache organization.

To deal with caches physically indexed, this work relies on *huge page* capability provided by current microprocessors and operating systems that allows the OS to map large amounts of consecutive physical memory to a unique page. Performance counters are also used as a transparent and direct way to monitor the system behavior, avoiding complex analysis of time-based metrics. Thus, based on *huge pages* and using performance counters this paper presents three experiments to determine LLC characteristics of current processors.

*Corresponding author. Tel.: +34 963877007 Ext.:75738. *E-mail address:* jofepre@gap.upv.es.

1877-0509 © 2013 The Authors. Published by Elsevier B.V.
Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.440



(a) Using 4 KB pages and a 4 MB LLC. (b) Using 2 MB *huge pages* and 4 MB LLC.

Fig. 1: Address translation.

The rest of this paper is organized as follows. Section 2 describes the experimental platforms. Section 3 discusses the address translation. Section 4 presents the experiments and establishes the LLC features of the studied platforms. Section 5 describes related work . Finally, Section 6 presents some concluding remarks.

2. Experimental platforms

Experimental evaluation was performed in two shared-memory quad-core Intel processors: a Xeon X3320 and a Core i5 750, which implement a two-level and a three-level cache hierarchy, respectively. The LLC size is 6MB in the Xeon X3320 and 8MB in the Core i5 750. Both systems run a Fedora Core 10 Linux distribution with kernel 2.6.29. The performance monitor *Pfmon* [8] is used to get the values of the performance counters offered by the target processors. In particular, we gathered the number of LLC accesses and misses. Experimental results were performed disabling hardware prefetching, since prefetching requests can interfere in the analysis.

3. Virtual memory and address translation for standard pages and *huge pages*

Figure 1(a) presents the address translation of virtual addresses into physical addresses that are used to access the cache. The virtual addresses are split in two fields: virtual page number and virtual page offset. The virtual page number is formed with the most significant bits of the virtual address. For common 4 KB pages, the page offset is formed with the 12 less significant bits of the address, leaving the 20 most significant bits to identify the virtual page. The TLB only translates the virtual page number into the physical page number so that the bits of the page offset are not affected. The physical address is broken down into physical address tag, cache index and block offset to access the cache memory. The example of the figure corresponds to a 4MB 16-way set-associative LLC with 64-byte lines and 4096 sets.

Notice that, assuming a 2^6 bytes line size, the cache index field for caches with more than 2^6 sets, which is the common situation in current LLCs, is formed with bits from both the page offset and the physical page number fields. Since a user-level process does not have information about the TLB address translation, the physical page number cannot be determined. Consequently, processes are not able to control the accessed cache sets.

To address this shortcoming, we used *huge pages* of 2 MB instead of standard 4KB pages. Figure 1(b) presents the address translation when the process memory is mapped using *huge pages*. Virtual addresses are broken down in two fields as in standard pages, but with different lengths. The 21 less significant bits of the virtual address form the *huge page* offset to map the 2MB, and the 11 most significants bits are used for the virtual *huge page* number. The main difference is that using *huge pages* all the bits that form the cache index field come from the page offset, and thus, we can precisely control the accessed cache sets.

4. Experimental tests

4.1. Determining the cache line size

The pseudocode of the microbenchmark used to detect the cache line size is presented in Figure 2. The only input parameter of the algorithm is the array row length. For each run of the algorithm, a different array row length is checked, and the LLC hit ratio of the execution is obtained. To determine the cache line size of the experimental platforms, array row lengths ranging from 16 to 256 bytes are evaluated.

The expected behavior is that when the row length of the array matches the cache line size, every access will map to a different cache set. Since the array at least doubles the cache size, a block will be always evicted between

```
Input: ARRAY_ROW_LENGTH
  N = 2 * CACHE_SIZE / ARRAY_ROW_LENGTH
  char A[N][ARRAY_ROW_LENGTH]
  for (r=0; r<REPS; r++) do
    for (i=0; i<N; i++) do
      A[i][0] = 1
```

Fig. 2: Algorithm 1: pseudocode to determine the cache line size.

```
Input: STRIDE and ARRAY_ACCESSES,
  N = ARRAY_ACCESSES * STRIDE
  char A[N][CACHE_LINE_SIZE]
  for (r=0; r<REPS; r++) do
    for (i=0; i<ARRAY_ACCESSES; i++) do
      A[i*STRIDE][0] = 1
```

Fig. 3: Algorithm 2: pseudocode to determine the number of sets and associativity.

consecutive accesses and hence, the LLC hit ratio will approach to zero. Smaller array row lengths will reduce the number of misses because some array accesses would hit in the same block. On the contrary, larger array row lengths will keep the hit ratio close to zero. Figure 4 presents the results of this experiment in the Xeon X3320 and determine that the cache line size is 64 bytes. The Core i5 750 presents similar results (not shown due to their similarity).

4.2. Determining the number of sets and number of ways of the cache

The pseudocode of the microbenchmark used to establish the number of sets and ways of the cache is presented in Figure 3. The algorithm has been designed to check the cache hit ratio while varying the number of different lines that are accessed in a set. Input parameters are the stride used to access the array and the number of array accesses, which help us to determine the number of sets and ways of the cache, respectively.

The algorithm is executed for different tentative values of both input parameters, and the LLC hit ratio is measured. Figure 6 presents the results of this experiment for both processors. As expected, the hit ratio starts decreasing when the number of accesses to a given cache set is higher than the number of cache ways. In Figure 6(a) we observe that the hit ratio in the Xeon X3320 begins to decrease when the number of array accesses exceeds 12. Thus, the number of ways of the LLC is 12, since with a higher value the blocks start to be evicted. For lower values, the hit ratio is close to 100% because the microbenchmark accesses less blocks than ways in the set the accessed cache has. Similarly, from Figure 6(b) we can establish the LLC associativity of the Core i5 750 is 16 ways. On the other hand, the number of sets corresponds to the lowest stride where the hit ratio starts to decrease. That is, 4096 sets in the Xeon X3320 and 8192 sets in the Core i5 750. For shorter strides, the hit ratio is kept to 100% since the accesses are distributed in more than one cache set.

4.3. Determining the number of caches

The following experiment determines how many LLCs a processor has and which cores share each LLC cache. The idea is to use a microbenchmark that fills a target set with blocks, which are continuously accessed without causing misses. Then, to determine if there are shared caches, two instances of it are executed concurrently in different cores. If the cores share the cache, the capacity of the target set is surpassed so resulting in conflict misses. This test is performed for each possible pair of cores. The used microbenchmark shares the pseudocode presented in Figure 3, setting the stride to the number of sets, and the number of different accesses to the number of ways. Such configuration offer a LLC hit ratio by 100% in stand alone execution.

Figure 5 presents results for these experiments. Results for the Xeon X3320 shows that it has two LLCs, one shared by cores 0 and 2, and the other shared by cores 1 and 3, since running a benchmark in both cores drops the

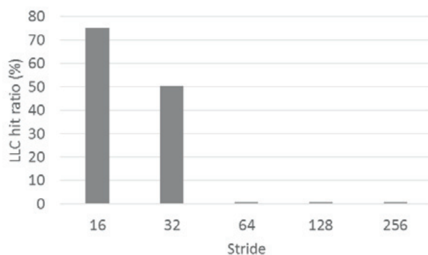


Fig. 4: Global cache hit ratio varying the array row length.

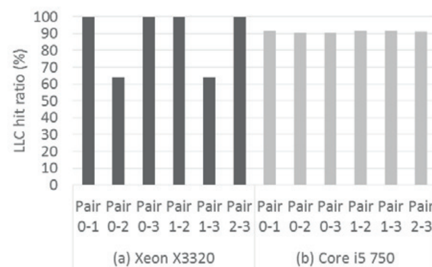
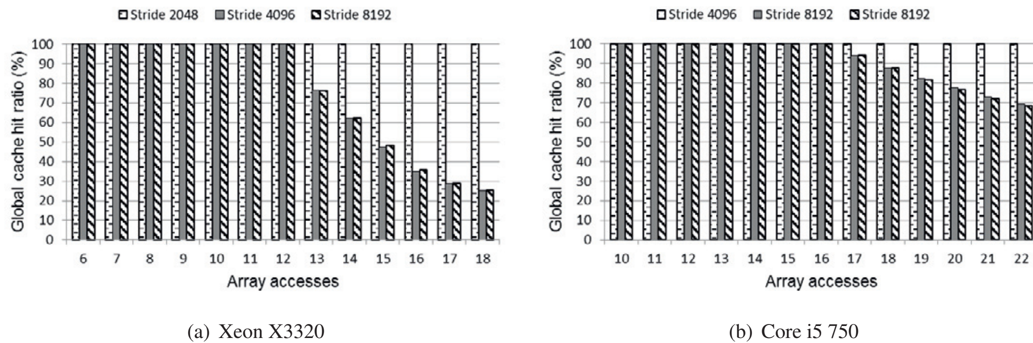


Fig. 5: Average cache hit ratio when running two instances of the algorithm 2 on different core pairs.



(a) Xeon X3320 (b) Core i5 750

Fig. 6: Global cache hit ratio varying the number of array accesses and stride.

LLC hit ratio to 60%. On the other hand, results for the Core i5 750 show that this processor has a single LLC shared by all the cores, since running two benchmark in any pair of cores drops the LLC hit ratio down to 90%.

5. Related work

A widely known algorithm to measure memory hierarchy parameters is the proposed by Saavedra [4]. To obtain the memory hierarchy features, this benchmark accesses regularly the components of a large array varying the stride between accesses and measuring the access latency. The study assumes that cache structures are virtually indexed, i.e., “an array which occupies a contiguous region of virtual memory also occupies a contiguous region of cache memory”. Since current LLC caches are commonly physically indexed and the LLC index cannot be determined from the virtual address, their approaches cannot be directly applied to current LLC caches. Moreover, time-based metrics, which require complex timing analysis to deduce the LLC characteristics, are used.

There also exist several tools [5] [6] [7] that provide some hardware parameters, such as cache capacity, block size and latency. However, they do not report other important memory hierarchy information like the cache associativity or the cache organization. Moreover, they use time-based metrics, requiring complex analysis.

6. Conclusions

As the cache hierarchies are becoming more and more complex and the main memory latency, measured in processor cycles, dramatically grows in current CMPs, the impact of the LLC performance on the overall system performance continues increasing. Therefore, when dealing with experimental research, information about LLC features is required. In this this paper we have designed three experiments that make use of *huge pages* and performance counters to determine for LLCs in current microprocessors i) the cache line size, ii) the associativity degree and the number cache sets, and iii) the number of LLCs and which cores share each LLC cache.

Acknowledgements

This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and Plan E funds, under Grant TIN2009-14475-C04-01, and by Programa de Apoyo a la Investigación y Desarrollo (PAID-05-12) of the Universitat Politècnica de València under Grant SP20120748.

References

[1] D. Xu, C. Wu, P.-C. Yew, On mitigating memory bandwidth contention through bandwidth-aware scheduling, in: PACT, 2010, pp. 237–248.
[2] J. Feliu, J. Sahuquillo, S. Petit, J. Duato, Understanding Cache Hierarchy Contention in CMPs to Improve Job Scheduling, in: Parallel Distributed Processing (IPDPS), 2012 IEEE International Symposium on, 2012, pp. 508 – 519.
[3] L. Tang, J. Mars, N. Vachharajani, R. Hundt, M. L. Soffa, The impact of memory subsystem resource sharing on datacenter applications, in: Proc. of the 38th ISCA, 2011, pp. 283–294.
[4] R. H. Saavedra-Barrera, Cpu performance evaluation and execution time prediction using narrow, Tech. rep., Berkeley, CA, USA (1992).
[5] L. McVoy, C. Staelin, lmbench: portable tools for performance analysis, in: Proc. of USENIX, pp. 279 – 294.
[6] K. Yotov, K. Pingali, P. Stodghill, X-ray: a tool for automatic measurement of hardware parameters, in: Quantitative Evaluation of Systems, 2005. Second International Conference on the, 2005, pp. 168 – 177.
[7] J. Gonzalez-Dominguez, G. Taboada, B. Fraguela, M. Martin, J. Tourio, Servet: A benchmark suite for autotuning on multicore clusters, in: IPDPS, 2010, pp. 1 –9.
[8] S. Eranian, What can performance counters do for memory subsystem analysis?, in: Proceedings of the 2008 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness, 2008, pp. 26–30.