An Analysis of the Effects of Miss Clustering on the Cost of a Cache Miss

Thomas R. Puzak, A Hartstein, P. G. Emma, V. Srinivasan, Jim Mitchell
IBM – T. J. Watson Research Center
PO Box 218
Yorktown Heights, NY 10598
914-945-4360
trpuzak, amh, pemma, viji, jammitch @ us.ibm.com

ABSTRACT
In this paper we describe a new technique, called pipeline spectroscopy, and use it to measure the cost of each cache miss. The cost of a miss is displayed (graphed) as a histogram, which represents a precise readout showing a detailed visualization of the cost of each cache miss throughout all levels of the memory hierarchy. We call the graphs 'spectrograms' because they reveal certain signature features of the processor's memory hierarchy, the pipeline, and the miss pattern itself. Next we provide two examples that use spectroscopy to optimize the processor's hardware or application's software. The first example demonstrates how a miss spectrogram can aid software designers in analyzing the performance of an application. The second example uses a miss spectrogram to analyze bus queueing. Our experiments show that performance gains of up to 8% are possible. 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.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: *Design studies, Measurement techniques, Modeling techniques, Performance attributes*

General Terms
Algorithms, Measurement, Performance

Keywords
Cache, Pipeline, Algorithm, Spectrogram

1. INTRODUCTION
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]. Chou, et. al. [7, 8] studied several techniques (out-of-order, runahead, value prediction, prefetching, and 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. Qureshi et al. [9] demonstrate that not all misses have the same cost and measures miss parallelism to improve cache performance by altering the replacement algorithm.

CF'07, May 7–9, 2007, Ischia, Italy.

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 for the cost of a 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 (e.g. amount of overlap between misses in the miss cluster). 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. 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 [10-17]. 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 misprediction 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 misprediction.

Additional performance tools are described in: Dean et al. [18] a technique for pairwise sampling used to track concurrent events to measure performance, Fields et al. [19] use 'shotgun profiling' to construct dependence graphs and study the performance of an

application, while Karkhanis et al. [20] use analytical models to study the performance of concurrent events.

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 that specific instruction sequence.

Next we provide two examples that show how the information displayed in a cache miss spectrogram can be used to optimize the processor's hardware or application's software. The first experiment uses a miss spectrogram to identify software inefficiencies and describes solutions to improve performance. The second example uses a miss spectrogram to highlight the effects of bus queueing and describes a mechanism to improve performance.

The rest of this paper is organized as follows: Section 2 contains definitions and terminology. Section 3 describes constructing a miss spectrogram. The simulation model is described in Section 4. In Section 5 we measure the cost of a data miss. In Section 6 we use the miss spectrogram to analyze software and hardware performance problems and describe solutions to improve performance. Summary and conclusions are discussed in Section 7.

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}) 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 (cache, and TLB). It is the limiting case in which the processor has a first-level cache that is infinitely large 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 "memory adder" term, CPI_{FCA}, 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

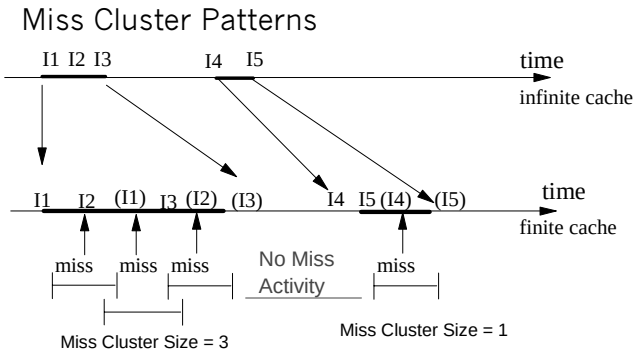
$$\left(\frac{Cycles}{Miss}\right) = (CPI_{OVERALL} - CPI_{INF})\left(\frac{Instructions}{Miss}\right) \quad (4)$$

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 that is 20 cycles away. 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 cache misses account for 300,000 cycles and the total $CPI = 1.3$ and CPI_{FCA}=.3. If the finite cache simulation run generates 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 Eq. 4 macroscopically to calculate the average cost of a miss over the total run time of an application. However, Eq. 4 can also be used microscopically to calculate the cost of a single miss. We will take a microscopic approach in using Eq 4 to calculate the cost of each miss and produce a miss spectrogram. As presented in Section 5, the information contained in a miss spectrogram represent the cost of all misses throughout all levels of the memory hierarchy, including the amount of overlap (parallelism) achieved between any two misses.

3. MAKING 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. 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 is size = 1 (a miss in isolation). 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.



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

We call these points (instructions) the infimum and supremum of the miss cluster. By convention, the infimum of a miss cluster is the last 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 of the second miss cluster is instruction 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 $[(I5EndOp - I4Decode)_{Finite\ Cache} - (I5EndOp - I4Decode)_{Infinite\ Cache}]$.

The reader will note that in the derivation above and in the equations presented in Section 2, no mention was made as to whether the processor is in-order or out-of-order. That is because out-of-order processing will not change the analysis. However, it may affect the manner in which the infinite cache running times for the sequence of instructions that surround a miss need to be determined. For example, if instruction processing is from an out-of-order processor, it may be necessary to save the sequence of instructions between the infimum and supremum of the miss (from the finite cache run) and use this same sequence of instructions (and their order) while determining the infinite cache run time.

By applying the above technique repeatedly, we can calculate the cost of a miss or miss cluster for an entire application. Both in-order and out-of-order processors can be evaluated this way. 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) 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 simulator has produced results similar to those shown in Sections 5 and 6 below. Each Simulator is cycle accurate and has been thoroughly validated against existing hardware. The processor model used in this paper is shown in Figure 2 and described in [21, 22], is a 4 issue superscalar, with address generation and cache access an independent out-of-order process. We use trace tapes produced for the IBM zSeries processor family. Instructions that typically produce addresses (LA, BXLE, SLL, SRL, ...) are pre-executed after the decode stage of the pipeline to avoid future pipeline stalls due to address interlocks. Loads are executed as soon as the datum fetched returns from the cache and the results are forwarded to all dependent instructions. 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, Endop and those instructions not pre-executed after the decode stage are completed and executed in-order. Future work is planned to measure the benefits of prefetching, SMT and SMT out-of-order execution.

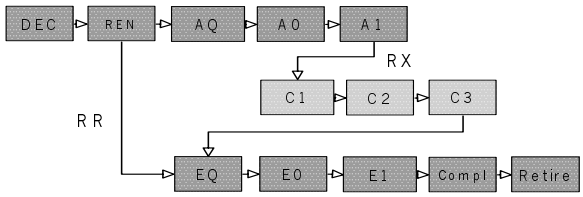


Fig. 2 shows the pipeline modeled in this study. The stages include: Decode, Rename, Agen Q, Agen, Cache Access, Execute Q, Execute, Completion and Retire.

We use instruction traces produced for the IBM zSeries 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 [23, 24] and oltp2, and oltp3); mcf, from SPEC 2000, SPECjbb 2000, and perf1 [23, 24] a large-processor simulator written in C++. Typically, trace lengths are 5 to 100 million instructions. The simulation environment can handle all of the SPEC suite; the application subset used for this work was chosen for its ability to stress L2 and L3 cache usage.

5. DATA MISS SPECTROGRAM

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. 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.

Using the techniques described above, Figure 3 shows the 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 cluster in cycles. The Y axis shows the percent of misses with that cost.

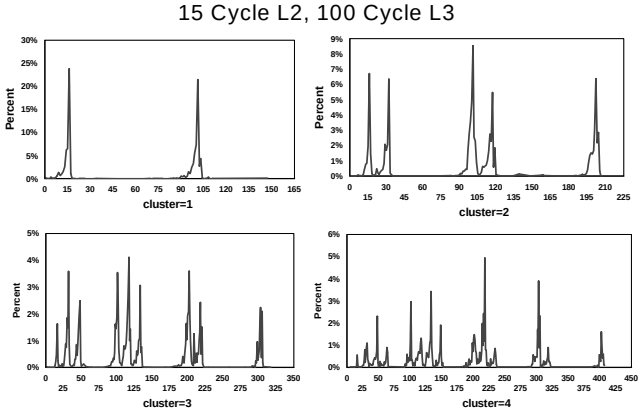


Fig 3, Miss Spectrogram for OLTP, L1=64K, L2=256K, Infinite L3

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 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. Each peak represents the amount of overlap between two misses. Integrating the area under each peak, we see that the costs are centered around 15, 30, 100, 115 and 200 cycles and have probabilities 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.

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, or 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 calculate the total finite-cache-adder to within 5% (one cluster at a time), and in many cases the error is less than 2%. This shows how accurately we can identify miss clusters and evaluate their costs.

Prefetching and bus delays can change the shape of the peaks in a spectrogram. Prefetching can broaden the left shoulder of any peak and show the degree that a prefetch is being issued in advance of the nominal miss penalty. Queueing and bus delays can increase the right shoulder of a peak, adding miss latency.

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 miss clusters of size 6 or less and over 40% of the misses are a miss in isolation. This was observed for most applications used in this study.

Figure 4. Misses by Cluster Size

Figure 5 plots the average cost of a miss versus cluster size. Notice how the average miss penalty decreases as the cluster size grows. The slope of the line indicates the degree that miss parallelism or miss overlap is occurring. Obviously, the greater the amount of miss overlap the greater the slope of the line. In this example the cost of a miss at a cluster size = 10 is approximately two thirds the cost of an isolated miss. This is far less than the potential for complete overlap.

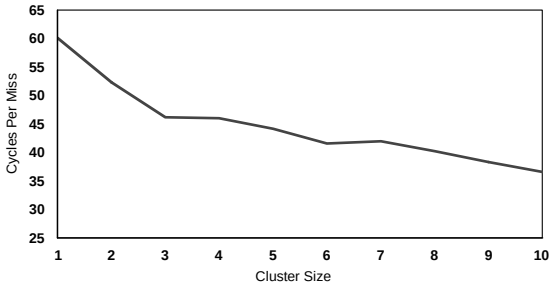


Figure 5 Average Cycles Per Miss Versus Cluster Size

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 6 shows the data miss spectrogram for cluster sizes = 1, 2, and 3.

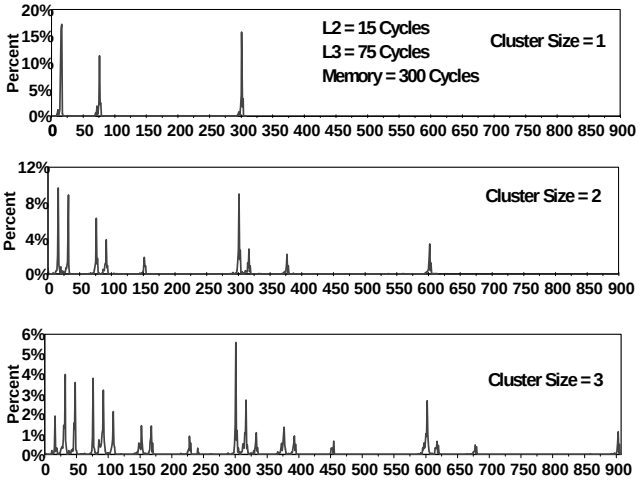


Figure 6. Data Miss Spectrogram for OLTP2, L1=64K, L2=256KB 15 Cycle Latency, L3=1Meg 75 Cycle Latency, Memory = 300 Cycle Latency

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 6 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. We show in Appendix A that for a memory hierarchy with N cache levels (L1, L2, L3,..., L_N, memory) and a miss cluster of size C, there are

$$\binom{C+N}{C} \qquad (5)$$

possible penalties (peaks) that characterize the canonical form hit/miss and overlap patterns. 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 have the possibility of causing zero delay. Using (5), and the memory hierarchy described in Figure 6, we see that plotting miss clusters of size 4, 5, and 6 could have 35, 56, and 84 peaks, respectively.

Each of the spectrograms above aids the 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 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, and 300 cycles for a miss to memory.

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. SPECTROGRAM ANALYSIS

In this section we provide two examples that use the information displayed in a cache miss spectrogram to optimize the processor's hardware or application's software. The first experiment demonstrates how a miss spectrogram can aid software designers in analyzing the performance of an application. The second example uses a miss spectrogram to analyze hardware performance.

Each experiment highlights the effects that bus queueing has on performance. Bus queueing can occur whenever the linesize of the cache is greater than the bus width and multiple cycles are needed to transfer a line between levels of caches during a miss. We begin by defining the terms that will be used throughout this

section. Bus queueing time is lengthened whenever the bandwidth between cache levels is decreased or the bus frequency ratio is increased.

Term	Definition
Bus Width	The number of bytes moved in or out of the cache per cycle during a miss.
Packet	Subsection of cache line. Equals size of bus width (in bytes). Multiple packets makeup a cache line
Packets Per Line	The ratio of cache linesize to bus width (packet size). This is also the number of bus cycles needed to move a line between levels of the memory hierarchy
Miss Latency	Number of cycles before first data (packet) returned after a cache mss is detected
Leading Edge penalty	Once a cache miss is detected, amount of time the processor stalls waiting for the first data (packet of line) to returned.
Trailing Edge Penalty	The observed cost of caching the remainder of the cache line after first packet is returned. The trailing-edge penalty would be zero if the entire line could be returned and installed in the cache in a single cycle. However, few designs can afford the implied luxuries of a line-wide data-return bus and cache-write capability.
Bus Frequency Ratio	The ratio of the processor frequency to the bus frequency. It is determined by logic speed, packaging and power limits.
Line Transfer Interval (LTI)	The number of processor cycles needed to move a cache line (Packets) on the bus and equals bus frequency ratio times the packets per line.

The baseline processor organization and memory hierarchy are described in Table 1. The L2 is set at 512 KB with a 15 cycle latency with an infinite size L3 set at 75 cycles (i.e. all L2 misses are resolved at the L3, 75 cycles away). A 256 byte line size is modeled throughout all three levels of the memory hierarchy. We consider an on-chip L2 and configure the bus connecting the L1 and L2 to be able to move 32 bytes each processor cycle. Thus the LTI for an L2 hit is 8 cycles.

L1 Cache Configuration	Split I and D each 64KB, 4 Way Set Associative 256 Byte Line 2 Cycle access
L2 Cache Configuration	Unified 1/2 Meg, 8-Way Set Associatvie, 256 byte Line, 15 cycle access
L3 Cache Configuration	4 Meg, 8-Way Set accociative, 256 byte line, 75 cycle access
Processor Pipeline	4 issue superscalar, out-of-order address generation and cache access, in-order execution, 32 entry instruction window

Table 1 Baseline Processor Configuration

We consider an off-chip L3 with a 32 byte bus and consider three different line transfer rates for an L3 hit: 32 bytes are transferred (from the L3 to the L2 and L1) every other cycle, every third cycle, or every fourth cycle. Thus it takes 16, 24, or 32 cycles to transfer a line from the L3 to the L2 and L1 (on a L3 hit).

Figure 7 shows a miss spectrogram for OLTP3 for cluster size= 1 using the three different LTIs for L3 hits. Immediately visible, in all three graphs, is the similarity in the peaks centered at 15 cycles (an L2 hit). Recall the L2 is on chip and all three processor configurations deliver 32 bytes every cycle for an L2 hit. Each graph has a small left shoulder (at 15 cycles) and shows some misses costing as low as 8 cycle (leading edge penalty). However, the right shoulder is much larger (in area) than the left shoulder with many misses costing more than the 15 cycle miss latency. Also notice, the peak at 75 cycles is different in the three graphs. The right shoulder grows as the line transfer interval grows from 16 cycles, 24 cycles, to 32 cycles. There are two very distinguishable sub peaks in the right shoulder of the three graphs. The spectrogram for a 16 cycle LTI has sub peaks at 84 and 90 cycles. Similarly the spectrograms for the 24 and 32 cycle LTIs have sub peaks at 92 and 98 cycles, and 100 and 108 cycles, respectively. Notice, as the line transfer intervals grows the sub peaks correspondingly shift eight cycles to the right.

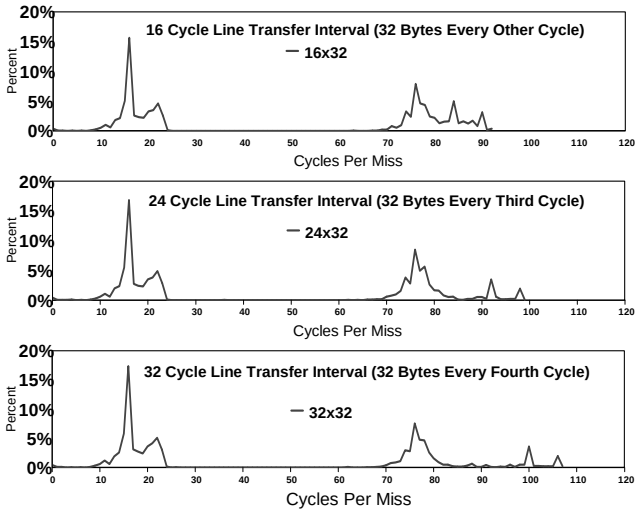


Figure 7, Cluster Size = 1 Miss Spectrogram for OLTP3 fofr 16, 24, and 32 Cycle Trailing Edge

In order to investigate the cause of these sub-peaks, we use two reports that are produced during a spectroscopy run: the Miss-Cost report and Cost-Analysis report[1]. Figure 8 gives an example of a small portion of the Miss-Cost report for the 24 cycle LTI spectrogram. It lists every miss by cluster size, cost, infimum and supremum instruction on the trace, miss address and instruction address that make up the miss spectrogram.

[1]In the analysis that follows we investigate the cause of the sub peak at 98 cycles shown in the 24 cycle LTI spectrogram. The sub peak at 92 cycles is caused by a similar addressing pattern and discussions regarding its removal are omitted for brevity (but accomplished in the same manner).

Cluster Num	Size	Cost	Infimum Inst	Supremum Inst	Miss Address	Inst Address	Inst Number
27973	1	98	348389	348417	20B1FA20	000668EC	348389
27984	3	75	348421	348449	20C1FD20	00066EEC	348421
27984	3	75	348421	348449	20C1FE40	00066EEC	348433
27984	3	75	348421	348449	20C1FF00	00066EEC	348441

Figure 8. Miss Cost Report For OLTP3

Figure 9 shows the Cost-Analysis report. It summarizes all misses according to the instruction that generated the miss and sorts them by frequency (number of times the instruction generated a miss) and by cost (total cost of misses). The top five

Highest count items:

ASID	Inst Addr	Count	% of Total
012E	93DAA	4410	0.49%
0134	9CDAA	3854	0.43%
0146	3B46E	3023	0.34%
0146	9724EA6	2596	0.29%
012E	3CF3DD62	2444	0.27%

Highest cost items:

ASID	Inst Addr	Total Cost	% of Total
0146	9724EA6	144045	0.33%
0146	3BE10A3E	142423	0.32%
0146	3BE10A02	139804	0.32%
0146	96F2A68	136345	0.31%
012E	93DAA	125253	0.28%

Figure 9 Cost Analysis Report for OLTP3

instructions generating the most misses (by frequency) along with their address-space-identifier (ASID) are shown on the left, while the top five instructions generating the most miss cost are shown on the right. Notice that the instruction positions change when sorted by frequency and then by cost. That is, only two of the instructions that produced the most misses (when sorted by frequency) are in the top five instructions that produced the most cost (when sorted by miss cost). Obviously, software tuning would focus on miss-penalty cost rather than miss frequency because most of the cost of a miss can be overlapped with another miss if it occurs in parallel with other misses. This was clearly highlighted in Figures 3 and 6 when examining cluster size 3 misses and observing miss penalty peaks at 15, 30, and 45 cycles.

Returning back to Figure 8, the file was sorted on cost and cluster size and misses that make up the sub peak at 98 cycles were identified. Once the infimum and supremum instructions on the trace are known, the simulation run is repeated and a detailed cycle-by-cycle instruction trace (report) is produced highlighting the events that occurred during the miss.

Examination of the cycle-by-cycle traces shows that the sub peaks were caused by two byte-compare instructions, two instructions apart. Figure 10 illustrates the cause for this cost. The miss address (generated by the first byte compare) was to the 6th 32 byte packet of a 256 byte line. This is the first packet that is returned on the bus when processing the miss. During the miss, packets are returned in increasing order (starting from the miss address) until the end of the line is reached, then processing continues to the beginning of the line until the entire line is transferred. Thus, the 8 packets that make up the line are returned in the following order: 6, 7, 8, 1, 2, 3, 4 and 5.

The next reference to the line (the second compare instruction, two instructions later) is 10 bytes prior to the miss address and depending on byte alignment is to the preceding 32 byte segment (the 5th packet) in the line. The targets of both compares are referenced using a common base register (data structure) and these two compare instructions always referenced two different bytes, 10 bytes apart. Since the packets in the line are returned in ascending order until the end-of-line is reached then wrapping around to the beginning of the line, the packet containing the second byte will not be available on the bus until the entire line is transferred to the cache (when they are not in the same packet). This accounts for the 8 cycle sub peak shift as the TLI is increased from 16, to 24, and 32 cycles. This instruction sequence occurred repeatedly on the trace (hundreds of times) and depending on byte alignment within a cache line would generate a miss cost of approximately 75 cycles (if both bytes were contained in the same 32 byte packet) or 98 cycles if the two bytes were in neighboring packets.

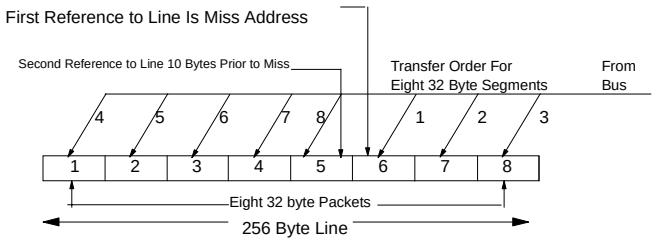


Figure 10. Miss Pattern For 96 Cycle Cost Per Miss For OLTP3

Once the cause of the miss cost was identified a second simulation run was conducted to test whether the sub peaks can be removed. The instruction trace was modified and the two bytes compared by each instructions were switched. This can be accomplished (in reality) by switching the locations of the two bytes in the data structure. Now the instruction that compares the first byte in the line (the first reference to the line) is 10 bytes prior to the second byte compared (referenced) in the line. With these bytes switched, the first reference will cause a cache miss (as before) however the second reference to the line will be to a byte that is 10 bytes after the miss location and be available on the same cycle, if contained in the first packet returned during a miss, or one packet (transfer) later.

The simulator was rerun using the modified trace and data structure (MtDStr) and the spectrogram for the original run and modified data structure are shown in Figure 11. Most of the two spectrograms overlap along the X axis between miss values of 0 to 70 cycles. This should be expected because the L2 bussing strategy was not changed between the two simulation runs.

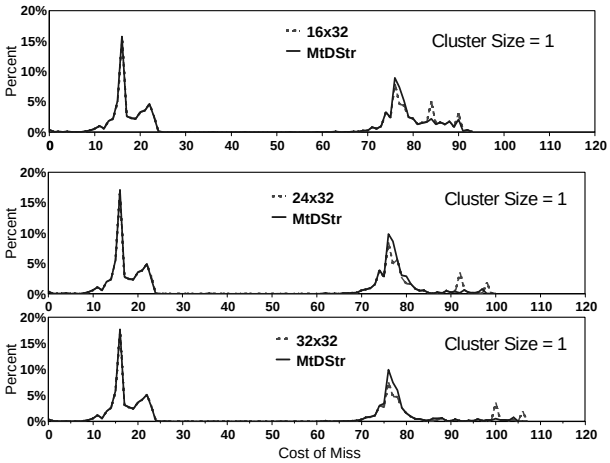


Figure 11 Spectrogram for OLTP3 with Location of Compare Bytes Switched

However, notice the change in the shape of the spectrogram between 75 cycles to 110 cycles is all three graphs. The sub peaks from the original run are gone and more misses now have a cost near 75 cycles, the miss latency for a L3 hit. The results show that swapping the locations of the bytes compared reduces the cost of the targeted misses and the sub peaks are removed. Recall, this instruction sequence was identified through the use of the miss spectrogram.

We should offer a few comments regarding this experiment. First, we only changed the addresses generated (compared) by two instructions and the results are clearly visible in the corresponding spectrogram. The total number of misses for each simulation run was virtually the same, however, the sub peaks from the original runs are gone and the peak at 75 cycles (for the new simulation run) grew indicating that the cost for these misses is shifted to the nominal miss penalty (latency) for an L3 hit. Second, individual misses can be identified and investigated using the miss-cost report or the cost analysis report. For example, a software designer can list the 'top 100 instructions' that caused the most misses (by cost). Once an instruction or miss is identified, a cycle-by-cycle trace will reveal the cause of the miss and miss cost. Substantial performance gains are possible even if only 10 to 20 percent of the misses (from the top 100 instructions) can be optimized. For example, the performance gain in just this one example was approximately 0.2 %. Third, pipeline spectroscopy has both the accuracy and precision we need to measure the cost of even a single cache miss. Accuracy is demonstrated by having the ability to reconstruct the true total finite cache adder by adding the cost of each individual miss cluster. Precision is demonstrated by the repeatability of the readings. For example, in the experiment above the LTI for an L3 hit was modeled at 16, 24, and 32 cycles and a shift of eight cycles was observed for each LTI change in the sub peaks shown in Figure 11.

In our next experiment we attempt to reduce the trailing edge penalty of a cache miss even further by modifying the order that the packets within a line are returned on the bus. That is, instead of returning packets in a fixed order (from the miss address, to the end-of-line address, then wrapping around to the beginning of the line) hardware is modeled that can transfer the line (packets) in the order they are referenced by the processor. Other mechanisms aimed at reducing the trailing edge penalty can be found in [25-28]. These describe techniques that multiplex data on the bus from concurrent misses, or can remember the referencing pattern generated by the processor during a prior miss and can rearrange the returning packets to match the previous referencing pattern when the miss or instruction causing the miss repeats.
In our scheme, we also have the ability to transfer the packets in a line in an out-of-order sequence. This applies to both L2 and L3 hits. Obviously, this involves adding (modeling) hardware that recognizes the order that the bytes are referenced within a line and supplying the packet (within the line) on the next available bus cycle along with the packet address. If there is not a new reference to the line during the time the line is being transferred between caches, the line is transferred in the original order as described above (the next in order packet).

Again we model the same L2, L3 and bus structure and model the three LTIs as before (16, 24, and 32 cycles). Figure 12 shows the before and after miss spectrogram for the OLTP3 workload for a miss cluster of size = 1. The modified data return is labeled MDR. The extent that this improves performance should be immediately visible by examining the shape of the miss spectrogram peaks centered at 15 (an L2 hit) and 75 (an L3 hit). Notice the change in the shape of the right shoulder for each peak, indicated a change in the trailing edge penalty. In each graph, the right shoulders for the peaks at 15 and 75 are reduced and more misses are shifted to have a nominal 15 or 75 cycle miss penalty.

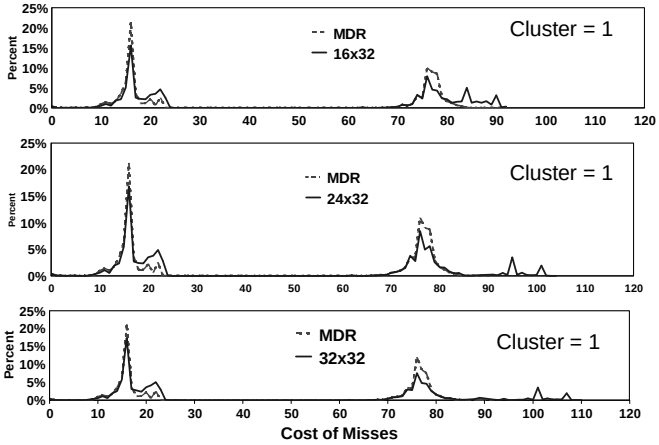


Figure 12 Modified Data Return for OLTP3

Each workload described in Section 4 was rerun though the simulator using the modified data return mechanism described above. Figure 13 shows the overall performance impact for the six workloads studied. The performance improvements of changing the order that the line is returned increases as the number of cycles to transfer a line increases from 16, to 24, and 32 cycles. Depending on the LTI, performance improvements range from .5% to over 8%.

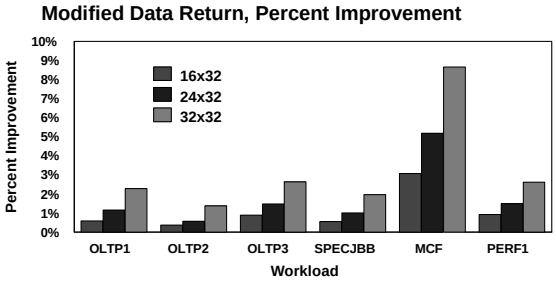


Figure 13 Percent Improvement

7. SUMMARY

A new technique has been presented for calculating the cost of a miss and displaying images 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 used this principle to produce a miss spectrogram, which represents a precise readout of the cost of every miss throughout all levels of the memory hierarchy. 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 memory level parallelism an application can achieve. Two examples were presented using the information contained in a spectrogram. The first identified software abnormalities and helped produce a more optimal data layout. The second identified bus queueing and miss processing stalls. Armed with this information hardware and software designers can devise solutions and evaluate performance improvements. Future work is needed to study in-order versus out-out-order effects on MLP, 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).

8. 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] Moinuddin Qureshi, Daniel Lynch, Onur Mutlu, Yale Patt, "A Case for MLP-Aware Cache Replacement" in 33rd International Symposium on Computer Architecture, June 2006

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

[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"

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

[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"

[14] 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.

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

[16] 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"

[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,

[18] J. Dean, J. E. Hicks, C. A. Waldspurger, W. E. Weihl, and G. Z. Chrysos. ProfileMe : Hardware support for instruction-level profiling on out-of-order processors. In MICRO'97: pages 292--302, 1997.

[19] Brian A. Fields, Rastislav Bodik, Mark D. Hill, Chris J. Newburn., Interaction cost and shotgun profiling. ACM Transactions on Architecture and Code Optimization, Vol 1, No. 3. Sept 2004.

[20] Tejas Karkhanis,James E. Smith, A First-Order Superscalar Processor Model. Proceedings of the 31st ISCA. pages 338–349, June 2004.

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

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

[23] 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.

[24] 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).

[25]US Patent 5,636,364 Method for enabling concurrent misses in a cache memory

[26] US Patent 5,233,702 Cache miss facility with stored sequences for data fetching

[27]IBM Technical Disclosure Bulletin, "A Protocol for Processing Concurrent Misses", Dec. 1993, vol. 36 No. 12.

[28]IBM Technical Disclosure Bulletin, vol. “Design for Improved Cache Performance via Overlapping of Cache Miss Sequences” vol. 25 No. 1B Apr. 1983 pp. 5962-5966.

[29] R. Bartoszynski, M Niewiadomska-Bugaj,Probability and Statistical Inference, (Wiley series in probability and statistics) 1996

9. APPENDIX A

We prove there are $\binom{C+N}{C}$ possible miss penalties in a cluster of C misses with a memory hierarchy of N levels (L1, L2, ..., L_N, and memory) with N distinct (non-multiple) miss latencies. Let each level of the memory hierarchy be represented by a distinct (non-multiple) number, there are N of them. We represent this problem as sampling with replacement to determine the number of unique combinations (sums) using these numbers. It is sampling with replacement because there is an inexhaustible supply of miss latencies regardless of the cluster size. We use the notation $\left[\begin{matrix} N \\ C \end{matrix} \right]$ to denote N items choose C (the cluster size) for sampling with replacement. The problem can then be expressed as determining the number of unique sums from N items as we vary the number of picks from 0 to C.

$$\sum \left[\begin{matrix} N \\ i \end{matrix} \right] 0 \le i \le C = \left[\begin{matrix} N \\ 0 \end{matrix} \right] + \left[\begin{matrix} N \\ 1 \end{matrix} \right] + \left[\begin{matrix} N \\ 2 \end{matrix} \right] + \left[\begin{matrix} N \\ 3 \end{matrix} \right] + , \ldots , + \left[\begin{matrix} N \\ C \end{matrix} \right] \quad \text{(1a).}$$

Note that i and C can be greater that N in (1a) because sampling is done with replacement.

From [29] $\left[\begin{matrix} N \\ k \end{matrix} \right] = \binom{N + K - 1}{K}$ so we can rewrite (1a) as

$$\binom{N-1}{0} + \binom{N}{1} + \binom{N+1}{2} + \binom{N+2}{3} \ldots \binom{N+C-1}{C} \quad \text{(2a)}$$

Also, from [29] $\binom{N}{k} = \binom{N-1}{k-1} + \binom{N-1}{k}$ (3a)

So we can combine the first two terms of (2a) and obtain

$$\binom{N+1}{1} + \binom{N+1}{2} + \binom{N+2}{3} \ldots \binom{N+C-1}{C} \quad \text{(4a)}$$

Applying (3a) repeatedly the series collapses and the desired sum is produced $\binom{C+N}{C}$.