Accurate Cache and TLB Characterization Using Hardware Counters

Jack Dongarra, Shirley Moore, Philip Mucci, Keith Seymour, and Haihang You

Innovative Computing Laboratory, University of Tennessee
Knoxville, TN 37996-3450 USA
{dongarra,shirley,mucci,seymour,you}@cs.utk.edu

Abstract. We have developed a set of microbenchmarks for accurately determining the structural characteristics of data cache memories and TLBs. These characteristics include cache size, cache line size, cache associativity, memory page size, number of data TLB entries, and data TLB associativity. Unlike previous microbenchmarks that used time-based measurements, our microbenchmarks use hardware event counts to more accurately and quickly determine these characteristics while requiring fewer limiting assumptions.

1 Introduction

Knowledge of data cache memory and data TLB characteristics is becoming increasingly important in performance optimization and modeling. Cache-conscious algorithmic design is the basis of tuned numerical libraries, such as the BLAS and LAPACK and has been shown to improve full application performance significantly [4,5]. Compilers and automatically tuned software systems such as ATLAS [11] and PHiPAC [2] need accurate information about cache and TLB characteristics to generate optimal code. This information is also important for performance modeling techniques such as those described in [8,9]. In [9], it is hypothesized that much of the error in the miss surfaces used for the performance modeling lies in the creation of the cache characterization surfaces. This and other performance modeling work would benefit from more accurate cache characterization for new architectures.

Published information about detailed cache and TLB characteristics can be difficult to find or may be inaccurate or out-of-date. Thus, it will often be necessary to determine or verify this information empirically. This paper describes a methodology of instrumenting some microbenchmark codes that exercise the cache and memory subsystem to collect hardware counter data for cache and memory events. These data can then be used to give accurate numbers for cache and TLB characteristics. We describe how our methodology builds on and extends previous work based on timings of microbenchmarks, give results for the Itanium2 processor, and discuss ideas for future work and applications of the results.

2 Methodology

Previous work measured the time to execute simple memory-bound loops on arrays of different sizes and with different strides, and estimated the cache and TLB characteristics from the results [7,10]. The data cache and TLB characteristics can be calculated once the number of memory references and the number of cache and TLB misses are known for the different array sizes and strides. Previous work inferred the numbers of misses from the timing results. In [7], an analytical model is developed for a single cache that identifies four cases that show up as four distinct regions on the timing curves. The model assumes a single blocking cache with an LRU replacement policy, and it can determine cache size, cache line size, and cache associativity from the timing results. The same model is used for TLB characterization, and the values for the array size and stride at which the TLB phenomena occur are different enough from those for the cache phenomena that the effects can be isolated from one another.

In [10], the analytical model from [7] is first extended from four to six cases to include transitional portions of the timing curves. The model is further extended to a system with two caches. The two-cache model assumes that the second-level cache includes the first-level, that both caches have the same block size, that associativity is non-decreasing, that there is not prefetching, and that the replacement policy for both caches is LRU. Six cases are identified and some fairly complicated timing formulas are given that allow the cache characteristics of the two caches to be calculated from timing results for the microbenchmarks. Next the model is extended to a two-cache system with a TLB. Some assumptions on the TLB operation are made, seven cases are identified, and another set of formulas is given.

The Calibrator microbenchmark described in [6] varies stride and array size in a small loop that executes a million memory reads in order to determine cache characteristics including cache size, cache line size, TLB size, page size, and miss latencies. The microbenchmark uses a pointer chasing scheme to try to prevent latency hiding. Results are given for SGI Origin 2000, Sun Ultra, Intel Pentium III, and AMD Athlon.

The above approaches break down when the simplifying assumptions do not hold – e.g., in the presence of non-blocking caches or prefetching. In some newer processors, the assumption of cache inclusivity does not always hold. These factors, combined with variability in timing results, make it difficult to obtain cache and TLB characteristics accurately on new architectures using the above approaches.

Our approach uses hardware counters to obtain precise values for the number of misses at different levels of data cache and for the data TLB. The cache characteristics can then be directly inferred from the miss counts, rather than indirectly from timing data. To collect these data, we use the PAPI portable interface to hardware performance counters [3]. PAPI defines both a standard set of routines for accessing the counters and a standard set of events. As many as possible of the PAPI standard events are mapped to native events on a given platform. On processors where they are available, we measure PAPI standard

Table 1. PAPI standard cache and TLB events

PAPI_L1_DCM	Level 1 data cache misses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L3_DCM	Level 3 data cache misses
PAPI_TLB_DM	Data TLB misses

events shown in Table 1 for our microbenchmarks. We then plot separate curves for each of these events. The cache and TLB characteristics can be read directly off each curve as well as being determined exactly by the numerical results.

For processors on which one or more of the above events are not available, we can use those events that are available to accurately determine the cache characteristics for those levels and then deduce the remaining characteristics using the formulas from [10]. On some platforms, another event may be available that is closely correlated to the missing event and thus can be used as a substitute.

Due to the fact that cache and TLB miss events exhibit little or no variability from run to run, whereas timing measurements are more variable, our hardware counter approach requires fewer trials to achieve accurate results. This savings in time is important for applications such as automatically tuned software that need to generate optimal code in a reasonable amount of time.

We have developed two microbenchmarks:

- `papi_cacheSize` which increases the size of the array being referenced with stride one until it exceeds the capacities of the various levels of cache memory. Both integer and floating point type data are tested, since on some architectures, such as the Itanium 2, these types are treated differently by the cache and memory system. The loop is timed and the number of cache misses at each level measured for each array size. A sharp increase in the number of cache misses occurs when the cache size is exceeded.
- `papi_cacheBlock` which increases the stride with which an array of fixed size is being referenced (with the array size chosen depending on the cache size) by powers of two from stride one to stride equal to the array size. Again, both integer and floating point data are tested. The total number of iterations is held constant. A sharp increase in the number of misses occurs at the cache line size, and a drop to zero occurs when the total number of array items referenced fits within one cache set.

For analysis of the `papi_cacheBlock` results, we consider the following cases, where C is the cache size, N is the array size, s is the stride, b is the cache block or line size, and a is the cache associativity:

1. $N > C$ and $1 \leq s \leq b$
   There is one miss every b/s accesses. Since we double the number of iterations as we double the stride, the total number of misses also doubles.
2. $N > C$ and $b \leq s < N/a$
   There is one miss every access. We reduce the number of different cache lines accessed by a factor of two every time we double the stride in this region,

but since we also reduce the number of cache congruence classes in use by the same factor, the number of misses stays constant in this region.

3. $N > C$ and $N/a \leq s \leq N$
Only $N/s \leq a$ array elements are accessed. Since all of them fit in a single cache set, there are no more misses.

3 Results

In this section, we show representative results for the Itanium 2 platform. Similar results can be obtained on any of the platforms on which PAPI is implemented, which include IBM POWER3/4, MIPS R10K/R12K/R14K/R16K, Sun UltraSparc, HP Alpha, Pentium, Opteron, and Cray X1, although on some of these platforms we are limited by the availability of hardware counter events for different levels of cache and TLB. We show the results in graphical form for illustrative purposes. Our benchmark codes produce exact numerical results by initially increasing the array sizes for the tests by powers of two and then using a more refined search to pinpoint the exact array size where the change in cache or TLB misses occurs.

Our results for the Itanium 2 are shown in Figures 1-6. The average memory access time for each test is calculated by dividing the CPU time for the loop in the test by the total number of memory references. Figure 1 shows the results of running the `papi_cacheSize` microbenchmark instrumented to count the PAPI_L1_DCM, PAPI_L2_DCM, and PAPI_L3_DCM events in addition to timing the execution of the loop for different array sizes up to 8 megabytes using integer data type. The curves show the L1 data cache size of 16KB and the L2 cache size of 256KB. For the L3 cache size, we refine the search (not shown due to space limitations) to determine the L3 cache size of 1.5MB. Figure 2 shows the results for the events PAPI_L2_DCM and PAPI_L3_DCM for double precision real data type. L1 data cache misses are not shown for double precision real data type, because floating point loads bypass the L1 data cache and L1 DTLB. The faster time for real data is most likely due to the high bandwidth data paths to and from the L2 cache and the floating point register file and the L1 cache bypass for floating point data. Figure 3 shows the `papi_cacheSize` microbenchmark instrumented to count L1 DTLB misses and PAPI_TLB_DM (which is mapped to native L2DTLB_MISSES) using integer data type. The L1 DTLB misses are generated by adding the counts for the native events L1DTLB_TRANSFER (L1DTLB misses hit in L2DTLB) and L2DTLB_MISSES. Because the L1 DTLB is a strict subset of the L2 DTLB, a L2 DTLB miss is also a L1 DTLB miss. We see that the L1 DTLB has 32 entries since L1 DTLB misses begin to occur at an array size of $32 \times 4K = 2^{17}$ bytes (see papi_cacheBlock results below for the page size). Similarly, we see that the L2 DTLB has 128 entries since the page size is 16K and misses begin to occur at an array size of $128 \times 16K = 2^{21}$ bytes.

The remaining figures show the results of running the `papi_cacheBlock` microbenchmark to determine the cache line sizes, TLB page sizes, and cache

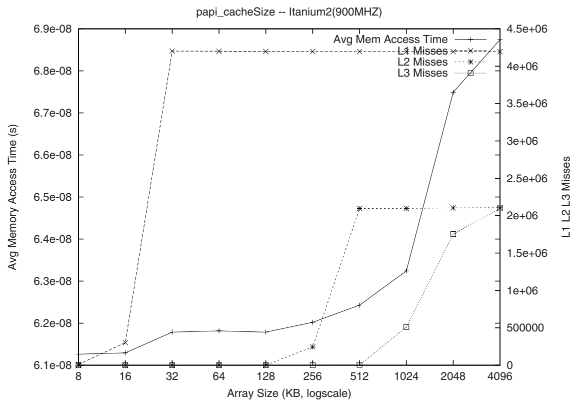

Fig. 1. Itanium2 Cache Sizes with Integer Data Type

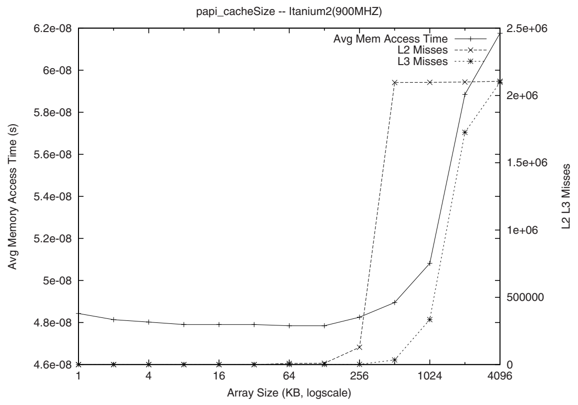

Fig. 2. Itanium2 Level 2 and 3 Cache Sizes with Real Data Type

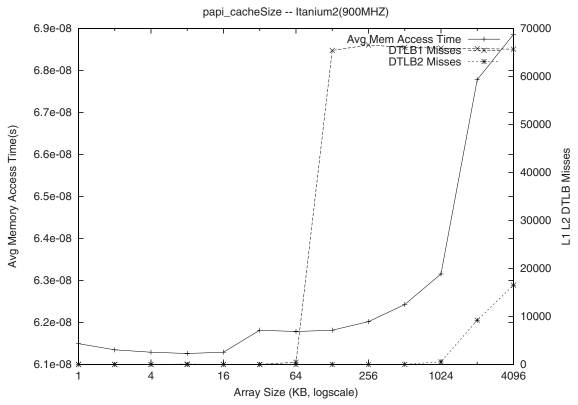

Fig. 3. Itanium2 Level 1 and 2 DTLB Sizes with Integer Data Type

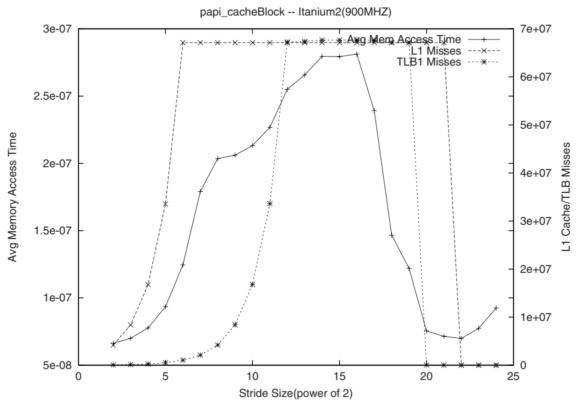

Fig. 4. Itanium2 Level 1 Cache and TLB Characteristics with Integer Data Type

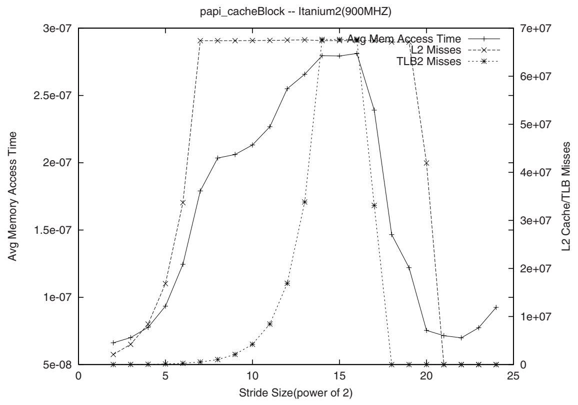

Fig. 5. Itanium2 Level 2 Cache and TLB Characteristics with Integer Data Type

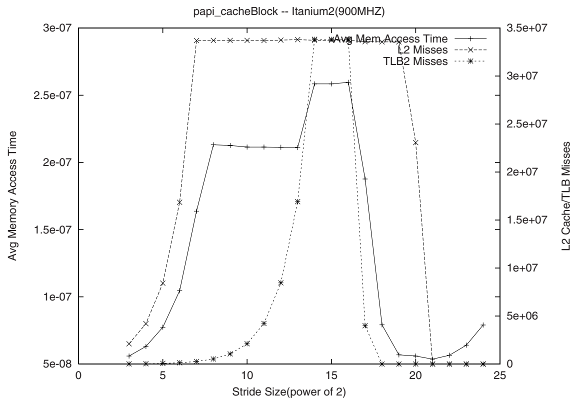

Fig. 6. Itanium2 Level 2 Cache and TLB Characteristics with Real Data Type

and TLB associativities. These quantities are calculated using the three cases described at the end of section 2. Figure 4 shows the results of running the `papi_cacheBlock` microbenchmark instrumented to count the PAPI_L1_DCM and native L1 DTLB miss events for the integer data type. We can see that the L1 cache line size is $2^6 = 64$ bytes and that the L1 cache has 4-way associativity. We can also see that the L1 DTLB page size is $2^{12} = 4K$ bytes and that the L1 DTLB is fully associative. Figure 5 shows the results of running the `papi_cacheBlock` microbenchmark instrumented to count the PAPI_L2_DCM and PAPI_TLB_DM (which is mapped to native L2DTBL_MISSES) events for integer data type, and Figure 6 shows the same results for double precision real data type. We can see that the L2 cache line size is $2^7 = 128$ bytes and that the L2 cache has 8-way associativity. We can also see that the page size is 16K bytes and that the L2 DTLB is fully associative. Note that the curves for Figures 5 and 6 have the same shape but that the execution time in Figure 6 is less. Again, this difference is most likely due to the high bandwidth data paths to and from the L2 cache and the floating point register file and the L1 cache bypass for floating point data. The reason for the difference in pages sizes for the L1 DTLB and L2 DTLB is that the L1 DTLB has a fixed page size of 4KB. Larger page sizes are supported by allocating additional L1 DTLB entries as a 4KB portion of the larger page [1].

4 Conclusions and Future Work

We have shown how hardware counter measurements can be used to generate accurate information about data cache and TLB characteristics on modern microprocessors. Even for processors where non-blocking caches, prefetching, and other latency-hiding techniques can make determining cache and TLB characteristics from timing results difficult, hardware counter measurements can produce accurate results.

We plan to release the `papi_cacheSize` and `papi_cacheBlock` microbenchmarks as part of PAPI 3.0. Scripts will be included for analyzing and plotting the data. The most portable and efficient means available will be used for determining the cache and TLB characteristics, starting with PAPI standard cache and TLB events. Next native cache and TLB events will be used, and finally other hardware events and timing results from which cache and TLB characteristics can be inferred.

Caches of some modern microprocessors are split into banks in order to allow interleaved access to data in different banks. For example, the Itanium 2 Level 2 data cache is organized into sixteen banks, with an interleaving of 16 bytes. We are investigating how to extend our microbenchmark approach to determine the number of cache banks, which can be an important factor affecting performance of the cache.

We plan to use automated determination of cache and TLB information in our research to extend semi-automatic tuning, already demonstrated for the BLAS [11] to higher level libraries, including sparse methods. The information will also be used to generate more accurate machine signatures for performance modeling efforts in which we are collaborating.

For more information about PAPI, including software and documentation, see the PAPI web site at `http://icl.cs.utk.edu/papi/`.

References

1. *Intel Itanium 2 Processor Reference Manual.* http://developer.intel.com/, Apr. 2003.
2. J. Bilmes, K. Asanovic, C.-W. Chin, and J. Demmel. Optimizing matrix multiply using PhiPAC: a portable high-performance ANSI C coding methodology. In *Proc. International Conference on Supercomputing*, Vienna, Austria, 1997.
3. S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *International Journal of High Performance Computing Applications*, 14(3):189–204, 2000.
4. T. M. Chilimbi, M. D. Hill, and J. D. Larus. Cache-conscious structure layout. In *Proc. 1999 ACM SIGPLAN Conference on Programming Languages and Implementation (PLDI)*, pages 1–12, 1999.
5. W. Jalby and C. Lemuet. Exploring and optimizing Itanium2 cache performance for scientific computing. In *Proc. 2nd Workshop on EPIC Architectures and Compiler Technology*, Istanbul, Turkey, Nov. 2002.
6. S. Manegold and P. Boncz. *Cache-Memory and TLB Calibration Tool.* http://homepages.cwi.nl/ manegold/Calibrator/calibrator.shtml, 2001.
7. R. H. Saavedra and A. J. Smith. Measuring cache and TLB performance and their effect on benchmark runtimes. *IEEE Transactions on Computers*, 44(10):1223–1235, 1995.
8. A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. A framework for performance modeling and prediction. In *Proc. SC2002*, Baltimore, MD, Nov. 2002.
9. E. S. Sorenson and J. K. Flanagan. Cache characterization surfaces and predicting workload miss rates. In *Proc. 4th IEEE Workshop on Workload Characterization*, pages 129–139, Austin, Texas, Dec. 2001.
10. C. Thomborson and Y. Yu. Measuring data cache and TLB parameters under Linux. In *Proc. 2000 Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pages 383–390. Society for Computer Simulation International, July 2000.
11. R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1-2):3–25, 2001.