Measuring The Cost Of A Cache Miss

Thomas R. Puzak, Allan Hartstein,
Philip E. Emma, Viji Srinivasan
IBM – T. J. Watson Research Center
Yorktown Heights, New York 10598
{trpuzak, amh, pemma, viji}@us.ibm.com

Abstract

It is vital that the cost of a cache miss be accurately measured in order for many hardware and software optimizations to occur. In this paper we describe a new technique, called pipeline spectroscopy, that allows pipeline delays to be monitored and analyzed in detail. We apply this technique to produce a cache miss 'spectrogram', which represents a precise readout showing a detailed histogram (visualization) of the cost of each cache miss. Cache miss spectrograms are produced by comparing instruction sequences and execution times that occurred near a miss in a 'finite cache' simulation run to the same set of instructions and execution times in an 'infinite cache' run. Cache misses are divided into clusters, and the miss penalty associated with each cluster is determined by identifying an upper and lower bound instruction around each miss cluster and calculating the cycle difference between these bounds. Detailed analysis of a spectrogram leads to much greater insight in pipeline dynamics, including effects due to miss cluster, miss overlap, prefetching, and miss queueing delays.

1. Introduction

The rapid pace of technology improvements (both in speed and circuit density) has seen the internal organization of a processor become more complex. In today's processors the pipelines are faster, deeper, and wider than ever before. However, the relative speed of memory has not kept pace with the processor's frequency (cycle time) and so a proportionally greater percent of a program's performance is lost due to cache misses. To reduce the time lost to cache misses, designers have added many levels of caches. It is common for a design to have two or three levels of cache between the processor and memory.

In order to improve the performance of a processor or an application, designers have increased the amount of parallelism between the levels of the memory hierarchy. This area of research, termed memory-level-parallelism (MLP), has been explicitly studied in [1, 2, 3], while early studies focused on modeling and evaluating performance with ILP processors [4, 5, 6]. Most recently, Chou et al. [7, 8] studied several techniques (out-of-order, runahead, value prediction, prefetching, store handling optimization) for increasing MLP in applications that are dominated by memory delays. They show that substantial amounts of performance gains are possible by increasing the MLP in these applications.

In this paper, we build on this work by describing a new technique to measure the amount of parallelism between the different levels of the memory hierarchy and describe a mechanism for displaying images that permits the visualization of the cost of each cache miss. We call this new technique 'pipeline spectroscopy' and the graphs representing the miss cost a 'spectrogram'. The graphs are called spectrograms because they reveal certain signature features of the processor's memory hierarchy, the pipeline, and the miss pattern itself (amount of overlap or parallelism that exists between each level of the memory hierarchy). Using pipeline spectroscopy, we are able to measure the rate that misses are satisfied from the different levels of the memory hierarchy and quantify the cost of each cache miss, with and without overlap. This quantification leads to a much greater understanding of the amount of parallelism or overlap that the micro architecture and application allow while a miss is in progress.

Several mechanisms that measure the cost of a miss are described in the patent literature [9-18]. Most embodiments describe the difficulty in determining an accurate measure for the cost of the miss and rely on hardware monitors to count events (cycles) that indicate when the decoder or execution unit is delayed (stalled) while waiting for an operand (data) to estimate this cost. However, not all of these events contribute to the loss of performance in a program. Today's processors have superscalar capabilities and parallel execution paths and a delay suffered in one component of a processor can be overlapped with other events to mask any loss due to the miss. For example, consider two events occurring in parallel: a branch miss-prediction and a cache miss. Simply counting the number of cycles an instruction (in the decoder or execution unit) is stalled waiting on a miss is not an accurate measure of the cost of the miss since many of the stall cycles are already overlapped with the delays caused by the branch miss-prediction.

We apply pipeline spectroscopy to produce a cache miss spectrogram which represents a precise readout showing a detailed histogram (visualization) of the cost of each cache miss, with and without overlap. Cache miss spectrograms are produced by comparing instruction sequences and execution times that occurred near a miss in a 'finite cache' simulation run to the same set of instructions and their execution times in an 'infinite cache' run. Cache misses are divided into clusters, and the miss penalty associated with each cluster is determined by a two step process. First, an upper and lower bound sequence of instructions is identified that bounds each miss. Second, the cost of the miss cluster is the difference (in time) between the finite cache and infinite cache execution time of the instruction sequence.

The rest of this paper is organized as follows: Section 2 contains definitions and terminology. Section 3 describes the process used to construct a miss spectrogram, and the simulation model is described in Section 4. In Section 5 we measure the cost of a data miss and in Section 6 we describe an instruction miss spectrogram. Section 7 shows how bus queueing changes the shape of a spectrogram. Summary and conclusions are discussed in Section 8.

2. Performance Terminology

The overall methodology used to calculate the cost of a miss and the visualization process are explained as a prelude to analyzing a miss spectrogram. First, the definitions and formulas used to calculate the cost of a miss are described, then a description is set forth relative to how misses cluster and affect the standard operation of a high performance processor, followed by a description of the visualization process.

The most commonly used metric for processor performance is, "Cycles Per Instruction" (*CPI*). The overall *CPI* for a processor system has two components: an "infinite cache" component (CPI_{INF}) sometimes called an ideal cache, and a "finite cache adder" (CPI_{FCA}).

$$CPI_{Overall} = CPI_{INF} + CPI_{FCA} \quad (1)$$

CPI_{INF} represents the performance of the processor in the absence of misses, even compulsory (cache, and TLB). It is the limiting case in which all memory operations are assumed to hit in the first-level cache and is a measure of the performance of the processor's organization with the memory hierarchy removed. CPI_{FCA} accounts for the delay due to cache misses and is used to measure the effectiveness of the memory hierarchy.

The CPI_{FCA} term can be expressed as the product of an event rate (specifically, the miss rate), and the average delay per event (cycles lost per miss):

$$CPI_{FCA} = \left(\frac{Misses}{Instruction}\right)\left(\frac{Cycles}{Miss}\right) \quad (2)$$

Substituting for CPI_{FCA} in (1), the overall performance for a processor can be expressed as:

$$CPI_{OVERALL} = CPI_{INF} + \left(\frac{Misses}{Instruction}\right)\left(\frac{Cycles}{Miss}\right) \quad (3)$$

By rearranging this formula, the average cost of a cache miss can be calculated. That is,

$$\left(\frac{Cycles}{Miss}\right) = (CPI_{OVERALL} - CPI_{INF})\left(\frac{Instruction}{Miss}\right) \quad (4)$$

We use this formula to calculate the amount of time (cycles) a processor loses due to each cache miss. The following example illustrates calculating cycles per miss using equation (4). Consider an application whose entire run length is one million instructions and a processor where each cache miss is satisfied from the L2 with a 20 latency. If an infinite cache simulation run takes one million cycles ($CPI_{INF} = 1$), and a finite cache simulation run takes 1.3 million cycles, then the cache misses account for an additional 300,000 cycles and the

total CPI = 1.3 and CPI_{FCA} = .3. If the finite cache simulation run has 25,000 misses, then

$$\left(\frac{Misses}{Instruction}\right) = \frac{25{,}000}{1{,}000{,}000} = \frac{1}{40}$$

and

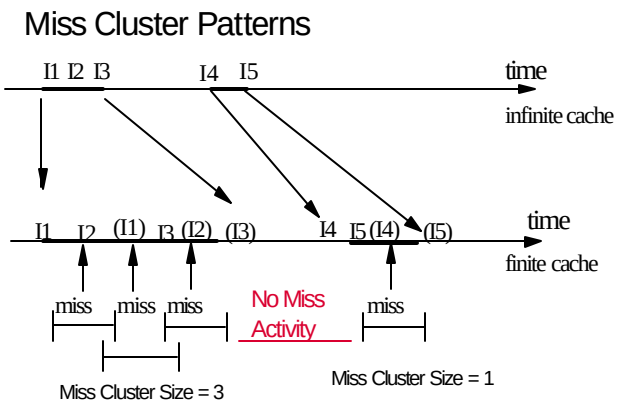
$$\left(\frac{Cycles}{Miss}\right) = \frac{300{,}000}{25{,}000} = 12.$$

By applying this equation over the entire length of an application, the average cost for all misses can be calculated.

In the example above, we applied (4) macroscopically to calculate the average cost of a miss over the total run time of an application. However, equation (4) can also be used microscopically to calculate the cost of a single miss. We will take a microscopic approach in using equation (4) to produce a miss spectrogram.

3. Constructing a Miss Spectrogram

A description of how misses can cluster and affect the performance of a processor is now described. Figure 1 shows the same five instructions executed as both an 'infinite cache' sequence of instructions and a 'finite cache' sequence of instructions. In the finite cache sequence, the instruction decode times are shown in bold and instruction completion or EndOp times are shown in parenthesis. In the infinite cache run only the instruction decode times (in bold) are shown.



Miss Cluster Size = Number of Misses During Miss Facility Busy Interval

Figure 1. Miss Cluster Patterns for an Application, Cluster Sizes of 1 and 3 Misses are shown

Associated with the finite cache run are two miss clusters, where a 'miss cluster' is a continuous interval of time characterized by at least one miss in progress at all times. The size of the miss cluster is the number of misses that started during this interval. In the finite cache run, the first miss cluster has three misses with overlap (size = 3) and the second miss cluster shows a miss in isolation (size = 1). The time to process the first miss cluster (in the finite cache run) is bounded by the decode time for instruction I1 and the EndOp time of I3, $(I3EndOp - I1Decode)_{Finite\ Cache}$ time. Instruction I1 represents the greatest lower bound of the miss cluster, while instruction I3 is the least upper bound of the cluster. We call these points (instructions) the infimum and supremum of the miss cluster. (By convention, the infimum of a miss cluster is the greatest instruction that decoded just prior to the beginning of the first miss in the miss cluster and the supremum is the first instruction that completed (EndOp) just after the last miss in the miss cluster finished.) Similarly, the infimum instruction of the second miss cluster is I4 and the supremum is I5. The time to process the second miss cluster is then $(I5EndOp - I4Decode)_{Finite\ Cache}$. To calculate the amount of delay associated with the first miss cluster we must subtract the amount of time to process the same set of instructions in an infinite cache run from the finite cache run. That is, $[(I3EndOp - I1Decode)_{Finite\ Cache} - (I3EndOp - I1Decode)_{Infinite\ Cache}]$ equals the number of cycles the pipeline was stalled due to the first miss cluster. Similarly, the amount of delay associated with the second miss cluster is $[(I5EndOp - I4Decode)_{Finite\ Cache} - (I5EndOp - I4Decode)_{Infinite\ Cache}]$.

By applying the above technique repeatedly, we can calculate the cost of a miss or miss cluster for both in-order and out-of-order processors. In the example above, I1, I2, and I3 can even be from three different threads running on a multithreaded processor (or three out-of-order instructions), but as long as the same three instructions (and their order from the finite cache run) are used to determine the infinite cache run time, the cost of the miss cluster can be determined.

There are certain boundary conditions that must be considered when determining the infimum and supremum of a miss cluster. For example, the infimum of a miss cluster can only be established after the supremum of the previous miss cluster has been determined. This ensures that one miss cluster is terminated before another starts. If the upper and lower bounds of a miss cluster cannot be uniquely established, the two adjoining miss clusters are combined into a larger miss cluster.

Also, when determining the infinite cache running time for an instruction sequence that occurred during a miss cluster, it may be necessary to prime the processor's pipeline with some of the instructions that

occurred prior to the infimum instruction. This ensures that the correct execution and EndOp times of the infimum instruction are preserved as it passes through the processor's pipeline. By grouping misses according to their cluster size and calculating the delay associated with a miss cluster (number of stall cycles) using the method described above, the amount of time a processor loses due to cache misses is produced.

4. Simulation Methodology

To date, pipeline spectroscopy has been implemented in three proprietary processor simulators. Each timer has produced results similar to those shown in Sections 5 and 6 below. Each timer is cycle accurate and has been thoroughly validated against existing hardware. The processor model used in this paper, shown in Figure 2 and described in [19, 20], is a 4 issue superscalar, with address generation and cache access as independent out-of-order processes.

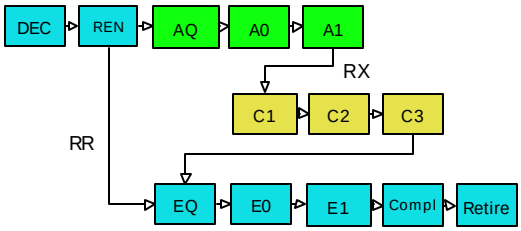


Fig. 2 shows the pipeline modeled in this study. The stages include: Decode, Rename, Agen Q, Agen, Cache Access, Execute Q, Execute, Completion and Retire.

The instruction window was set at 32 entries. Separate L1 instruction and data caches were modeled at 64 KB, the L2 size varies from 256K to 1 MB, and the L3 (when modeled) was varied from 1 MB to 4 MB. This processor model was chosen to illustrate the technique used to construct a spectrogram, and does not represent any existing or planned processor design. In our initial studies, execution and completion are in-order. Future work is planned to measure the benefits of out-of-order execution, prefetching, SMT, and SMT with out-of-order execution.

We use instruction traces produced for the IBM zSeries (S/390) processor family. In order to stress different levels of the memory hierarchy (L1, L2 or L3), we use applications with large instruction and data footprints capable of stressing caches up to 4 Megabytes. Typically, commercial database applications have these characteristics [7]. In our study, we use six workloads drawn from database workloads, SPEC 2000, and a C++ application. We use three proprietary commercial database applications running on zSeries servers, (oltp described in [21, 22] and oltp2, and oltp3); mcf, from SPEC 2000, SPECjbb 2000, and perf1 [21, 22] a large-processor simulator written in C++. Typically, trace lengths are 5 to 100 million instructions. The simulation environment can handle the entire SPEC suite; the application subset used for this work was chosen for its ability to stress L2 and L3 cache usage.

5. Miss Spectrogram For Data Misses

In order to examine the miss spectrogram for data misses alone, we model an infinite or perfect instruction L1 cache, and set the data L1 cache to 64KB. The L2 is set to 256KB with a 15 cycle latency, and set L3 latency to 100 cycles. All L2 misses are resolved in the L3 (infinite L3). The line size and bus width are set at 128 bytes. No data prefetching was modeled. In fact, data prefetching is very difficult for many of these applications. However, prefetching will be explored when we examine the instruction miss spectrogram.

Using the techniques described above, Figure 3 shows the cache miss spectrogram for the oltp workload. The overall hit ratio of the L2 was approximately 50%. The miss spectrograms for cluster sizes = 1, 2, 3, and 4 are shown. The X axis represents the cost of the miss for a given miss cluster size. The Y axis shows the percent of misses that had that delay.

The cluster =1 plot (in Figure 3) shows two peaks. The first peak is centered near 15 cycles (the L2 miss latency), and the second peak is near 100 cycles (the L3 miss latency). The integral of the areas under each peak is approximately the percentage of L1 misses resolved in the corresponding level of the memory hierarchy (i.e., the hit rates for the L2 and L3, or 50% in each).

The cluster size = 2 plot shows peaks at 15 and 30, 100 and 115, and 200 cycles. Integrating the area under each peak, we see that the five miss clusters, costing 15, 30, 100, 115 and 200 cycles, have a probability of .138, .168, .288, .191, and .215, respectively. The peaks at 15 and 30 represent two L1 misses that both hit in the L2 but highlight two distinctively different outcomes. In the first case (peak at 15), both misses had a high degree of overlap (MLP) and the overall cost was approximately the L2 miss latency while in the second case there was little overlap and the cost of the miss cluster was the sum of two L2 hits.

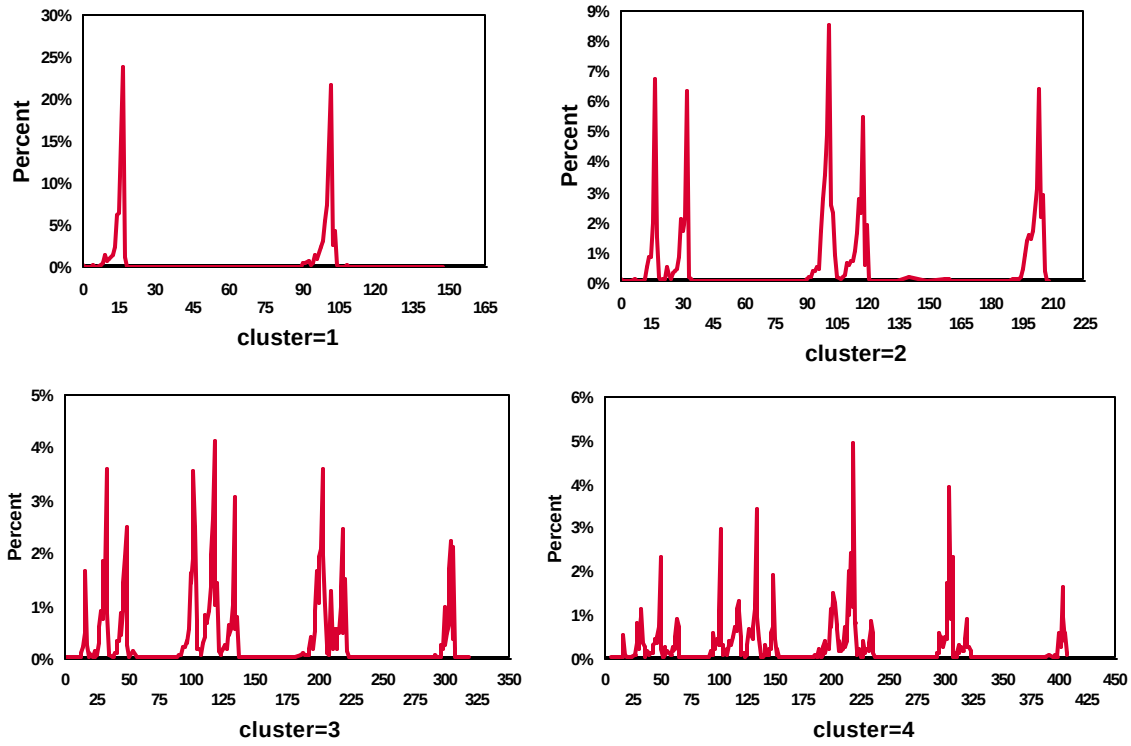


Fig 3, Miss Spectrogram for OLTP, L1=64K, L2=256K 15 Cycles Latency, Infinite L3 100 Cycles Latency

The peak at 100 again identifies two misses that were overlapped (had a high degree of MLP). Whether it was two misses that hit in the L3, one miss that hit in the L2 and one that hit the L3, the overall cost of the misses in the cluster was just the L3 miss latency.

The peak at 115 identifies two misses that had little or no overlap. Here, one miss hit in the L2 and one miss hit in the L3 but the cost of the miss cluster was the sum of the individual miss latencies. Finally, the peak at 200 identifies two misses that were resolved in the L3 and there was little overlap.

The peaks in the cluster = 3 graph represent all of the hit/miss combinations (with and without overlap) of length 3 using the two miss latencies (15, 100) for the L2, and L3. For example, the peaks at 15, 30, and 45 present three L2 hits where two misses were overlapped, one miss was overlapped or no miss was overlapped with the other misses in the cluster. However, the peak at 300 represents three L3 hits with little overlap. Obviously, three dependent misses that are resolved in the L3 can cause this miss penalty. Finally, the peaks in the cluster = 4 graph show all of the hit/miss, overlap/no-overlap, combinations of length 4 using the miss latencies 15 and 100.

Each peak represents the amount of time the group of cache misses (cluster) stalled the pipeline. By summing the 'stall cycles' calculated for each miss cluster, we can reconstruct the finite-cache-adder for the entire run, one cluster at a time. In many cases this involves summing the delay associated with 10s of thousands to over 100,000 miss clusters. Using this technique, we have always been able to reconstruct, within 5%, the true finite-cache-adder for a run.

Figure 4 plots cluster size versus the amount of misses that occurred in that cluster. Even though the maximum miss cluster for the run was well over 1000 misses, typically the average miss cluster size is much smaller. For example, over 80% of the misses occur to

Figure 4. Misses by Cluster Size

miss clusters of size 6 or less. This was observed for most applications used in this study.

Prefetching and bus delays can change the shape of the peaks in a spectrogram. Prefetching can broaden the left shoulder of any peak and reduce miss latency. Queueing and bus delays can increase the right shoulder of a peak, adding miss latency. These features will be explored below. Each spectrogram has enormous value to hardware and software designers by identifying potential performance problems associated with the processor's hardware or the application's software.

Next, we repeat the above experiment but use the oltp2 workload with the following memory hierarchy: data L1=64KB, L2=256KB 15 cycle latency, L3=1MB 75 cycle latency, and 300 cycle memory latency. Figure 5 shows the data miss spectrogram for cluster sizes = 1, 2, and 3.

The cluster = 1 plot shows three peaks. The first peak is centered near 15 cycles (the L2 miss latency), the second peak is near 75 cycles (the L3 miss latency) and the third peak at 300 cycles (the memory latency). Integrating the area under each peak is approximately the hit ratio (regarding L1 misses) for that level of the memory hierarchy (i.e. the hit ratios for the L2, L3, and memory). Examining the plots for cluster sizes equal 2 and 3, we see that they show all of the hit/miss, overlap/no-overlap combinations of length 2 and 3 using the 3 miss latencies: 15, 75 and 300. Obviously the peak at 15 for the cluster size = 3 indicates a great deal of overlap among the three misses. However, the peak at 390 (for cluster size =3) indicates very little overlap among the three misses. Here, one miss was resolved in the L2, one in the L3, and one went to memory but the total miss penalty for the whole cluster is the sum of the individual miss latencies (15, 75, and 300). Similar hit/miss patterns, and overlap/no-overlap conclusions can be drawn for examining any peak in the miss spectrogram.

We will refer to spectrograms like the one shown in Figure 5 as the 'canonical' representation for the cost of a miss in a multilevel memory hierarchy. It is a canonical form because it represents the most general form (combinations) of the miss patterns in a memory hierarchy. Obviously, prefetching and bus queueing can alter the miss patterns, and costs. By including the possibility of a peak at zero, a miss spectrogram can have all possible combinations of the miss latencies from each level of the memory hierarchy for a given cluster size. A peak at zero has the physical meaning that a miss or cluster of misses has zero delay. Prefetches, if issued far enough in advance of their use, speculative misses that do not interfere with any other cache accesses, or unused prefetches (in rare cases) have the possibility of causing zero delay.

Each of the spectrograms above aids the

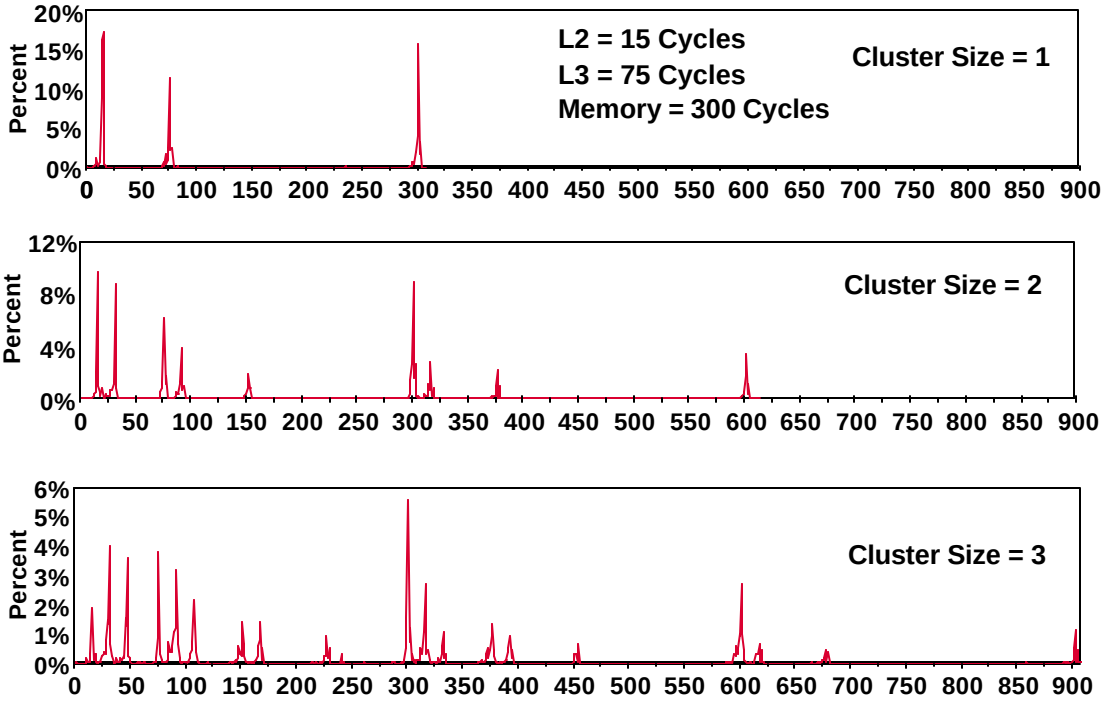


Figure 5. Data Miss Spectrogram for OLTP2, L1=64K, L2=256KB 15 Cycle Latency, L3=1Meg 75 Cycle Latency, Memory = 300 Cycle Latency

hardware/software designer by measuring the cost of a miss and by identifying potential performance bottlenecks. For example, merely identifying that an instruction is always causing a cache miss is not always sufficient to identify a performance problem. Consider a processor with the following memory hierarchy: L1, L2, and memory with latencies of 15 cycles for an L2 hit. Now consider a three miss cluster where all three misses are resolved in the L2 (L2 hits). If a software designer identifies that an instruction is always causing a miss and the cost of the miss cluster is 15 cycles, there is probably little benefit in removing (improving) the miss latency associated with those instructions since two out of the three misses are overlapped. However, if the cost of the miss cluster is 45 cycles, then very little of the miss latency is overlapped and it is probably worth the effort to investigate the source code to improve performance. Pipeline spectroscopy gives this information.

6. Miss Spectrogram For Instruction Misses

In the next set of experiments, we set the instruction cache to 64KB, the L2 to 256KB with a 15 cycle latency, the L3 to 1MB with a 75 cycle latency, and the memory to a miss latency of 300 cycles (the compliment of the experiment shown in Figure 5). Instruction fetching is guided by a 32K-entry branch target buffer (BTB) that runs well ahead of instruction fetching and the decoder, predicting branches and aiding prefetching. The benefits of using a BTB to guide instruction fetching and prefetch instructions have been well documented [23-30].

Figure 6 shows the instruction spectrogram for cluster sizes = 1, 2, 3, and 4 for oltp. These spectrograms are different from their data counterparts. The spectrograms for cluster sizes 1, 2, and 3 have dominant peaks at 0, 15, 75, and 300. The amount of overlap, in the cluster = 2 and 3 spectrograms, is substantial. The cluster size = 2 spectrogram is still dominated by peaks at 15, 75 and 300, even though there are two misses. Obviously, one of the misses is overlapped with the first. In the cluster size = 3 spectrogram, there are four non-zero dominate peaks: 15, 30, 75, and 300. Here we have three misses but only the peak at 30 indicates that two out of the three misses were not overlapped. In the other cases (peaks at 15, 75, and 300), the cost of the miss cluster is approximately the cost of a single miss (either to the L2, L3, or memory) and two out of the three misses were overlapped with the first. Recall that the data spectrograms for these cluster sizes had miss penalties extending to 600 and 900 cycles, respectively. Even the cluster size = 4 spectrogram has substantial peaks at 15, 75, and 300 cycles, indicating all misses were overlapped. Analyzing the peak at 150 indicates two of the four misses were resolved in the L3 and not overlapped, while the other two misses were overlapped. Similarly, the peak at 375 indicates one miss went to memory and the other was resolved in the L3 (the other two misses were overlapped).

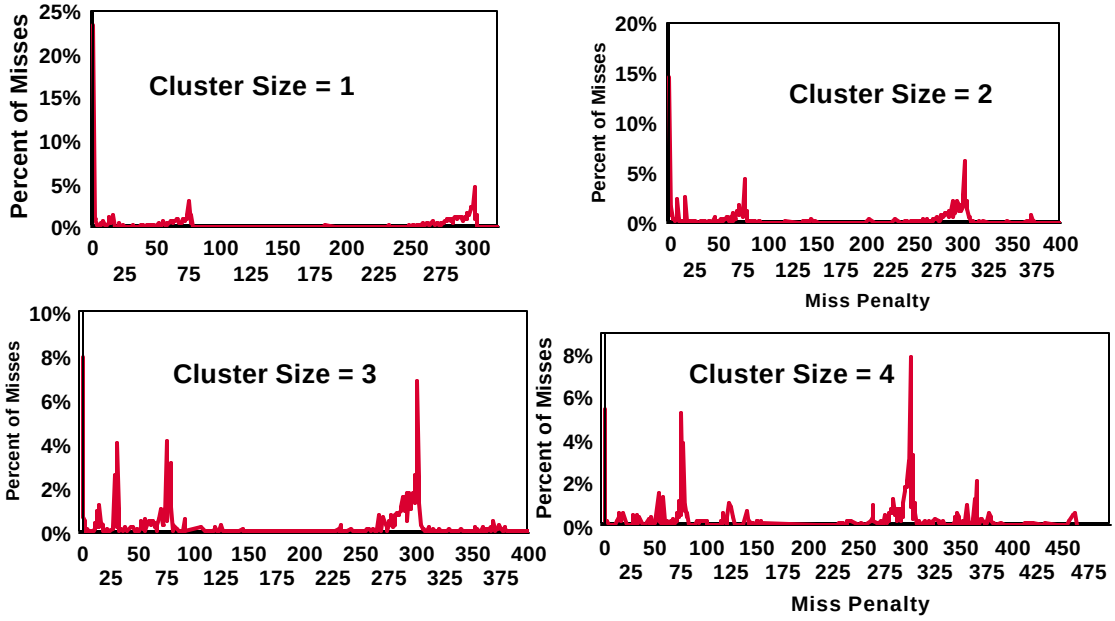


Figure 6. Instruction Miss Spectrogram. L1=64KB, L2=256KB 15 Cycle Latency, L3=1MB 75 Cycle Latency, Memory = 300 Cycle Latency

The spectrograms also show that the BTB is able to substantially reduce the cost of a miss through prefetching. Notice the large left shoulders on each of the peaks in all the spectrograms. Additionally, each spectrogram has a large peak at 0 indicating that the BTB was able to prefetch instruction misses far enough in advance of their used to hide all of the miss latency. For example, the cluster = 4 spectrogram shows approximately 6% of the miss clusters had zero cycles penalties. Here we have four misses that occur in parallel, but yet the BTB was able to prefetch them far enough in advance of their use to cause zero cycles difference between the finite cache simulation run and an infinite cache simulation run.

In Figure 7, we change the range of the X and Y axis of the cluster = 1 spectrogram to emphasize the left shoulder of the 15 and 75 peaks. We see that the left shoulder of the peak beginning at 75 cycles extends to nearly 20 (indicating that prefetches are issued nearly 55 cycles in advance of their use).

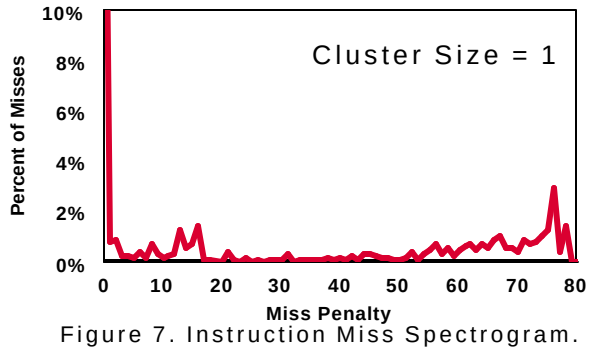


Figure 7. Instruction Miss Spectrogram. Data for OLTP Workload

7. Bus Queueing and Spectrogram Shape

In our last experiment we highlight the effects bus queueing has on the shape of a spectrogram. We repeat the experiment shown in Figure 3 but change the memory latency to 50 cycles and model a 16 byte bus that transfers a packet of information every other cycle, as opposed to the optimistic 128B bus transfer used previously. Now, it takes 16 cycles to transfer a line between the caches. Figure 8 shows the spectrograms for clusters = 1, 2 and 3. Notice the effects that bus queueing have on the shape of the right shoulder.

In the cluster = 1 spectrogram, we clearly have peaks centered at 15 and 50 cycles, but the peaks are not as sharp as when the line was transferred in one cycle. However, in the cluster = 2 spectrogram, the original five peaks shown in Figure 3 broaden and merge. Now, there are three large peaks, (not five as seen earlier) with one peak between 30 and 60, one between 60 and 80, and one between 80 and 120 with a large right shoulder. Obviously, there are four hit/miss combinations that describe the L2 hit/miss patterns but the cost of each miss cannot be identified as clearly as when the bus transfer interval was 1 cycle. Finally, in the cluster = 3 spectrogram, all three peaks start to merge between a range of 30 to 180. Clearly, queueing is increasing the cost of each miss or mis s cluster well beyond the miss latency.

8. Summary

A new technique has been presented for calculating the cost of a miss and displaying images

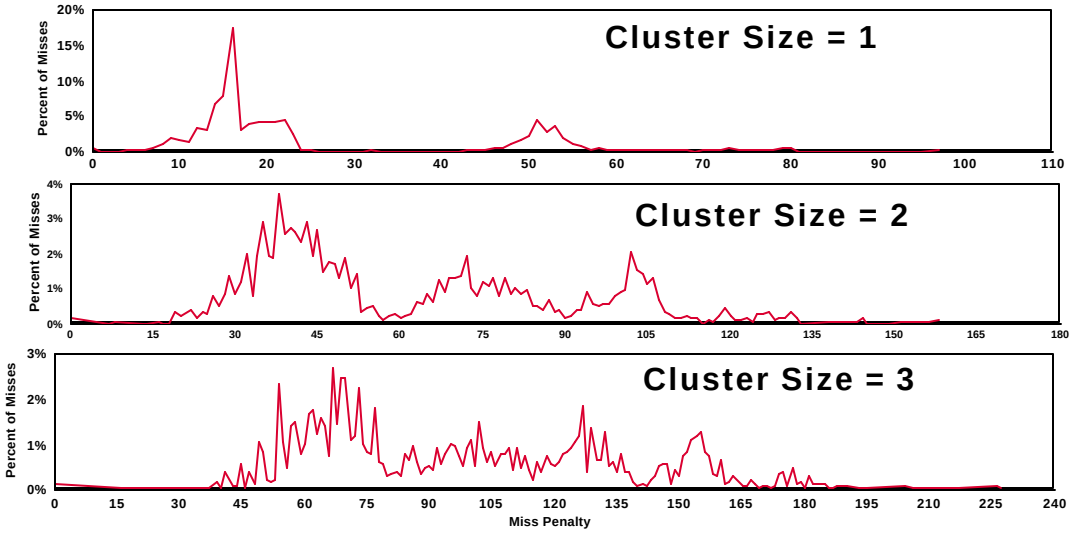


Figure 8. Data Miss Spectrogram.L1=64KB, L2=256KB 15 Cycle Latency, L3= 50 Cycle Latency, Line Transfer Interval= 16 Cycles

that represent their cost. We call this technique pipeline spectroscopy. The underlying principles of this technique are very simple: the cost of a miss can be determined by knowing the finite cache and infinite cache execution times for the same sequence of instructions. The difference between these two times is the cost of the miss (cluster).

We use this principle to produce a miss spectrogram, which represents a precise readout of the cost of every miss. A miss spectrogram has enormous value in analyzing the performance of an application or microarchitecture. Detailed analysis of a spectrogram leads to insights in pipeline dynamics, including effects due to prefetching, bus queueing, and underlying architectural features that allow or inhibit memory level parallelism.

In this study, we have demonstrated that pipeline spectroscopy can be used to explore the amount of miss processing parallelism that exists between each level of the memory hierarchy. Future work is needed to study in-order versus out-out-order effects on miss parallelism, as well as SMT processor organizations, in addition to the shape of a spectrogram (left and right shoulders), and positions of each peak (and sub-peak).

9.0 References

[1] A. Glew, "MLP yes! ILP no!," in ASPLOS Wild and Crazy Ideas Session , October 1998.

[2] V. Pai and S. Adve, "Code Transformations to Improve Memory Parallelism," in 32nd International Symposium on Microarchitecture, November 1999.

[3] H. Zhou and T. Conte, "Enhancing Memory Level Parallelism via Recovery-Free Value Prediction," in International Conference on Supercomputing, June 2003.

[4] D. Sorin et al, "Analytic Evaluation of Shared-Memory Systems with ILP Processors," in 25th International Symposium on Computer Architecture, 1998.

[5] V. Pai, P. Ranganathan and S. Adve, "The Impact of Instruction- Level Parallelism on Multiprocessor Performance and Simulation Methodology," in International Symposium on High Performance Computer Architecture, February 1997.

[6] P. Ranganathan, K. Gharachorloo, S. Adve and L. Barroso, "Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors," in ASPLOS-VIII, 1998.

[7] Y Chou, B. Fahs, and S Abraham, "Microarchitecture Optimizations for Exploiting Memory-Level Parallelism Exploiting Memory-Level Parallelism" in 31st International Symposium on Computer Architecture, 2004.

[8]Yuan Chou, Lawrence Spracklen, Santosh G. Abraham. "Store Memory-Level Parallelism Optimizations for Commercial Applications," pp. 183-196, 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05), 2005.

[9] A. Zahir, V. Hummel, M. Kling, T Yeh, US. Patent 6,353,805 B1, "Apparatus and Method for Cycle Accounting in Microprocessors" Assigned to Intel Corporation.

[10] A. Zahir, V. Hummel, M. Kling, T Yeh, US. Patent 6,353,802, "Apparatus and Method for Cycle Accounting in Microprocessors" Assigned to Intel Corporation.

[11] B. Gaither, R. Smith, US Patent 6,892,173 B1, "Analyzing Effectiveness of a Computer Cache By Estimating a Hit Rate Based on Applying a Subset of Real-time Addresses to a Model of the Cache", Assigned to Hewlett-Packard Development.

[12] H. Ravichandran, US Patent 6,341,357 B1, "Apparatus and Method for Processor Performance Monitoring", Assigned to Sun Microsystems.

[13] R. Trauben, US Patent 5,594,864, "Method and apparatus for unobtrusively monitoring Processor States and Characterizing Bottlenecks in a Pipeline Processor Executing Grouped Instructions", Assigned to Sun Microsystems.

[14] G. Brooks, US Patent 6,081,868, "System and Methods For Performing Cache Latency Diagnostics in Scalable Parallel Processing Architectures Including Calculating CPU Idle Time and Counting Number of Cache Misses, Assigned to Hewlett-Packard Company.

[15] G. Brooks, US Patent 5,845,310 "System and Methods For Performing Cache Latency Diagnostics in Scalable Parallel Processing Architectures Including Calculating CPU Idle Time and Counting Number of Cache Misses., Assigned to Hewlett-Packard Company.

[16] W. Flynn, US Patent 6,256,775 B1, "Facilities For Detailed Software Performance Analysis in a Multithreaded Processor", Assigned to IBM.

[17] F. Levine, B. McCredie, W. Starke, E. Welbon, US Patent 5,894,575 " Method and System for Initial State Determination for Instruction Trace Reconstruction, Assigned to IBM.

[18] F. Levine, B. McCredie, W. Starke, E. Welbon, US Patent 5,862,371, "Method and System for Instruction Trace Reconstruction Utilizing Performance monitor outputs and bus Monitoring" Assigned to IBM.

[19] A. Hartstein and T. Puzak. The optimum pipeline depth for a microprocessor, 29th International Symposium on Microarchitecture, pages 7-13 May 2002.

[20] A. Hartstein and T. Puzak. Optimum power/performance pipeline depth. 36th Annual IEEE/ACM International Symposium on Microarchitecture in MICRO, Dec. 2003.

[21] T. Puzak, P. Emma, A. Hartstein, V. Srinivasan, "When prefetching Improves/Degrades Performance" Conference on Computing Frontiers Proceedings of the 2nd conference on Computing frontiers 2005, Ischia, Italy May 04 - 06, 2005.

[22] P. Emma, A. Hartstein, T. Puzak, V. Srinivasan, "Exploring the Limits of Prefetching", IBM Journal of Research and Development Volume 49 , Issue 1 (January 2005).

[23] J. Cocke, B. Randell, H. Schorr, and E.H. Sussenguth, "Apparatus and Method in a Digital Computer for Allowing Improved Program Branching with Branch Anticipation, Reduction of the Number of Branches, and Reduction of Branch Delays," U.S. Patent #3577189, assigned to IBM Corporation, Filed Jan. 15, 1965, Issued May 4, 1971.

[24] E.H. Sussenguth, "Instruction Sequence Control," U.S. Patent #3559183, assigned to IBM Corporation, Filed Feb. 29, 1968, Issued Jan. 26, 1971.

[25] S. Hanatani, M. Akagi, K. Nigo, R. Sugaya, and T. Shibuya, "Instruction Prefetching Device with Prediction of a Branch Destination Address," U.S. Patent #4984154, assigned to NEC Corporation, Filed Dec. 19, 1988, Issued Jan. 8, 1991.

[26] J.S. Liptay, "Design of the IBM Enterprise System/9000 High-End Processor," IBM Journal of Research and Development, Vol. 36, No. 4, pp. 713-731, July, 1992.

[27] D.B. Fite, J.E. Murray, D.P. Manley, M.M. McKeon, E.H. Fite, R.M. Salett, and T. Fossum, "Branch Prediction," U.S. Patent #5142634, assigned to Digital Equipment Corporation, Filed Feb. 3, 1989, Issued Aug. 25, 1992.

[28] P.G. Emma, J.W. Knight, J.H. Pomerene, R.N. Rechtschaffen, and F.J. Sparacio, "Branch Target Table," IBM Technical Disclosure Bulletin, p. 5043, Apr. 1986.

[29] C.H. Perleberg, and A.J. Smith, "Branch Target Buffer Design and Optimization," IEEE Transactions on Computers, Vol. 42, Issue 4, pp. 396-412, Apr., 1993.

[30] B. Calder, and D. Grunwald, "Fast and Accurate Instruction Fetch and Branch Prediction," Proceedings of the 21st Annual International Symposium on Computer Architecture, pp. 2-11, April 18-21, 1994.