MEASURING DATA CACHE AND TLB PARAMETERS UNDER LINUX

Clark Thomborson
Yuanhua Yu
Computer Science Department
University of Auckland
Private Bag 92019
New Zealand
cthombor@cs.auckland.ac.nz
http://www.cs.auckland.ac.nz/~cthombor

Keywords: computer performance, performance analysis, computer software, cache memory, translation lookaside buffer.

ABSTRACT

We develop an analytic model, and a set of microbenchmark programs for the measurement of the structural parameters of data cache memories and data TLBs. Running under Linux, our microbenchmarks accurately measure data cache capacity, data cache line size, data cache associativity, effective cache latency, effective data path parallelism, data TLB size, data TLB associativity, and TLB latency. We present experimental results from running our microbenchmarks on Pentium II and Pentium III workstations.

1 INTRODUCTION

As processor speed continues to increase faster than memory speed, optimizations to use the cache memory hierarchy efficiently become more important, especially in high performance computer systems. Cache memories help bridge the cycle-time gap between microprocessors and relatively slower main memories, by taking advantage of data locality in programs. Compilers and application programmers are increasingly designed with knowledge of caches [1, 2, 5, 7, 8]. In recent studies, cache-conscious algorithmic design improved the performance of an operating system by more than 30% [11], some standard search algorithms by a factor of 2 to 5 [3], and full application programs by 27% to 42% [3]

Any person (or compiler) who optimises memory operations for high-performance software will need accurate information about the structural and performance parameters of the memory system. The structure of a cache is primarily characterized by its cache capacity *C*, line size *B* and associativity *A*. Of secondary importance are its replacement and prefetching strategies. In the past, these parameters were not available to compiler designers, application programmers and system administrators. Even now, it is difficult to discover anything more than the most basic parameters such as cache capacity. We believe the most important performance parameters of a memory hierarchy are the data cache miss latency, the TLB (Translation Lookaside Buffer) miss latency, and the effective data path parallelism. Such parameters are difficult or impossible to evaluate from published data, and we know of no automated system for their evaluation on Pentium based workstations.

In this paper, we develop an analytical model to estimate the structural and performance parameters of cache memory and TLB by considering the runtime of simple memory-bound inner loops. We develop a set of micro benchmarks, called MBCP under the Linux operating system to automate the data collection process and to partially automate the parameter estimation process. Finally we present experimental results on Pentium II/266 and Pentium III/500 workstations.

2 LITERATURE REVIEW

In the last three decades, great progress has been made on cache and TLB research in three areas: evaluation techniques, such as trace/trap-driven simulation and some analytical methods; code optimization techniques, such as loop transformation; and cache protocol development. But careful study of the measurement and estimation of the structural parameters of cache and TLB seems to have begun only in the last decade.

Pyo and Lee made an estimation of cache parameters based on reference distance. They defined reference distance as the number of memory blocks referenced in a "reuse interval," which in turn is defined as the sequence of memory references between two consecutive references to the same memory block. They measured the distribution of reference distances on programs from the Perfect benchmark suite. Pyo and Lee obtained the cache capacity, line size and set size for set-associative cache, by finding to the cost-effective points on their "reuse cover curves." Pyo and Lee's method of estimation of cache parameters was a byproduct of their study and was not intended to measure the parameters of an unknown cache [10].

Saavedra and Smith developed a set of narrow spectrum benchmarks or micro benchmarks to measure the physical and performance characteristics of the memory hierarchy in uniprocessors, in particular, the primary and secondary caches and TLB. Saavedra's program makes two hundred observations of a single test covering all the combinations of: 1) the size of the sequential address space touched by the experiment and 2) the distance (stride) between two consecutive addresses sent to the cache or TLB. Saavedra's micro benchmarks work under following conditions: instruction caches and data caches are separate, and the lowest available address bits are used to select the cache set [9].

Li and Thomborson extended Saavedra and Smith's research on designing micro benchmarks to measure data cache parameters. Unlike Saavedra and Smith, Li and Thomborson characterized read accesses separately from write, and their benchmarks were valid for wider range of address mapping functions. Their micro benchmarks used randomised access sequences, and were developed under Windows NT. They measured cache physical parameters such as capacity, block size, and associativity. They also determined the cache write policies (allocate on write, write-back, and write-through). Li and Thomborson's work was limited to uniprocessor systems and did not cover the measurement of the parameters of TLB; and it did not give accurate estimates of associativity A when $A \geq 8$ [6].

3 CHARACTERIZING THE PERFORMANCE OF CACHE MEMORY SYSTEMS

Before explaining our experimental methodology, we note our underlying assumptions:

- Instruction fetches in compact inner loops will not affect the data cache;
- Memory is byte-addressable, so that the memory access strides can be measured in bytes;
- The number of sets S_i in level i (i = 1, 2) cache is C_i/A_iB_i, where C_i is the cache capacity in bytes, A_i is the degree of cache associativity, and B_i is the line (or block) size in bytes;
- The replacement strategy is LRU;
- When indexing the cache to find the set corresponding to a memory address x, we use the least-significant bits of x/B_i where B_i is the block size of the cache.

Our methodology is based on inner loops that read data, or write data, or compute a simple function on each of a subset of elements taken from a very large one-dimensional array of N 4-byte elements: $\{e_0, e_1, e_2, e_3, \ldots, e_{N-1}\}$. Our loops make "strided" array accesses to every s-th element. The reference sequence thus consists of the N/s elements $\{e_0, e_{s/4}, e_{2s/4}, e_{3s/4}, \ldots, e_{N-s/4}\}$ where s is the stride (in bytes) of our array accesses. In our benchmark experiments we modify s to change the rate at which misses are generated. The magnitude of s varies from 4 to $2N$ in powers of two. Our array size N is also a power of 2.

3.1 A SYSTEM WITH ONE CACHE

Depending on the values of the array size N, the access stride s, the primary cache capacity C_1, the cache line size B_1, the cache associativity A_1, and the number of sets S_1, we identify six cases of operations. These are summarised in **Table 1**. Most but not all of these cases were previously identified by Saavedra and Smith [9].

Case 1.1: $1 \leq 4N \leq C_1$ and $1 \leq s$

The whole array fits in the cache. Regardless of the stride s, there are no cache misses after the array is loaded into the cache for the first time. The execution time per iteration of our inner loop is thus some constant $T_1 = T_0$.

Case 1.2: $C_1 < 4N < C_1 + S_1B_1$ and $1 \leq s < B_1$

The array is slightly bigger than the cache. Some cache sets attract only A_1 references, so references to these sets never miss. Other sets attract A_1+1 references, causing frequent misses under our LRU assumption. Let x be the number of sets in which misses occur, with $1 \leq x \leq S_1$. The array size $4N = x(A_1 + 1)B_1 + (S_1 - x)A_1B_1 = x\,B_1 + S_1A_1B_1 = x\,B_1 + C_1$. Thus $x = (4N - C_1)/B_1$. Because there are $4N/s$ references to the array, and each set in x causes a total of A_1+1 misses each time our inner loop cycles through the whole array, the miss ratio is $x(A_1 + 1)/(4N/s) = (s/B_1)(1 - C_1/4N)(A_1+1)$. The execution time per iteration of our loop is $T_1 = T_0 + (s/B_1)(1 - C_1/4N)(A_1 + 1)T_{miss1}$, where T_{miss1} is the additional time needed to read the data into primary cache.

Case 1.3: $C_1 < 4N < C_1 + S_1B_1$ and $B_1 \leq s < 4N/A_1$

This is similar to Case 1.2, except the number of sets x where misses occur is $x = (4N - C_1)/s$. The iteration time is $T_1 = T_0 + (1 - C_1/4N)(A_1+1)T_{miss1}$.

Case 1.4: $C_1 + S_1B_1 \leq 4N$ and $1 \leq s < B_1$

The array is bigger than the cache. The first of the B_1/s consecutive accesses to each cache line always misses, because every line is displaced from the cache before it is re-used. The execution time per iteration is thus $T_1 = T_0 + (s/B_1)T_{miss1}$.

Case 1.5: $C_1 + S_1B_1 \leq 4N$ and $B_1 \leq s < 4N/A_1$

The array is much bigger than the cache. There is a cache miss for every iteration, because each element of the array maps to a different cache line. Every cache line is displaced from the cache before it can be re-used. The execution time per iteration is $T_1 = T_0 + T_{miss1}$.

Case 1.6: $C_1 + S_1B_1 \leq 4N$ and $4N/A_1 \leq s$

Because the stride s is so large, the associativity A_1 of the cache is wide enough to capture all $4N/s$ accesses to the array. The execution time per iteration is $T_1 = T_0$.

Cases	**Size of Array**	**Stride**	**Frequency of L1 misses**	**Time per Iteration T_1**
1.1	$1 \leq 4N \leq C_1$	$1 \leq s$	None	T_0
1.2	$C_1 < 4N < C_1 + S_1B_1$	$1 \leq s < B_1$	Some	$T_0 + (s/B_1)(1 - C_1/4N)(A_1 + 1)T_{miss1}$
1.3	$C_1 < 4N < C_1 + S_1B_1$	$B_1 \leq s < 4N/A_1$	Some	$T_0 + (1 - C_1/4N)(A_1 + 1)T_{miss1}$
1.4	$C_1 + S_1B_1 \leq 4N$	$1 \leq s < B_1$	One per B_1/s elements	$T_0 + (s/B_1)T_{miss1}$
1.5	$C_1 + S_1B_1 \leq 4N$	$B_1 \leq s < 4N/A_1$	One per element	$T_0 + T_{miss1}$
1.6	$C_1 + S_1B_1 \leq 4N$	$4N/A_1 \leq s$	None	T_0

Table 1. Cache miss patterns as a function of N and stride s in the primary cache.

3.2 A SYSTEM WITH TWO CACHES

In this section, we extend Saavedra and Smith's analysis to cover two-cache systems. We start by considering the following desirable properties for hierarchical caches:

> **P1: Set-refinement.** The set-mapping function f_2 *refines* the set-mapping function f_1 if $f_2(x) = f_2(y)$ implies $f_1(x) = f_1(y)$, for all blocks x and y. Note: the set mapping function f_i for cache L_i is used to select a set index (in the range 0, ... S_i - 1), given memory address x.
>
> **P2: Inclusion.** Cache L_2 *includes* a cache L_1 if, for any block x after any series of references, the statement "x is resident in cache L_1" implies the statement "x is resident in cache L_2." That is, if cache L_2 includes cache L_1, we know that cache L_2 always contains a superset of the blocks in cache L_1.

In addition to the assumptions we made for the one-cache system, we further assume that

- f_2 refines f_1 (property **P1**),
- the associativity is non-decreasing ($A_2 \geq A_1$), and
- both caches have the same blocksize $B = B_2 = B_1$.

These assumptions imply property **P2** by the following theorem of Hill [4].

Theorem. Cache L_2 includes Cache L_1 (property **P2**) if property **P1** (set – refinement) holds, if both caches have the same blocksize, if $A_2 \geq A_1$, if there is no prefetching, and if the replacement policy in both caches is LRU.

Depending on the array size $4N$, stride s, capacity C_2, block size B, the associativity A_2, and the number of sets S_2, we identify six cases of operation. See **Table 2**.

Case 2.1: $1 \leq 4N < C_2$

The entire array fits in secondary cache, so there are no misses at this level. The reference time is the same as predicted in Table 1: $T_2 = T_1$.

Case 2.2: $C_2 < 4N < C_2 + S_2B$ and $1 \leq s < B$

The array is bigger than the secondary cache, so misses may occur. The analysis is similar to Case 1.2 above, with $x = (4N - C_2)/B$ and $T_2 = T_1 + (s/B)(1 - C_2/4N)(A_2 + 1)T_{miss2}$, where T_{miss2} is the L2 miss penalty: the additional time needed to read the data into secondary cache.

Case 2.3: $C_2 < 4N < C_2 + S_2B$ and $B \leq s < 4N/A_2$

There is only one reference per cache line. The miss ratio is accordingly a factor of B/s larger than in Case 2.2, so $T_2 = T_1 + (1 - C_2/4N)(A_2 + 1)T_{miss2}$.

Case 2.4: $C_2 + S_2B \leq 4N$ and $1 \leq s < B$

There is one miss per B/s consecutive accesses to each cache line because each cache line is displaced before it is re-used. The execution time per iteration is $T_2 = T_1 + (s/B)T_{miss2}$.

Case 2.5: $C_2 + S_2B \leq 4N$ and $B \leq s < 4N/A_2$

Similar to Case 2.4, except that there is only one reference to each cache line: $T_2 = T_1 + T_{miss2}$.

Case 2.6: $C_2 + S_2B \leq 4N$ and $4N/A_2 \leq s$

Similar to Case 1.6, the stride is so large that the $4N/s$ references can be handled without misses by the associativity A_2 of the secondary cache. The access time is $T_2 = T_1$.

Case	Size of Array	Stride	Frequency of L2 Misses	Time per Iteration T_2
2.1	$1 \leq 4N < C_2$	$1 \leq s < 4N/2$	None	T_1
2.2	$C_2 < 4N < C_2 + S_2B$	$1 \leq s < B$	Some	$T_1 + (s/B)(1 - C_2/4N)(A_2 + 1)T_{miss2}$
2.3	$C_2 < 4N < C_2 + S_2B$	$B \leq s < 4N/A_2$	Some	$T_1 + (1 - C_2/4N)(A_2 + 1)T_{miss2}$
2.4	$C_2 + S_2B \leq 4N$	$1 \leq s < B$	One per B/s references	$T_1 + (s/B)T_{miss2}$
2.5	$C_2 + S_2B \leq 4N$	$B \leq s < 4N/A_2$	One per reference	$T_1 + T_{miss2}$
2.6	$C_2 + S_2B \leq 4N$	$4N/A_2 \leq s$	None	T_1

Table 2. Cache miss patterns as a function of N and stride s in the secondary cache

3.3 A TWO-CACHE SYSTEM WITH A TLB

Theoretically speaking, the performance characteristics of a TLB are very similar to those of a cache, because a TLB is nothing more than a data cache specialised to hold page table entries. However, in practice the measurement of the parameters of a TLB is usually more difficult than for a cache, because it is difficult to cause TLB misses without also causing cache misses. In this section we describe two novel access patterns that allow us to isolate the effects of TLB misses.

Before characterizing the performance of the TLB, we must make some additional assumptions as follows.

- The instruction TLB (ITLB) is distinct from the data TLB (DTLB). We focus on the DTLB.
- Each entry in the TLB holds just one page table entry (PTE) of at most 8 bytes. We measure the capacity C_T of the TLB by counting the PTEs it holds, so $B_T = 1$.
- The TLB can be completely from the primary cache, using at most half its capacity ($8C_T \leq C_1/2$).
- The TLB uses a strict LRU replacement strategy.
- If the TLB is not fully associative, then the least significant address bits in the page-frame number are used to index the TLB.
- The page size P of the virtual memory system is a power of 2.

In our measurements, we use a similar access pattern to the one we used in the previous section to measure cache parameters. We define a set of M stride-modifying constants r_i such that $0 \leq r_i < s$ for all i in the range $1 \leq i \leq M = 4N/s$. Our microbenchmark sequentially accesses elements $\{e_{r_1}, e_{s + r_2}, e_{2s + r_3}, e_{3s + r_4}, \ldots, e_{(M-1)s + r_M}\}$ from an array $\{e_0, e_1, e_2, e_3, \ldots, e_{N-1}\}$ of N four-byte words. Thus our access sequences for TLB estimation have M word references at approximately (but not exactly) stride s. Note that $M = 4N/s$ is an integer, because both N and s are powers of 2.

We use two access patterns to discover the performance parameters of a TLB. In our "incremented offset" access sequence, we take $r_i = (iBC_1) \bmod s$. In our "random offset" access, we choose our r_i independently from the uniform distribution on the range $[0,1, .., s - 1]$.

Depending on the values of the reference count M, stride s, stride offset r_i, TLB size C_T, and associativity

A_T, we find seven cases. See **Table 3**. Note that $M \leq C_1/16$ throughout our analysis, because we have assumed the TLB can be repopulated from primary cache.

Case 3.1: $1 \leq M < C_TP/s$

In this case, there are P/s consecutively accessed elements using the same PTE in the TLB. All of the PTEs of the accessed elements can fit into the TLB, so there are no TLB misses. The execution time is $T_3 = T_0$, where T_0 is the primary cache hit time.

Case 3.2: $C_TP/s \leq M < (1+1/A_T)C_TP/s; \forall i\ r_i \leq P; s \leq P$

The number of PTEs required to reference all the elements is slightly bigger than the TLB, so there are some TLB misses. Let x be the number of TLB sets on which $(A_T + 1)$ PTEs are mapped; with the constraint that $0 \leq x < C_T/A_T$. The other TLB sets have just A_T references. We have $M = (A_T + 1)(P/s)x + A_T (P/s)(C_T/A - x) = xP/s + C_TP/s$, so $x = sM/P - C_T$. Substituting this value for x into the constraint $0 \leq x < C_T/A_T$ we find the boundary of this case: $C_TP/s \leq M < (1+1/A_T)C_TP/s$. Each set in x causes $A_T + 1$ misses during our access sequence of length M. The TLB miss ratio is $x(A_T + 1)/M = (s/P - C_T/M)(A_T + 1)$. Our offset sequences scatter the array data quite uniformly through the primary cache, so there are very few conflicts between PTEs and data array elements: all will be resident in the primary cache after the cold-start period. The average access time is $T_3 = T_0 + (s/P - C_T/M)(A_T + 1)T_{MissT}$, where T_{MissT} is the time needed to read one PTE into the TLB.

Case 3.3 $C_TP/s \leq M < (1+1/A_T)C_TP/s; \exists i\ (s + r_i) \geq (P + r_{i-1}); s \leq P$

This is similar to the previous case, except that at least one of the offsets r_i is large enough, in comparison to the previous offset r_{i-1}, that a page will be "skipped" in our reference sequence. This case arises only in our random offset sequence. The miss rate is difficult for us to analyse, as it depends on how many of the large offsets occur on the x TLB sets in which misses occur. Fortunately, we have no need of an exact formula for this case when analysing the results of our microbenchmark.

Case 3.4: $C_TP/s \leq M < (1+1/A_T)C_TP/s; s > P$

This is similar to Case 3.2, except that there is only one array reference per PTE in our TLB. We have $x = M - C_T$ and $T_3 = T_2 + (1 - C_T/M)(A_T + 1)T_{MissT}$.

Case 3.5: $M \geq C_T(P/s)(1+1/A_T); \forall i\ r_i \leq P_{size}; s > P$

Each element maps to a different TLB, and every line in the TLB is displaced before it can be re-used. Therefore the average access time is $T_3 = T_2 + T_{MissT}$.

Case 3.6: $M \geq C_T(P/s)(1+1/A_T); \exists i\ (s + r_i) \geq (P + r_{i-1}); s > P$

The variability in the offsets will decrease the miss rate observed in Case 3.5, but as in Case 3.3 we are unable to be precise in our analysis.

Case 3.7: $M \geq C_T(P/s)(1+1/A_T); s \leq P$

There are P/s consecutive accesses to each PTE. Each PTE misses on its first reference in the TLB: $T_2 + (s/P)T_{MissT}$.

Cases	Reference Count M	Stride s	Offset r_i	TLB Miss	Time per Iteration T_3
3.1	$1 \leq M < C_TP/s$	any	any	None	T_0
3.2	$C_TP/s \leq M < (1+1/A_T)C_TP/s$	$s \leq P$	$\forall i\ r_i \leq P$	Some	$T_0 + (s/P - C_T/M)(A_T + 1)T_{MissT}$
3.3	*Same as case 3.2*	$s \leq P$	$\exists i\ (s + r_i) \geq (P + r_{i-1})$	Some	$T_0 + z(s/P - C_T/M)(A_T + 1)T_{MissT}$
3.4	*Same as case 3.2*	$s > P$	any	Some	$T_0 + (1 - C_T/M)(A_T + 1)T_{MissT}$
3.5	$M \geq (1+1/A_T)C_TP/s$	$s > P$	$\forall i\ r_i \leq P$	Always	$T_0 + T_{MissT}$
3.6	*Same as case 3.5*	$s > P$	$\exists i\ (s + r_i) \geq (P + r_{i-1})$	Some	$T_0 + z(s/P - C_T/M)(A_T + 1)T_{MissT}$
3.7	*Same as case 3.5*	$s \leq P$	any	One per	$T_0 + (s/P)T_{MissT}$

Table 3. TLB miss patterns as function of $M \leq C_1/16$, stride s and offset r_i in the TLB, for incremented and random offset access patterns.

4 EXPERIMENTAL RESULTS

We have measured the structural and performance parameters of data cache memory and data TLB on our Pentium II/266 and Pentium III/500 workstations by running our micro benchmarks MBCP.

4.1 MEASUREMENT OF CACHE PARAMETERS

The upper curve in Figure 1 (at the end of this paper) shows our memory read access time per iteration on a PII/266 as a function of *4N* (the size of the accessed array, in bytes), for stride $s = 32$ bytes. We observed significantly faster accesses for $s = 16$. When $s = 64$, we see essentially the same data as for $s = 32$. This tells us that the blocksize $B_1 = 32$ bytes, and that the three linear portions of the upper curve in Figure 1 correspond to Cases 1.1, 1.3, and 1.5 in our analysis. Thus C_1 = 16KB, A_1 = 4-ways, the read latency of L_1 cache is 11 ns (3 machine clocks), and the read latency of L_2 cache is 60 – 11 = 49 ns (13 machine clocks). With other measurements (data not shown), we estimate the L_2 miss penalty (DRAM read time) as 230 – 60 = 170 ns (45 clocks), C_2 = 512 KB, B_2 = 32 bytes, and A_2 = 4 ways.

The lower curve in Figure 1 (along with our data for $s = 16$ and $s = 64$, not shown) tells us that our PIII/500 workstation has C_1 = 16KB, A_1 = 4-ways, $B_1 = 32$. The read latency of L_1 cache is 6 ns (3 machine clocks) and the read latency of L_2 cache is 44 – 6 = 38 ns (19 machine clocks). With other measurements (data not shown), we estimate the L_2 miss penalty is 140 – 44 = 96 ns (48 clocks), C_2 = 512 KB, B_2 = 32 bytes, and A_2 = 4 ways.

We also measure what we call the "effective data path parallelism" P_d, by constructing access sequences that allow a fixed degree of parallelism in an inner loop. For example, a 2-way parallelisable loop follows two pointer chains. We measure the average iteration time kT_k in a loop that makes k parallel accesses, for $1 \le k \le 6$, and we define $P_d = T_1/min(T_k)$. We find that $min(T_k)$ = 4.6 ns on the PII, and 2.5 ns on the PIII, so (taking T_1 from Figure 1), we see both CPUs have effective data path parallelism P_d = 2.5. In other words, they can run somewhat more than two accesses in parallel.

4.2 MEASUREMENT OF DATA TLB

Figure 2 shows our measurements of some parameters of the TLB on our PII/266 workstation. We show the curves for $s = 4096$ and $s = 8192$, for the incremented offset sequence, and for $s = 8192$ for the random offset sequence. When $s = 4096$, the random offset sequence gives almost identical results to the incremented offset sequence, which (according to Case 3.3) indicates that the page size $P = 4096$ bytes in our Linux environment. We characterize the remaining parameters of this TLB as follows: 64 PTEs, 4-ways, 11ns (3 machine clocks) hit time, and 30ns (8 machine clocks) miss time to L_1.

Similarly, from the data plotted in Figure 3, we characterize the DTLB of the PIII/500 as follows: 64 PTEs, 4-ways, 6ns (3 machine clocks) hit time, 16ns (8 machine clocks) miss time to L_1, 4KB page size under Linux.

5 CONCLUSION

Our measurement of cache memories and TLB not only reveal the most important structural parameters of cache memories and TLB, such as cache capacity, cache associativity, cache line size, TLB capacity and its associativity, but also provide some performance parameters. These performance parameters include the cache miss latency, minimum latency of TLB on a miss, and effective data path parallelism. In our experience, hardware manufacturers rarely provide these performance parameters.

We have presented experimental evidence of our ability to measure TLB and cache parameters for the PII/266 and the PIII/500, using our micro benchmark MBCP and the analytical approach described in this paper.

Further research could be directed to the refinement of our evaluation method and the extension of its scope to 1) "superpage" PTEs in TLB, 2) two-level TLBs of cache memory. A short-term improvement could be the transformation of our MBCP from a set of independent benchmarks to a standard C/C++ class library, so that algorithm designers, performance programmers and system administrators can use them. In addition, we might develop a commercial benchmark suite based on our MBCP.

6 REFERENCES

[1] Cragon, H.G., *Memory Systems and Pipelined Processors*, Jones and Bartlett Publishers, 1996.

[2] C. Steve, S.M. Kathryn and C.W. Tseng, "Compiler Optimizations for Improving Data Locality, in *Proc 6th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 252-262, 1994.

[3] Chilimbi, T.M., M.D. Hill and J.R. Larus, "Cache-conscious structure layout," in *Proc. 1999 ACM SIGPLAN Conf. on Programming Languages and Implementation (PLDI)*, 1-12, 1999.

[4] Hill, M.D. and A.J. Smith, "Evaluating associativity in CPU caches," *IEEE Transactions on Computers 38(12)*: 1612-1630, 1989. Correction in *41(3):* 371, 1991.

[5] LaMarca, A.G., *Caches and Algorithms*, Ph.D. dissertation, University of Washington, 1996.

[6] Li, E. and C. Thomborson, "Data cache parameter measurements," *Proc.IEEE Int'l Conf. on Computer Design (ICCD)*, 376-383, Oct. 1998.

[7] Lam, M.S., E.E. Rothberg and M.E. Wolf, "The cache performance and optimizations of blocked algorithms," in *Proc 4th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 63-74, 1991.

[8] Przybylski, S.A., *Cache and Memory Hierarchy Design: A Performance-directed Approach*, Morgan Kaufmann Publishers, Inc., 1990.

[9] Saavedra R.H. and A. J. Smith, "Measuring cache and TLB performance and their effect on benchmark running times," *IEEE Trans. Computers 44(10)*: 1223-1225, 1995.

[10] Pyo, C. and G. Lee, "Estimation of cache parameters based on reference distance (Summary)," Korea Electronics and Telecommunication Research Institute Project 96251, 1996.

[11] Xia, C., *Exploiting Multiprocessor Memory Hierarchies for Operating Systems*, Ph. D dissertation, University of Illinois at Urbana-Champaign, 1996.

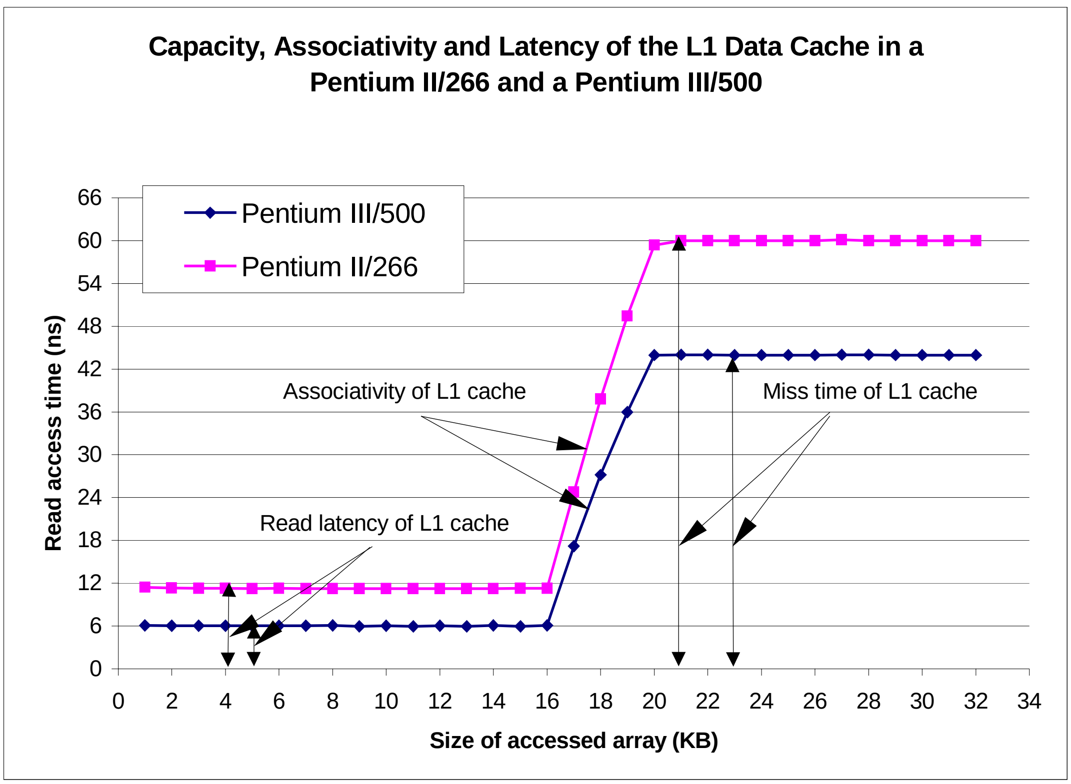

Figure 1 Measurement of primary caches on PII/266 and PIII/500

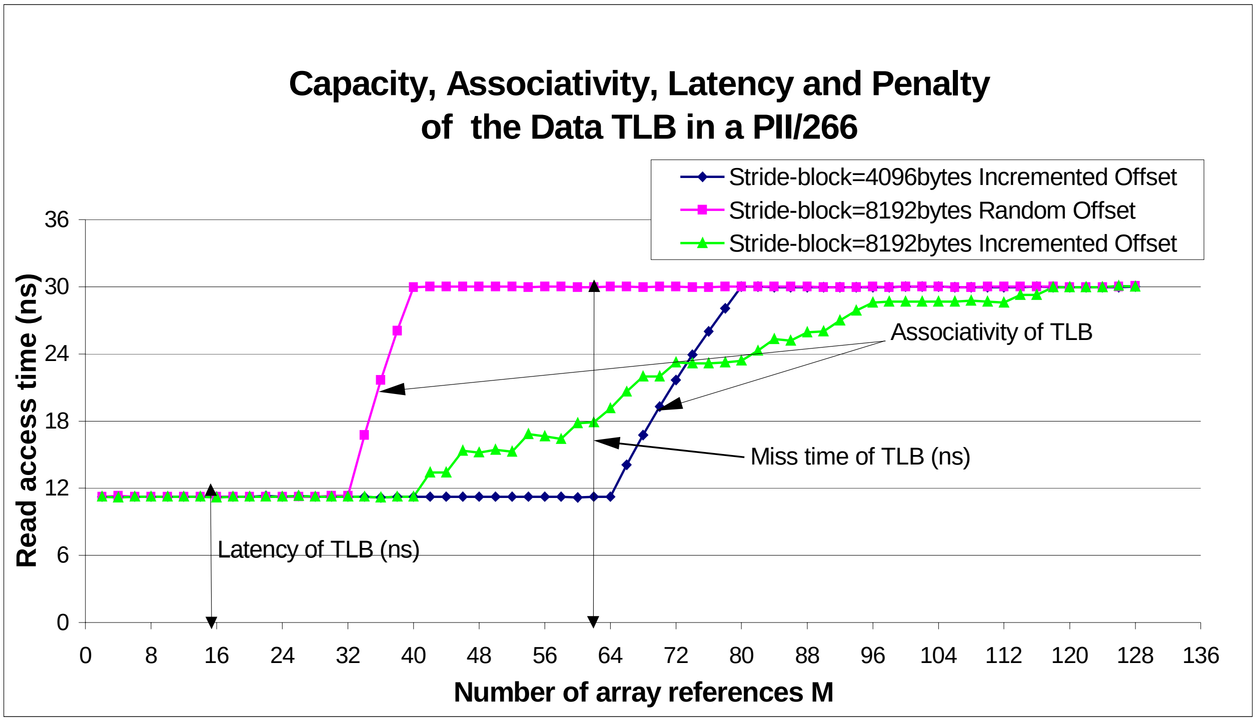

Figure 2. Measurement of the data TLB on our PII/266 Workstation

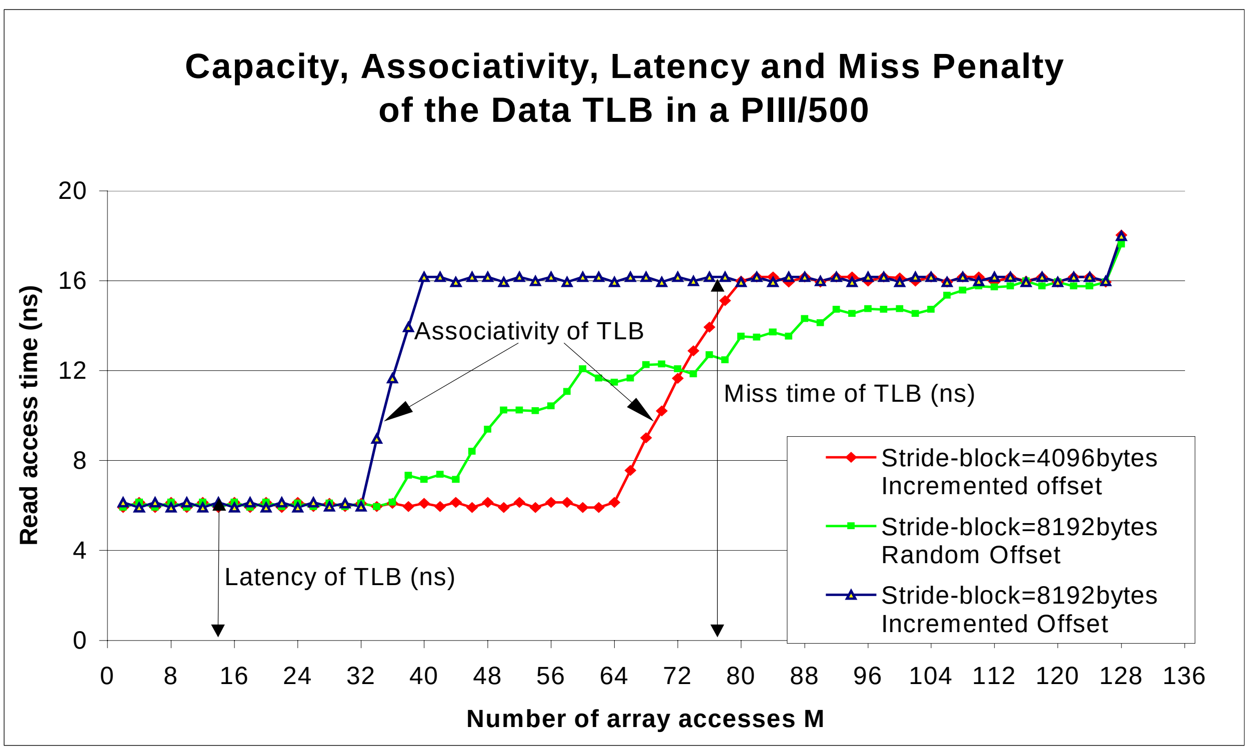

Figure 3 Measurement of data TLB on PIII/500 Workstation