Cache-Memory and TLB Calibration Tool

Stefan Manegold[1] Peter Boncz[2]

[1] *CWI*
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
S.Manegold@cwi.nl
http://www.cwi.nl/~manegold

[2] *Data Distilleries*
Kruislaan 402, 1098 SM Amsterdam, The Netherlands
P.Boncz@datadistilleries.com

1. Calibrating the Memory System

The idea underlying our calibrator tool is to have a micro benchmark whose performance only depends on the frequency of cache misses that occur. Our calibrator is a simple C program, mainly a small loop that executes a million memory reads. By changing the *stride* (i.e., the offset between two subsequent memory accesses) and the size of the memory area, we force varying cache miss rates.

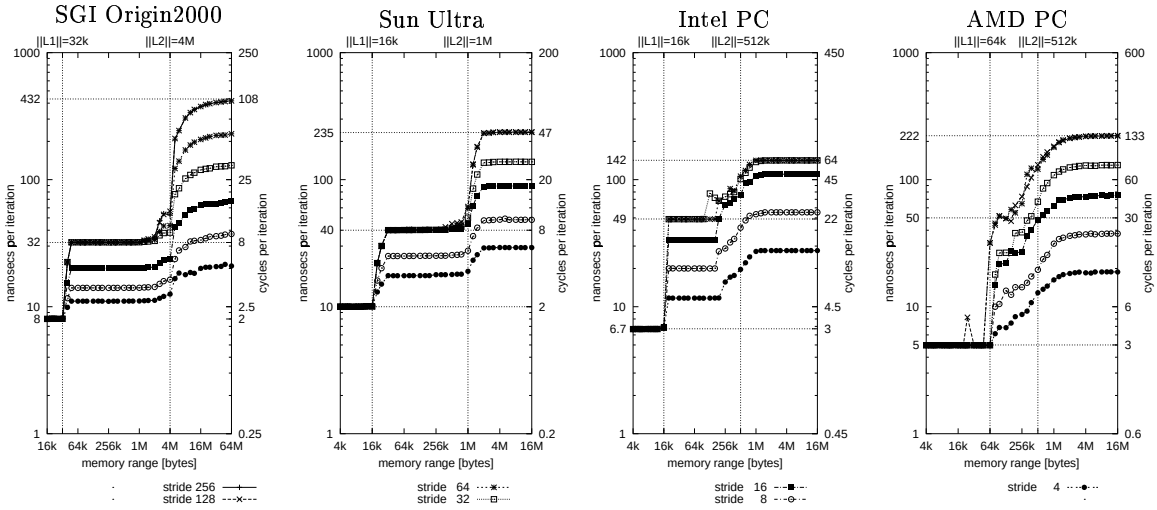
In principle, the occurance of cache misses is determined by the array size. Array sizes that fit into the L1 cache do not generate any cache misses once the data is loaded into the cache. Analogously, arrays that exceed the L1 cache size but still fit into L2, will cause L1 misses but no L2 misses. Finally, arrays larger than L2 cause both L1 and L2 misses.

The frequency of cache misses depends on the access stride and the cache line size. With strides equal to or larger than the cache line size, a cache miss occurs with every iteration. With strides smaller than the cache line size, a cache miss occurs only every n iterations (on average), where n is the ratio cache_line_size/stride.

Thus, we can calculate the latency for a cache miss by comparing the execution time without misses to the execution time with exactly one miss per iteration. This approach only works, if memory accesses are executed purely sequential, i.e., we have to ensure that neither two or more load instructions nor memory access and pure CPU work can overlap. We use a simple pointer chasing mechanism to achieve this: the memory area we access is initialized such that each load returns the address for the subsequent load in the next iteration. Thus, super-scalar CPUs cannot benefit from their ability to hide memory access latency by speculative execution.

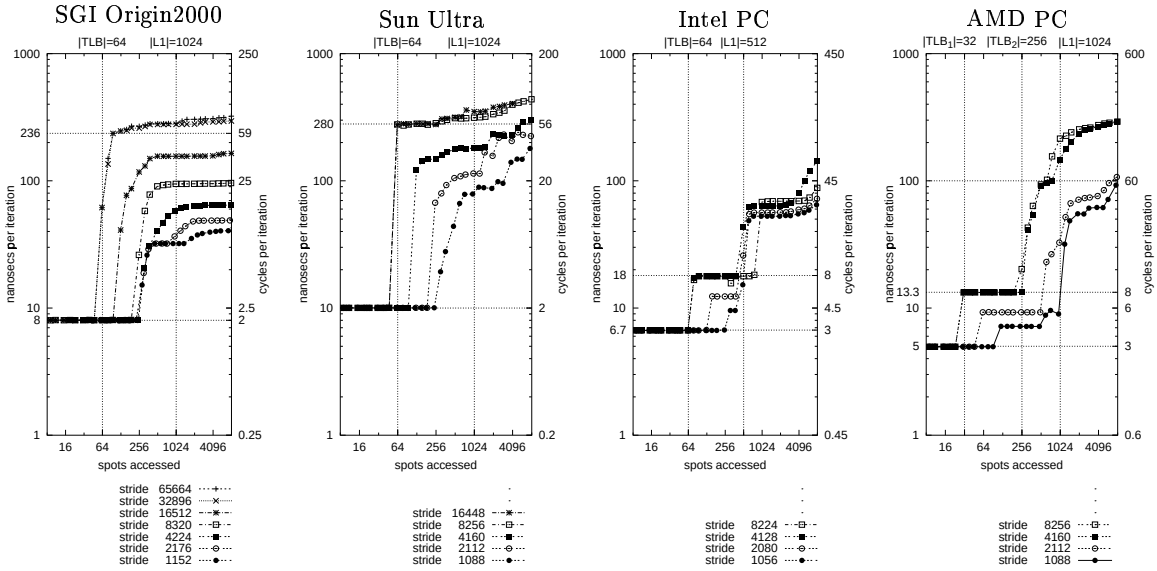
To measure the cache characteristics, we run our experiment several times, varying the stride and the array size. We make sure that the stride varies at least between 4 bytes and twice the maximal expected cache line size, and that the array size varies from half the minimal expected cache size to at least ten times the maximal expected cache size.

Figure 1 depicts the resulting execution time (in nanoseconds) per iteration for different array sizes on an SGI Origin2000 (MIPS R10000, 250 MHz = 4ns per cycle), a Sun Ultra (Sun UltraSPARC, 200 MHz = 5ns per cycle), an Intel PC (Intel PentiumIII, 450 MHz = 2.22ns per cycle), and an AMD PC (AMD Athlon, 600 MHz = 1.66ns per cycle). Each curve represents a different stride. All curves show two steps, indicating the existence of two cache levels and their sizes. Matching curves mean, that the cache miss frequency has reached its maximum (one miss per iteration), i.e., that the respective stride is equal to (or larger than) the cache line size.



(Vertical grid lines indicate cache sizes, horizontal grid lines indicate latencies.)

Figure 1: Calibration Tool: Cache sizes, line sizes, and latencies



(Vertical grid lines indicate number of TLB entries, horizontal grid lines indicate latencies.)

Figure 2: Calibration Tool: TLB entries and TLB miss costs

2. Calibrating the TLB

We use a similar approach as above to measure *TLB miss costs*. The idea here is to force one TLB miss per iteration, but to avoid any cache misses. We force TLB misses by using a stride that is equal to or larger than the systems page size, and by choosing the array size such that we access more distinct spots than there are TLB entries. Cache misses will occur at least as soon as the number of spots accessed exceeds the number of cache lines. We cannot avoid that. But even with less spots accessed, two or more spots might be mapped to the same cache line, causing *conflict misses*. To avoid this, we use strides that are not exactly powers of two, but slightly bigger, shifted by L2 cache line size.

	SGI Origin2000	Sun Ultra	Intel PC	AMD PC
OS	IRIX64 6.5	Solaris 2.5.1	Linux 2.2.12	Linux 2.2.12
CPU	MIPS R10000	Sun UltraSPARC	Intel PentiumIII	AMD Athlon
CPU speed	250 MHz	200 MHz	450 MHz	600 MHz
main-memory size	64 GB (4 GB local)	512 MB	512 MB	384 MB
L1 cache size	32 KB	16 KB	16 KB	64 KB
L1 cache line size	32 bytes	16 bytes	32 bytes	64 bytes
L1 cache lines	1024	1024	512	1024
L2 cache size	4 MB	1 MB	512 KB	512 KB
L2 cache line size	128 bytes	64 bytes	32 bytes	64 bytes
L2 lines	32,768	16,384	16,384	8192
TLB entries	64	64	64	32
TLB_2 entries	-	-	-	256
page size	32 KB	8 KB	4 KB	4 KB
TLB size	2 MB	512 KB	256 KB	128 KB
TLB_2 size	-	-	-	1 MB
L1 miss latency	24 ns = 6 cycles	30 ns = 6 cycles	42.2 ns = 19 cycles	45 ns = 27 cycles
L2 miss latency	400 ns = 100 cycles	195 ns = 39 cycles	93.3 ns = 42 cycles	172 ns = 103 cycles
TLB miss latency	228 ns = 57 cycles	270 ns = 54 cycles	11.1 ns = 5 cycles	8 ns = 5 cycles
TLB_2 miss latency	-	-	-	87 ns = 52 cycles

Table 1: Calibrated Performance Characteristics

Figure 2 shows the results for four machines. The X-axis now gives the number of spots accessed, i.e., array size divided by stride. Again, each curve represents a different stride. For the SGI and the Sun, the curves depict a single distinctive step, indicating a single TLB with 64 entries. The impact of L1 misses when more than 1024 spots are accessed is hardly visible as L1 miss penalty is small compared to TLB miss penalty. On the Intel PC, the first step relates to the 64-entry TLB and the second step relates to L1 misses, which are more expensive than TLB misses on the Intel PC. On the AMD PC, there are two TLBs with 32 and 256 entries, respectively. The third step in the curves again relates to L1 misses. The page sizes can be derived just like the cache line sizes before.

Table 1 gathers the results for all four machines. The PCs have the highest L2 access latencies, probably as their L2 caches are running at only half the CPUs' clock speed. Main-memory access, however, is faster on the PCs than it is on the SGI and the Sun. The TLB miss latency of the PentiumIII and the Athlon (TLB_1) are very low, as their TLB management is implemented in hardware. This avoids the costs of trapping to the operating system on a TLB miss, that is necessary in the software controlled TLBs of the other systems. The TLB_2 miss latency on the Athlon is comparable to that on the R10000 and the UltraSPARC.