The Spatial Characteristics of Load Instructions

Joshua J. Yi, Resit Sendag, and David J. Lilja
Department of Electrical and Computer Engineering
Minnesota Supercomputing Institute
University of Minnesota - Twin Cities

Abstract

Relatively little background work has been done to examine the miss behavior of all static and dynamic load instructions, especially in the context of the entire program. This study addresses this gap in knowledge by presenting the whole-program (as opposed to sampling) profiling results for load behavior. Specifically, this study confirms the conclusion of previous work in that a very small percentage of static loads (i.e. PCs) account for a disproportionately large percentage of the total L1 cache misses. This study also shows that an equally small percentage of unique effective addresses (EAs) account for a comparable percentage of the total L1 misses. In other words, a few "hot" PCs and EAs are responsible for a large percentage of the total L1 misses. Furthermore, a surprisingly large percentage of the dynamic loads can bypass the memory hierarchy by using store forwarding. However, the main contribution of this work is to quantify the whole-program behavior of static and dynamic loads with several different metrics that either confirm or contradict conventional thought or previous work.

1 Introduction

One of the most difficult - yet most important - problems in computer architecture is the result of the increasing disparity between the processor and memory speeds. As the two speeds diverge, the number of cycles needed to service L1, L2, or even L3 misses will increase. As a result, the fraction of the program's execution time devoted to memory accesses will increase.

The significance of this problem [Balasubramonian01, Burger97-2, and Collins01-1] has inspired a plethora of solutions [Annavaram00, Annavaram01, Balasubramonian01, Chen97,

Collins01-1, Collins01-2, Collins01-3, Kim97, Luk01, Pierce96, Roth98, Yang00, Zilles00, and Zilles01]. While these papers propose microarchitectural mechanisms that have different degrees of complexity and performance improvement, they do not thoroughly examine the fundamental characteristics of load instructions over the entire program. The fundamental characteristics of loads may include load access patterns, the properties of static loads, the temporal nature of dynamic loads, and so forth. While it is not essential to quantify or fully understand these characteristics before proposing a new microarchitectural mechanism - since the nature of this problem is relatively well-understood, a thorough examination of these characteristics may not only improve existing solutions, but also may reveal additional characteristics that can be exploited in novel ways. Therefore, by verifying the correctness of conventional thought and confirming the accuracy of examining only a sub-set of the loads (static or dynamic) or a sub-set of the entire program (i.e. sampling), this study provides a firm foundation for future solutions to the memory problem. To narrow the focus of this study to a tractable problem, we initially study only the spatial characteristics of static and dynamic loads while deferring a detailed examination of their temporal characteristics to future work.

The contributions of this study are as follows:

1. It examines the spatial characteristics of all static and dynamic loads in the context of the entire program.
2. It examines the invariance of these spatial characteristics for different program inputs.
3. It examines the effect of different cache configurations on these spatial characteristics.

The remainder of this paper is organized as follows: Section 2 describes the programs

that were profiled, the methodology that was employed, and the different processor configurations that were used, while Section 3 presents and analyzes the results. Section 4 describes some related work. Section 5 concludes and describes some items for future work.

2 Profiled Programs, Methodology, and Processor Configurations

2.1. Profiled Programs

Table 1 shows the benchmarks that were profiled in this study and the specific input sets that were used for each.

Table 1: Benchmarks Profiled in this Study and Their Input Sets

Benchmark	Suite	Type	Input Set A	Input Set B
099.go	SPEC 95	Integer	Train	Reduced Test
124.m88ksim	SPEC 95	Integer	Train	Test
126.gcc	SPEC 95	Integer	Test	Train
129.compress	SPEC 95	Integer	Train	Test
130.li	SPEC 95	Integer	Train	Test
132.ijpeg	SPEC 95	Integer	Test	Train
134.perl	SPEC 95	Integer	Test	Train
164.gzip	SPEC 2000	Integer	Reduced Small	Reduced Medium
175.vpr	SPEC 2000	Integer	Reduced Medium	Reduced Small
177.mesa	SPEC 2000	Floating-Point	Reduced Large	Test
179.art	SPEC 2000	Floating-Point	Reduced Large	Test
181.mcf	SPEC 2000	Integer	Reduced Medium	Reduced Small
183.equake	SPEC 2000	Floating-Point	Reduced Large	Test
188.ammp	SPEC 2000	Floating-Point	Reduced Medium	Reduced Small
197.parser	SPEC 2000	Integer	Reduced Medium	Reduced Small
255.vortex	SPEC 2000	Integer	Reduced Medium	Reduced Large
300.twolf	SPEC 2000	Integer	Test	Reduced Large

All benchmarks were compiled with SimpleScalar's [Burger97-1] version of gcc (version 2.6.3) with full optimizations (-O3). To control the execution time for several of the benchmarks, reduced input sets were used. Benchmarks that use a reduced input set exhibit behavior similar to when the benchmark is executed using the reference input set [KleinOsowski00]. For 134.perl and 175.vpr, each input set is composed of two separate inputs

(Jumble and Primes for 134.perl, Input Set A and Place and Route for 175.vpr). To avoid losing any information about each benchmark's behavior with a different input, the result for each input is reported separately.

2.2 Methodology

In this study, loads were classified according to their program counter (PC) value, their effective address (EA), and their reorder buffer (ROB) entry number (i.e. position in the ROB). By classifying the loads according to their PCs, the characteristics for static load instructions could be easily extracted. By further classifying the loads according to their EAs, the characteristics for each memory address could be readily determined. Finally, by further classifying each load according to their ROB position, we were able to estimate the impact that each "hot" load (i.e. a load with a large number of misses) had on the program's execution time. Due to the nature of this profiling, this method of classification did not affect how the results were processed or analyzed.

For each load, the total number of accesses that that particular load made was also stored. For this study, the total number of accesses (and the total number of hits) includes both the correct-path loads and the wrong-path loads. (A wrong-path load is a load that is on a mispredicted control, i.e. branch, path.) In addition to the preceding information, Table 2 shows the remaining information that was gathered for each load.

The load-store queue (LSQ) is a queue between the processor core and the memory hierarchy that maintains the total ordering of all loads and stores that have not finished executing (i.e. received or written their value). If a load has the same address as a preceding store, the store can non-speculatively "forward" its value to the load. We refer to this case as a hit in the LSQ.

Table 2: Information Gathered for Each Load

Level in Memory Hierarchy	Information
Load-Store Queue (LSQ)	Total Number of LSQ "Hits"
	Total Number of Wrong-Path "Hits"
	Total Hit Latency (Cycles)
L1 D-Cache	Total Number of L1 D-Cache Hits
	Total Number of Wrong-Path "Hits"
	Total Hit Latency (Cycles)
	Total Partial Miss Latency (Cycles)
L2 Unified Cache	Total Number of L2 Cache Hits
	Total Number of Wrong-Path "Hits"
	Total Hit Latency (Cycles)
	Total Partial Miss Latency (Cycles)
Main Memory	Total Number of Memory "Hits"
	Total Number of Wrong-Path "Hits"
	Total Hit Latency (Cycles)

A partial miss is a miss to a cache block that is already in transit to the L1 D-Cache. For instance, if two loads reference the same cache block and if the first load is a miss, then the second load is classified as a partial miss since the cache block is already in transit to the L1 D-Cache (due to the miss from the first load). Therefore, the second load does not incur the full miss latency.

2.3 Processor Configurations

To store all of this information for all possible combinations of PC, EA, and ROB position, we modified sim-outorder from the SimpleScalar tool suite to accommodate a b-tree. Sim-outorder is a state-of-the-art event-driven simulator that models a superscalar processor. Table 3 shows the processor configuration that was used in this study while Table 4 shows the cache configurations.

All 16 possible combinations of L1 D-Cache and L2 Cache sizes and associativities were simulated. For all 16 cache configurations, only the L1 D-Cache and L2 sizes and associativities

were varied; all the other cache and processor parameters were held constant.

Table 3: Processor Configuration

Parameters	Value(s)
# of Integer ALUs	6
# of FP ALUs	4
# of Integer Multipliers/Dividers	2
# of Floating-Point Multipliers/Dividers	2
# of Instruction Fetch Queue Entries	64
Decode, Issue, Commit Width	8-Way
# of Reorder Buffer Entries	128
# of Load-Store Queue Entries	64
# of Memory Ports	4
L1 I-Cache Size, Associativity, Block Size, Replacement Policy, Latency	32KB, 2-Way, 32 Bytes, Least Recently Used, 1 cycle
Memory Latency (First, Following Blocks)	120 Cycles, 5 Cycles
Branch Predictor	2-Level
Branch Misprediction Penalty	3 Cycles

Table 4: Cache Configurations

Parameters	Value(s)
L1 D-Cache Sizes	8KB, 32KB
L1 D-Cache Associativities	1-Way, 4-Way
L1 D-Cache Block Size, Replacement Policy, Latency	32 Bytes, Least Recently Used, 1 cycle
L2 Unified Cache Sizes	128KB, 512KB
L2 Unified Cache Associativities	2-Way, 8-Way
L2 Unified Cache Block Size, Replacement Policy, Latency	64 Bytes, Least Recently Used, 10 cycles

The cache sizes were deliberately chosen to be smaller than what would be appropriate for an 8-way issue processor to account for the smaller memory footprint – as compared to the reference input set – of the input sets that we used for each benchmark. The cache sizes and associativities were scaled based on the preliminary results of an on-going cache miss study and the results from [Cantin01].

3 Results and Analysis

This section presents the information gathered in this profiling study in terms of five

major results: 1) The distribution of hits in the memory hierarchy, 2) The distribution of misses among EAs and PCs, 3) The characteristics of the top 16 loads with respect to the number of L1 misses, 4) The average number of EAs that are referenced by each PC and the average number of PCs that reference each EA, and 5) The effect of the program's inputs on the aforementioned four results. Note that results 2 and 3 explicitly focus on the characteristics of the most problematic static and dynamic loads, that is the "hot" loads.

3.1 Distribution of Hits in the Memory Hierarchy

One of the goals of this study was to determine what the hit distribution is across the memory hierarchy (LSQ, L1 D-Cache, L2, and main memory) for all dynamic loads. Figure 1 shows the distribution of hits in the memory hierarchy. Figure 1A shows the percentage of hits in each level of the memory hierarchy while Figure 1B shows the approximate percentage of cycles spent waiting for loads in each level of the memory hierarchy. Both figures show the results when using a 32KB, 4-way L1 D-Cache and a 512KB, 8-way L2 Cache.

These two figures both confirm and contradict elements of conventional thought. First of all, the distribution of hits across the different levels of the memory hierarchy is close to what one would intuitively expect. Conventional thought dictates that caches closer to the processor core will account for a very large percentage of the hits (i.e. supply most of the values for the loads). With the exception of 179.art, at least 88% of all dynamic loads hit in either the LSQ or the L1 D-Cache. The relatively low percentage of LSQ and L1 D-Cache hits in 179.art is due to the relatively large memory footprint of that program compared to the sizes of the LSQ and the L1 D-Cache.

Secondly, these figures also confirm conventional thought that a disproportionate

percentage of the total load latency is the result of hits to the L2 Cache and main memory. This is obviously due to the higher latencies of the L2 Cache and main memory and in spite of the large percentage of hits to the LSQ and L1 D-Cache.

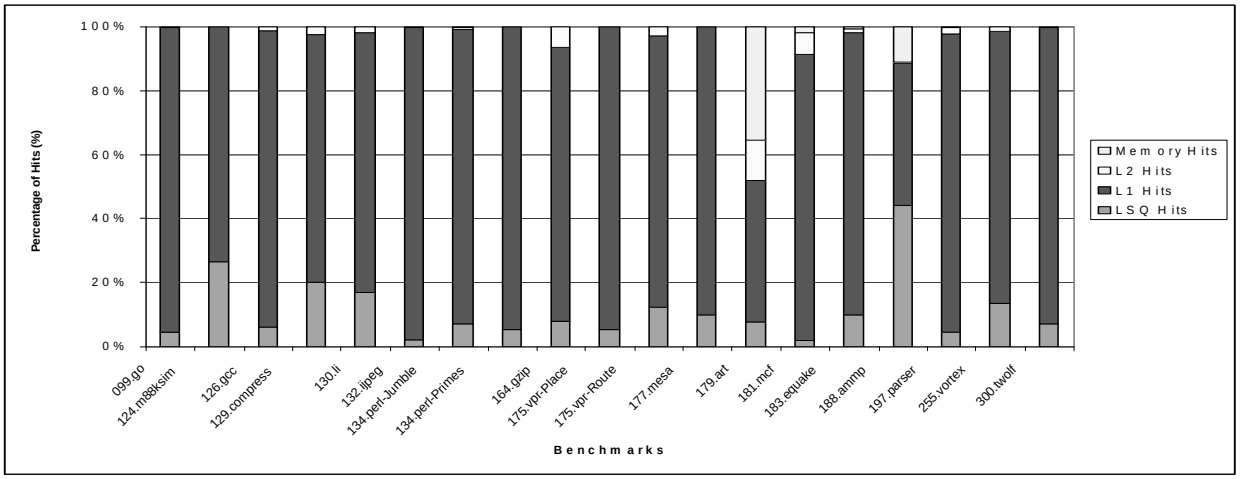

Figure 1A: Distribution of Hits in the Memory Hierarchy

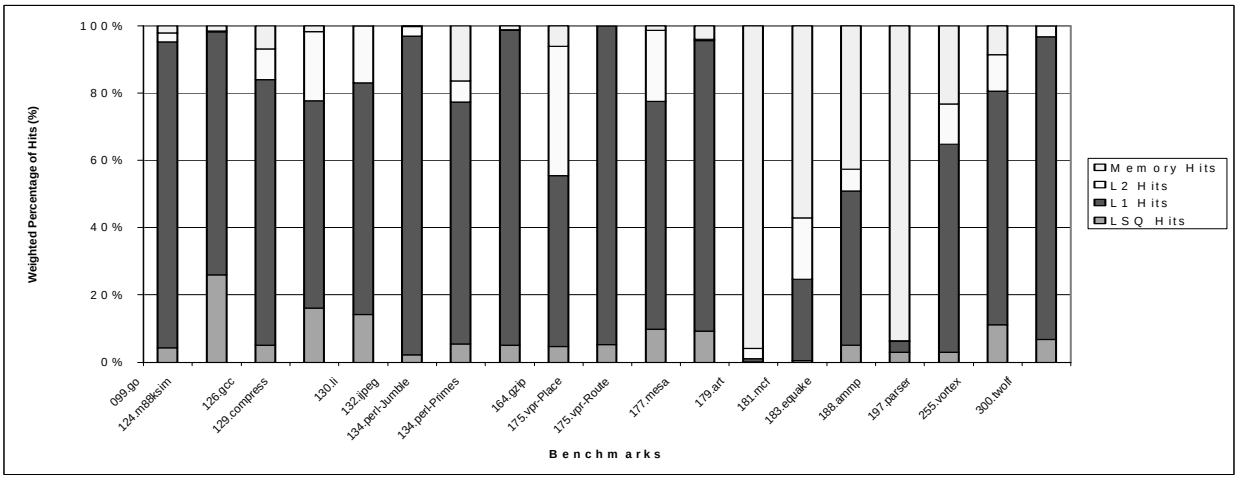

Figure 1B: Distribution of Hits in the Memory Hierarchy, Weighted by Hit Latency

Figure 1: Distribution of Hits in the Memory Hierarchy, Not Weighted and Weighted by the Hit Latency; 32KB, 4-Way L1 D-Cache and 512KB, 8-Way L2 Unified Cache

Finally, the most surprising result contradicts conventional thought. In particular, we found that the percentage of loads that "hit" in the LSQ due to store forwarding is very high. In

6 of the 19 benchmarks, at least 10% of all the dynamic loads hit in the LSQ. While the 64 entry LSQ could act as a fully associative L0 Cache, the primary reason for this result is that for these particular benchmarks, there is an extraordinary temporal correlation between a significant percentage of the loads and the last store to write to the load's effective address. In other words, for these benchmarks, at least 10% of all the loads were within 128 instructions (number of ROB entries) of the last store to that effective address. It is important to note that in the SimpleScalar architecture, loads are dispatched to the memory sub-system only after the effective addresses for all preceding stores are known. This result is most prominently displayed by 124.m88ksim (26.51%), 129.compress (20.13%), and 188.ammp (44.19%). This result demonstrates the efficacy of store forwarding, especially in the case of 188.ammp.

Increasing the L1 D-Cache size and associativity has a large effect on the distribution of hits in the memory hierarchy for most of the benchmarks. The only exceptions are for 179.art and 188.ammp. For 179.art, increasing the L1 D-Cache size and associativity increases the hit percentage of the L1 D-Cache by only 1% because of the large memory footprint of the program. However, for 188.ammp, increasing the L1 D-Cache size and associativity increases the hit percentage of the L1 D-Cache by only 1% because a very high percentage of loads are already LSQ hits. Therefore, for these two benchmarks, a larger, more highly associative cache has virtually no effect on the memory performance, albeit for two completely different reasons.

For the other benchmarks, the percentage of L1 D-Cache hits increases as its size and associativity are increased. The increase in the percentage of L1 D-Cache hits ranges from 2.73% (124.m88ksim) to 16.82% (255.vortex), with a median of 5.24%. Therefore, as expected, a larger, more highly associative L1 D-Cache decreases the number of L2 Cache and main memory hits (e.g. L1 D-Cache misses) due to a reduction in the number of conflict misses.

3.2 Distribution of Misses among EAs and PCs

Another goal of this study was to determine how the L1 misses were spread out among the EAs and PCs. In other words, what percentage of the total L1 misses are solely due to the hot EAs (PCs)? Figures 2 and 3 show the distribution of misses among EAs and PCs, respectively. The cache configuration in Figures 2A and 3A is an 8KB, 1-way L1 D-Cache and a 128KB, 2-way L2 Cache while the cache configuration in Figures 2B and 3B is a 32KB, 4-way L1 D-Cache and a 512KB, 8-way L2 Cache.

In these figures, the EAs (PCs) were first sorted in descending order of their L1 D-Cache misses. Only EAs and PCs that missed at least once are included in this graph because EAs and PCs that do not miss have relatively no effect on the execution time. After sorting, the results were then graphed. The x-axis shows the percentage of EAs (PCs) while the y-axis shows the percentage of L1 misses. Since the EAs (PCs) were first sorted in descending order of their L1 misses, the EAs (PCs) with the most misses are shown on left side of each figure.

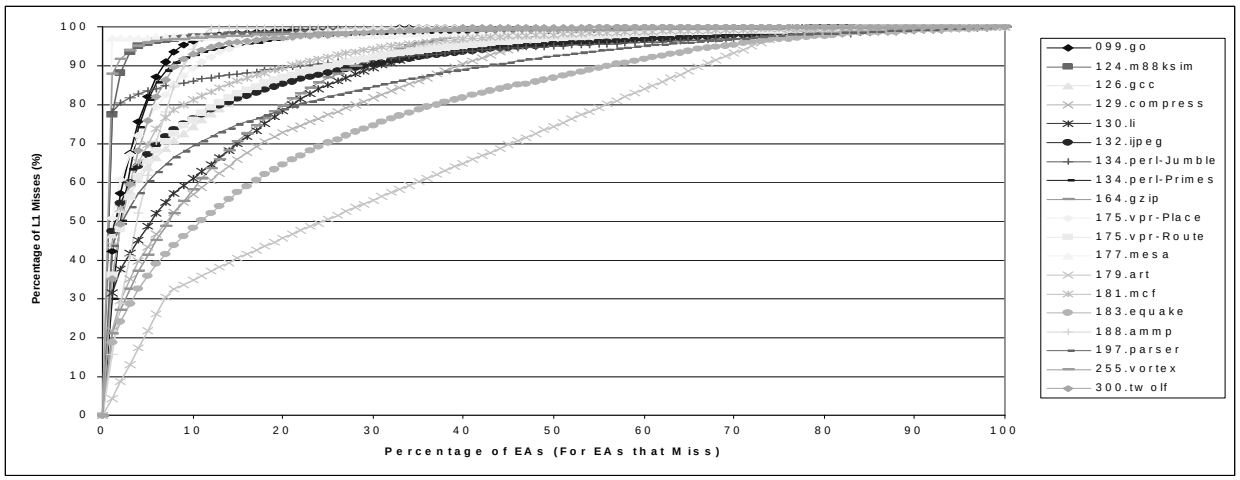

Figure 2A: 8KB, 1-Way L1 D-Cache and 128KB, 2-Way L2 Cache

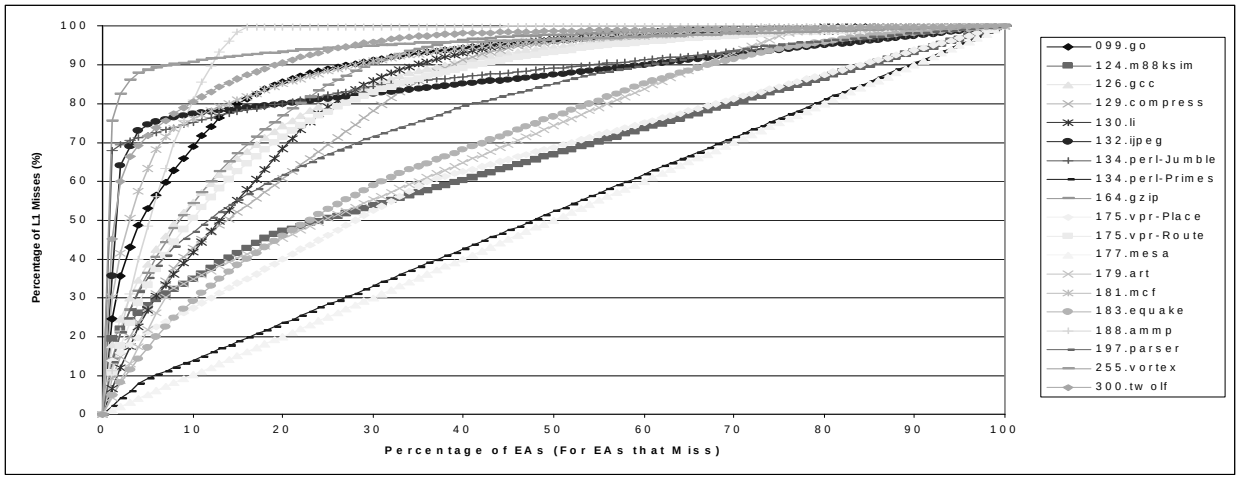

Figure 2B: 32KB, 4-Way L1 D-Cache and 512KB, 8-Way L2 Cache

Figure 2: Distribution of L1 Misses Among EAs; Only for EAs that Misses at Least Once

These figures show two very important results. First of all, in both configurations, a disproportionately high percentage of the L1 misses are due to a very small percentage of EAs. In 177.mesa, for instance, for the smaller and less associative L1 D-Cache, less than 1% of the EAs accounted for 97.08% of the total L1 misses. Therefore, a very small percentage of the EAs are hot spots. This conclusion can be found by examining the left-hand side of either graph. The sharp rise that is present in both graphs indicates that a large percentage of the L1 misses are concentrated in just a few EAs.

The second result is that increasing the L1 D-Cache size and associativity have very different effects on the hot EAs for each benchmark. For some benchmarks, such as 255.vortex, increases in the L1 D-Cache size and associativity have only a small effect on the percentage of L1 misses due to the hot EAs. For 255.vortex, using the 8KB, 1-way L1 D-Cache configuration, less than 1% of the EAs accounted for over 88% of the L1 misses. For the 32KB, 4-way L1 D-Cache configuration, the same percentage of EAs still accounted for over 75% of L1 misses. On the other hand, for some benchmarks, the change in the cache configuration almost completely

eliminates the memory hot spots. In the case of 177.mesa, the change in the cache configuration alters the distribution of L1 misses so that all EAs have approximately the same number of misses (i.e. no hot EAs). Finally, for some benchmarks, the change in the cache configuration greatly reduces the number of L1 misses due to the hot EAs, although these EAs still account for a significant percentage of the L1 misses. An example of this behavior is found in 181.mcf. For the 8KB, 1-way L1 D-Cache, less than 1% of the EAs account for 43.90% of the L1 misses. For the 32KB, 4-way L1 D-Cache, the same percentage of EAs account for 30.23% of the L1 misses. Therefore, from all of these results, we expect that further increases in the cache size and associativity will continue to decrease the number of misses due to the hot EAs, but not dramatically so.

Putting these two results together shows that, for almost all benchmarks, a very small percentage of EAs accounts for a very significant percentage of the total L1 misses, regardless of the L1 D-Cache size and associativity. For the 8KB, 1-way L1 D-Cache, in 18 of the 19 benchmarks, less than 1% of all EAs accounted for at least 15.69% of all the L1 misses. For the 32KB, 4-way L1 D-Cache, in 14 of the 19 benchmarks, the same percentage of EAs accounted for at least 9.23% of all the L1 misses. While these results were expected, we were somewhat surprised to see that the actual percentage of L1 misses that were due to these hot EAs was as high as it was.

Figures 3A and 3B are the PC equivalent versions of Figures 2A and 2B. They show that a relatively small number of hot PCs account for a disproportionately large percentage of L1 misses (similar observations have been made in [Abraham94 and Collins01-1]) and that increasing the L1 D-Cache size and associativity does not dramatically decrease the number of L1 misses due to these hot PCs. For the 8KB, 1-way L1 D-Cache, in all benchmarks, less than

1% of all PCs accounted for at least 25.53% of all the L1 misses. For the 32KB, 4-way L1 D-Cache, in all benchmarks, the same percentage of PCs accounted for at least 20.53% of all the L1 misses.

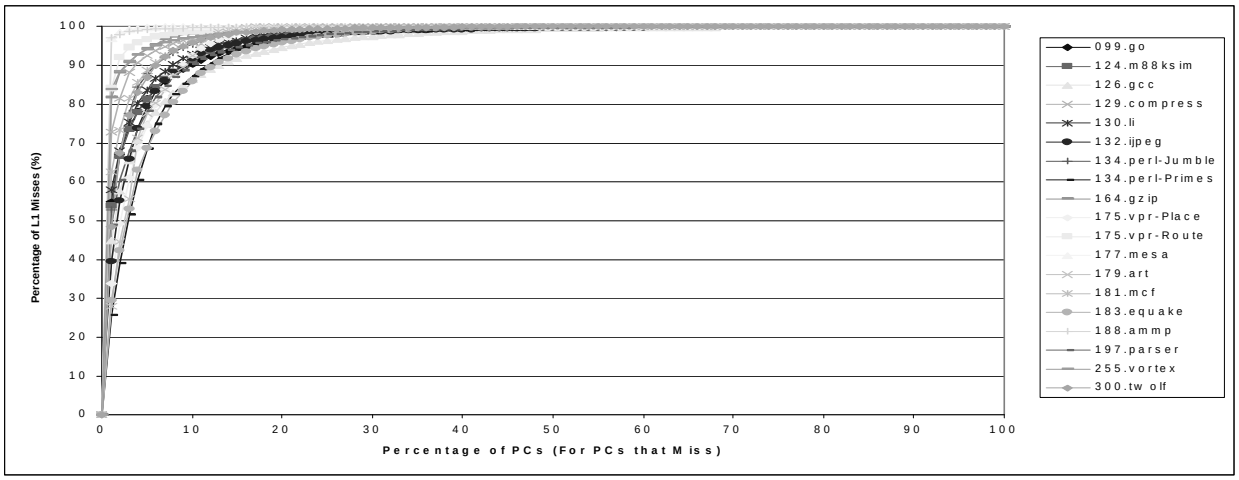

Figure 3A: 8KB, 1-Way L1 D-Cache and 128KB, 2-Way L2 Cache

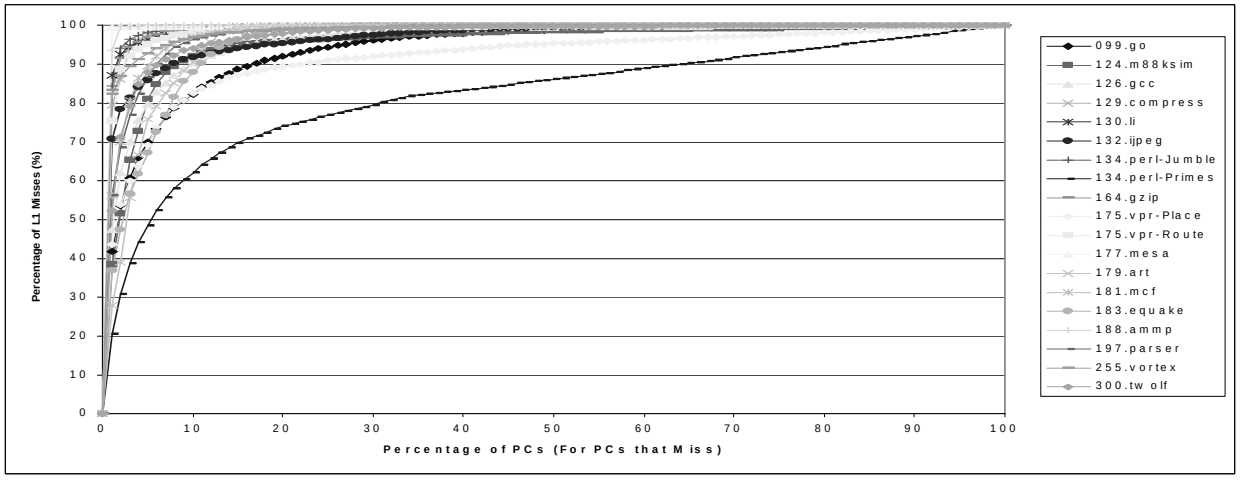

Figure 3B: 32KB, 4-Way L1 D-Cache and 512KB, 8-Way L2 Cache

Figure 3: Distribution of L1 Misses Among PCs; Only for PCs that Misses at Least Once

However, there are three key differences between the hot EAs and the hot PCs. First of all, the hot PCs account for a much higher percentage of the L1 misses than the same number of

hot EAs do. Second, the hot PCs account for approximately the same percentage of L1 misses across all benchmarks. Finally, the cache configuration does not reduce the percentage of L1 D-Cache misses due to the hot PCs as much as it does for the hot EAs, i.e. the percentage of misses due to the hot PCs is comparatively unaffected by the cache configuration.

3.3 Characteristics of the Loads with Top 16 L1 Misses

The previous subsection showed that the hot PCs account for a higher percentage of the L1 misses, as compared to the hot EAs, and that those hot PCs are less affected by the cache configuration, again, compared to the hot EAs. This result is not particularly surprising since many of the hot PCs load from several EAs, a few of which are hot EAs. The following tables show the number of L1 misses for the hot EAs and hot PCs for two different cache configurations. These tables show the results for 164.gzip, which was a benchmark that was approximately in the "middle" of all the benchmarks in Figures 2 and 3.

Table 5: Top 16 Hot EAs, by L1 Misses, in 164.gzip; L1 D-Cache Configurations: 8KB, 1-Way and 32KB, 4-Way

8KB, 1-Way			32KB, 4-Way		
EA	L1 Misses	Number of PCs	EA	L1 Misses	Number of PCs
268453560	74381	266	268636576	8137	224
268453528	41727	1061	268661876	7295	100
268453780	37522	1714	268658224	7292	196
268453572	37168	410	**268637296**	6957	205
268451660	36466	1	268663932	6799	50
268451572	30463	189	**268685184**	6781	102
268685184	21303	173	268690668	6694	68
268668800	20898	116	**268668800**	6635	75
268676528	18987	153	268674288	6483	52
268656376	18873	274	**268663284**	6230	67
268680944	18860	170	268680372	6228	56
268637296	18634	349	268683056	6213	133
268663284	18504	98	268682692	6070	101
268664172	18485	100	268676732	6013	86
268639788	18329	78	268691860	5930	69
268688948	18262	87	268672168	5912	36

Table 5 shows two key results. First of all, with the exception of 4 EAs (shown in bold in Table 5), the hot EAs that are present for each cache configuration are different. Increasing the cache size and associativity has the effect of reducing the L1 misses for some hot EAs more than for others. Therefore, the set of hot EAs changes. Second, almost all the EAs in Table 5 are referenced by at least 36 different PCs. That is, a single EA is referenced by at least 36 different instructions. This result suggests that these EAs may have some temporal locality.

Table 6: Top 16 Hot PCs, by L1 Misses, in 164.gzip; L1 D-Cache Configurations: 8KB, 1-Way and 32KB, 4-Way

8KB, 1-Way			32KB, 4-Way		
PC	L1 Misses	Number of EAs	PC	L1 Misses	Number of EAs
4198912	5161474	762658	4198912	2918182	452989
4198352	4551759	904866	4198352	2238433	498674
4202040	2331656	47834	4202040	851276	32810
4198384	350186	118449	4199536	317783	11592
4199536	319006	11646	4199664	317657	27409
4199664	318185	25175	4268480	161152	3080
4198368	301311	96587	4198384	141507	48139
4268480	184197	5862	4314832	140568	4943
4269256	183016	4242	4199352	137584	4364
4314832	176988	6671	4201536	110311	23916
4201536	176626	35363	4314800	105786	43271
4199352	156896	4610	4198368	99066	35628
4314800	102328	36062	4314768	94260	34115
4314768	100120	34595	4269256	93517	1360
4201992	89341	18933	4200240	51239	25337
4200240	86023	36045	4247656	24657	4241

As shown in Tables 5 and 6, the number of L1 misses due to the 16 hottest PCs is much higher than the number of L1 misses due to the hottest 16 EAs. As in the case of the EAs, increasing the L1 D-Cache size and associativity dramatically decreases the number of L1 misses due to these PCs. However, the number of misses for each PC is still very significant. Another difference between the hot EAs and PCs is that almost all of the hot PCs are the same when using either cache configuration. Therefore, in the case of hot PCs, changing the cache

configuration decreases the number of misses for the 16 hottest PCs, but it does not significantly change which PCs are hot. Finally, the number of EAs that are touched by each of these hot PCs is very high, at least 1360. Therefore, in the absence of a stride access pattern, this characteristic would make it extremely difficult to consistently prefetch the value for any particular PC.

Finally, we note that the results in Tables 5 and 6 are fairly representative of the results for all benchmarks.

3.4 EAs per PC and PCs per EA

While the preceding subsection showed the number of EAs that are referenced by a hot PC and the number of PCs that reference a hot EA, this section generalizes that result to all EAs and PCs. Figure 4 is a histogram of the average number of EAs that are referenced by each PC, across all configurations, while Figure 5 is a histogram of the average number of PCs that reference each EA, again, across all configurations. Note that the x-axis bin ranges are logarithmic. Due to space limitations, only 164.gzip is shown. However, the results from the other benchmarks are quite similar.

Figure 4 shows that a sizeable percentage of the PCs reference hundreds or thousands of different EAs. As a result, in the absence of a significant amount of spatial locality, these PCs will produce a large number of L1 or L2 misses.

In Figure 4, the peaks at 10 and 100 are due the logarithmic nature of the histogram. For x-axis values greater than 9, the width of each bin is not just a single number, but, rather, a range of numbers. For instance, the “10” bin represents 10-19 while the “100” bin represents 100-999. As a result, the peaks are merely the result of using a range of numbers that is 10 times larger than the range of numbers to its immediate left.

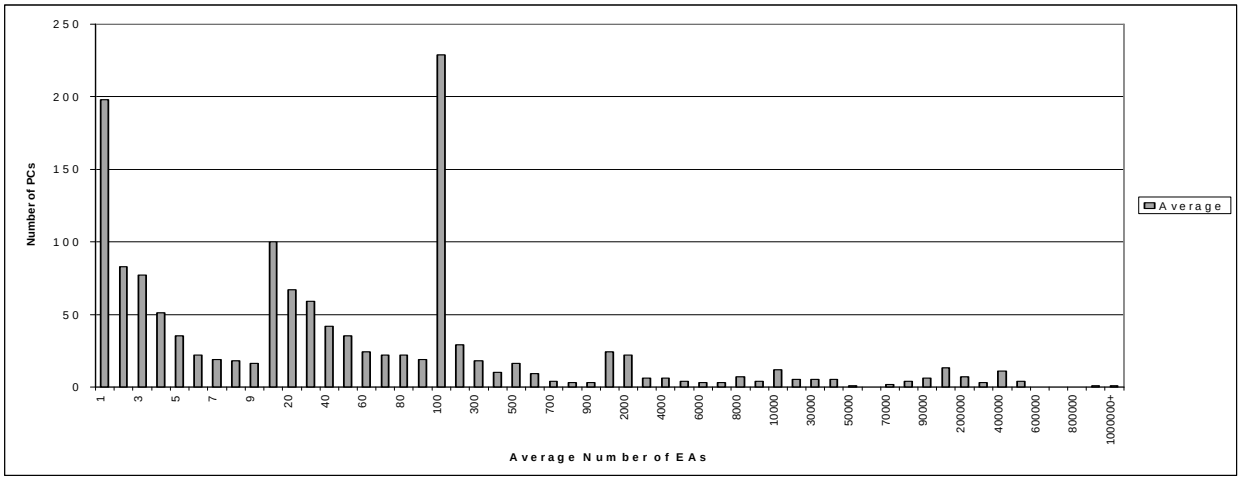

Figure 4: Average Number of EAs Referenced per PC; Results for 164.gzip

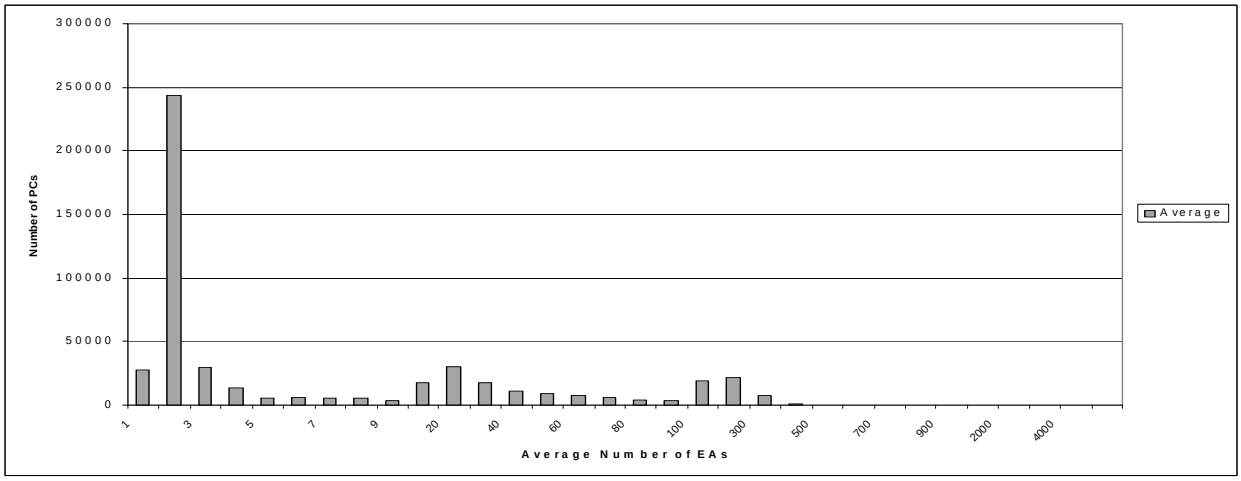

Figure 5: Average Number of PCs that Reference Each EA; Results for 164.gzip

In stark contrast to the results in Figure 4, the results in Figure 5 show that very few different PCs reference each EA. In fact, most of the unique EAs are referenced by only two different PCs.

3.5 The Effect of Program Inputs

This section examines the effect of the benchmark's inputs on the load characteristics that

were presented in the previous four subsections. In general, a comparison of the results for the two input sets showed that the input set did not alter the conclusion that was drawn for each type of analysis. There were two exceptions to this general conclusion. First, in the case of the hit distribution across the different levels of the memory hierarchy (Figure 1), the results for both input sets were fairly similar. The biggest difference was in the percentage of hits to the LSQ. For some benchmarks, the LSQ hit rate was significantly higher for one input set as compared to the other. However, the higher percentage of LSQ hits did not consistently occur in the input set that had the higher instruction count. For example, in 124.m88ksim, 130.li, and 188.ammp, the percentage of LSQ hits was significantly higher for the input set that had the lower instruction count. On the other hand, for 177.mesa and 179.art, the input set that had the higher instruction count was the one that had the higher percentage of LSQ hits. We leave a detailed examination of this phenomenon to future work.

While the LSQ hit rate can differ markedly for different input sets, the combined hit rate for the LSQ and L1 D-Cache remains almost constant across input sets. Therefore, since the LSQ and L1 D-Cache have the same hit latency in this architecture, there is no performance degradation due to a lower LSQ hit rate when there are sufficient memory ports.

The other exception to the general conclusion of invariance for these load characteristics across input sets is which specific EAs and PCs are the hot EAs and hot PCs, respectively. In other words, using a different input set may result in a different group of hot EAs or hot PCs. While there may be a significant number of EAs or PCs that have a high number of L1 misses independent of the program's input set, some specific EAs or PCs have a high number of L1 misses only when using a certain input set. Table 7 shows the number of hot EAs or PCs that are common to both input sets and that ranked among the Top 64 EAs or PCs in terms of the highest

numbers of L1 misses, for either input set. In other words, Table 7 shows the number of hot EAs and PCs that are hot for both input sets. The second and fourth columns show the number of hot EAs and PCs, respectively, that are common in both input sets. The third and fifth columns show the percentage of the Top 64 EAs and PCs that are represented by the number of EAs and PCs in the second and fourth columns, respectively. Since only the hottest 64 EAs and PCs are compared across input sets, the maximum number of EAs and PCs that are common between both input sets is 64. Finally, since three separate inputs are used as the inputs for 134.perl, for completeness, Table 7 shows all three possible comparisons.

Table 7: Number of Hot EAs and Hot PCs that are Present in Both Input Sets

Benchmark	# of Common EAs	% of Top 64	# of Common PCs	% of Top 64
099.go	7	10.94	57	89.06
124.m88ksim	0	0.00	26	40.63
126.gcc	2	3.13	48	75.00
129.compress	0	0.00	50	78.13
130.li	0	0.00	35	54.69
132.ijpeg	3	4.69	40	62.50
134.perl-Jumble vs. Scrabbl	1	1.56	29	45.31
134.perl-Primes vs. Scrabbl	0	0.00	18	28.13
134.perl-Jumble vs. Primes	1	1.56	15	23.44
164.gzip	55	85.94	62	96.88
175.vpr-Place	25	39.06	38	59.38
175.vpr-Route	1	1.56	50	78.13
177.mesa	6	9.38	15	23.44
179.art	13	20.31	58	90.63
181.mcf	2	3.13	55	85.94
183.equake	34	53.13	55	85.94
188.ammp	63	98.44	58	90.63
197.parser	8	12.50	17	26.56
255.vortex	8	12.50	46	71.88
300.twolf	0	0.00	46	71.88

With the exceptions of 164.gzip, 175.vpr-Place, 179.art, 183.equake, and 188.ammp, the number of hot, common EAs is relatively low (7 or less). On the flip side, with the exceptions of 134.perl-Primes vs. Scrabbl, 134.perl-Jumble vs. Primes, 177.mesa, and 197.parser, the number of hot, common PCs is fairly high (26 or more). While the number of hot, common PCs is

relatively low for 134.perl-Primes vs. Scrabbl, 134.perl-Jumble vs. Primes, 177.mesa, and 197.parser, the number of hot, common PCs for these benchmarks is still significant. On the other hand, the number of hot, common EAs is astonishingly high for 164.gzip, 175.vpr-Place, 179.art, 183.equake, and 188.ammp. We think that these two results are important enough that they could form the basis for alternative caching structures.

In summary, the input set does not dramatically affect the load behavior for most of the benchmarks and configurations. The two exceptions to this general conclusion are the different LSQ hit rates across input sets and the number of hot, common EAs and PCs for a select group of benchmarks.

4 Related Work

There have been several studies that evaluated the cache performance for the SPEC CPU benchmarks (92, 95, and 2000). Prior studies [Gee93, Charney97, Sair00, and Cantin01] have shown that the SPEC benchmarks do not significantly stress the memory sub-system. Our profiling results on a collection of SPEC 95 and SPEC 2000 benchmarks confirm this conclusion by showing that only a few benchmark programs place more than modest stress on the memory sub-system.

Gee *et al* [Gee93] reported the cache miss rates for the SPEC 92 benchmark suite while Charney and Puzak repeated this study for the SPEC 95 benchmarks. In addition to these cache miss rates, Charney and Puzak also studied the prefetching behavior of the SPEC 95 benchmarks. Sair and Charney [Sair00] analyzed the cache miss rates for SPEC 2000 benchmark suite and confirmed the results from [Gee93 and Charney97]. Finally, Cantin and Hill [Cantin01] presented cache miss rates for benchmarks from the SPEC 2000 benchmark suite

for various reference input sets and cache configurations. In comparison, the primary goal of these four studies was to examine the cache miss rates of the three SPEC benchmark suites while the primary goal of this study was to present a comprehensive analysis of the load behavior, from the perspective of the memory subsystem. The complete set of results for this study can be found in [Yi02].

Finally, a number of profiling tools have been proposed [Lebeck94, Anderson97, Mowry97, and Thornock00] to monitor the program caching behavior (cache bottlenecks, hot misses, functional unit contention, etc.). All of these profiling tools use sampling and are memory-bound since they require substantial amounts of memory to store the sampled data before it is processed. In our study, we use a comprehensive, continuous method of profiling that fully captures the load behavior for the entire program.

5 Conclusion and Future Work

This study shows the spatial characteristics of static and dynamic load instructions in the context of a whole program by examining: 1) The hit distribution of dynamic loads across the different levels of the memory hierarchy, 2) The distribution of misses among the effective addresses (EAs) and PCs that miss, 3) The characteristics of the 16 EAs and PCs that had the highest numbers of L1 misses, 4) The number of EAs that are referenced by each PC and the number of PCs that reference each EA, and 5) The effect of the input set on the preceding four examinations. Furthermore, since a primary goal of this study was to provide firm foundation for future work on the memory hierarchy, this study also examined the effect that the L1 D-Cache and L2 Cache configuration had on the previously listed examinations.

This study showed that almost all loads hit in either the load-store queue (LSQ) or the L1

D-Cache, with the exception of 179.art. For this benchmark, the number of L2 hits and main memory "hits" was comparable to the number of LSQ and L1 D-Cache hits. These results confirm conventional thought on the expectation of how the hits are distributed across the different levels of the memory hierarchy. Surprisingly, the percentage of LSQ hits was greater than 20% of all the dynamic loads for several benchmarks, which was contrary to expectations.

This study also showed that the majority of L1 misses were due to a disproportionately small percentage of EAs and PCs. This result somewhat fulfilled our initial expectations; the only surprise was that the hot EAs accounted for a higher percentage of the L1 misses than expected. Additionally, when the cache configuration was increased in both size and associativity, the percentage of L1 misses due to these hot EAs and PCs did not change dramatically for most benchmarks.

This study also showed that the 16 hottest PCs reference thousands of EAs while the 16 hottest EAs were referenced by tens of PCs. Furthermore, this set of PCs accounted for more L1 misses than the set of EAs did. Additionally, this study showed that the average PC generally references tens and hundreds of EAs while the average EA will be referenced by only a few PCs. Again, these results agreed with conventional thought.

Finally, our results showed that the program's input has relatively little impact on the preceding load characteristics. The key exceptions are that the LSQ hit rate can vary significantly based on the input set and that the number of hot EAs and PCs that are common to both input sets are comparatively low and high, respectively, with a few notable exceptions.

As items of future work, we are currently performing a similar set of examinations to determine the temporal characteristics of load instructions. We think that the combination of this study, plus any extensions to it, coupled with the temporal characterization will provide an

invaluable resource and foundation to future memory sub-system improvements. Furthermore, we plan on performing the same analyses for L2 misses and answering a few outstanding questions.

References

[Abraham94] S. Abraham and B. Ray; “Predicting Load Latencies Using Cache Profiling”; Hewlett-Packard Technical Report HPL-94-110, 1994.

[Anderson97] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S. Leung, R. Sites, M. Vandevoorde, C. Waldspurger, and W. Weihl; “Continuous Profiling: Where Have All the Cycles Gone?”; Symposium on Operating Systems Principles, 1997.

[Annavaram00] M. Annavaram, J. Patel, and E. Davidson; "Call Graph Prefetching for Database Applications"; International Symposium on High Performance Computer Architecture, 2000.

[Annavaram01] M. Annavaram, J. Patel, and E. Davidson; "Data Prefetching by Dependence Graph Precomputation"; International Symposium on Computer Architecture, 2001.

[Balasubramonian01] R. Balasubramonian, S. Dwarkadas, and D. Albonesi; "Dynamically Allocating Processor Resources between Nearby and Distant ILP"; International Symposium on Computer Architecture, 2001.

[Burger97-1] D. Burger and T. Austin; “The SimpleScalar Tool Set, Version 2.0”; University of Wisconsin Computer Sciences Department Technical Report 1342.

[Burger97-2] D. Burger, J. Goodman, and A. Kagi; "Limited Bandwidth To Affect Processor Design"; IEEE Micro, Vol. 17, No. 6, November-December 1997; Pages 55-62.

[Cantin01] J. Cantin and M. Hill; “Cache Performance for SPEC CPU2000 Benchmarks”; Computer Architecture News, Vol. 29, No. 4, September 2001.

[Charney97] M. Charney and T. Puzak; “Prefetching and Memory System Behavior of the SPEC95 Benchmark Suite”; IBM Journal of Research and Development, Vol. 41, No. 3, May 1997.

[Chen97] I. Chen, C. Lee, and T. Mudge; "Instruction Prefetching Using Branch Prediction Information"; International Symposium on Microarchitecture, 1997.

[Collins01-1] J. Collins, H. Wang, D. Tullsen, C. Hughes, Y. Lee, D. Lavery, J. Shen, "Speculative Precomputation: Long-range Prefetching of Delinquent Loads"; International Symposium on Computer Architecture, 2001.

[Collins01-2] J. Collins and D. Tullsen, “Runtime Identification of Cache Conflict Misses: The Adaptive Miss Buffer,” ACM Transactions on Computer Systems, Vol. 19, No. 4, November 2001, pp. 413-439.

[Collins01-3] J. Collins, D. Tullsen, H. Wang, and J. Shen; "Dynamic Speculative Precomputation"; International Symposium on Microarchitecture, 2001.

[Gee93] J. Gee, M. Hill, D. Pnevmatikatos, and A. Smith; “Cache Performance of the SPEC92 Benchmark Suite”; IEEE Micro, Vol. 13, No. 4: Pages 17-27, August 1993.

[Kim97] S. Kim and A. Veidenbaum; "Stride-directed Prefetching for Secondary Caches"; International Conference on Parallel Processing, 1997.

[KleinOsowski00] A. KleinOsowski, J. Flynn, N. Meares, and D. Lilja; "Adapting the SPEC 2000 Benchmark Suite for Simulation-Based Computer Architecture Research"; Workload Characterization of Emerging Computer Applications, L. Kurian John and A. M. Grizzaffi Maynard (eds.), Kluwer Academic Publishers, Pages 83-100, 2001.

[Lebeck94] A. Lebeck and D. Wood; “Cache Profiling and the SPEC Benchmarks: A Case Study”; IEEE Computer, Vol. 27, No. 10, Pages 15-26, October 1994.

[Luk01] C. Luk; "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors"; International Symposium on Computer Architecture, 2001.

[Mowry97] T. Mowry and C. Luk; “Predicting Data Cache Misses in Non-Numeric Applications Through Correlation Profiling”; International Symposium on Microarchitecture, 1997.

[Merten99] M. Merten, A. Trick, C. George, J. Gyllenhaal, and W. Hwu; “A Hardware-Driven Profiling Scheme for Identifying Program Hot Spots to Support Runtime Optimization”; International Symposium on Computer Architecture, 1999.

[Pierce96] J. Pierce and T. Mudge; "Wrong-Path Instruction Prefetching"; International Symposium on Microarchitecture, 1996.

[Roth98] A. Roth, A. Moshovos, and G. Sohi; "Dependence Based Prefetching for Linked Data Structures"; International Conference on Architectural Support for Programming Languages and Operating Systems, 1998.

[Sair00] S. Sair and M. Charney; “Memory Behavior of the SPEC2000 Benchmark Suite”; IBM Research Report, RC 21852 (98345), 2000.

[Thornock00] N. Thornock and J. Flanagan; “Using the BACH Trace Collection Mechanism to Characterize the SPEC 2000 Integer Benchmarks”; Workshop on Workload Characterization, 2000.

[Yang00] C. Yang and A. Lebeck; "Push vs. Pull: Data Movement for Linked Data Structures"; International Conference on Supercomputing, 2000.

[Yi02] J. Yi, R. Sendag, and D. Lilja; "The Spatial Characteristics of Load Instructions"; www-mount.ece.umn.edu/~jjyi/Mem_Prof/index.html

[Zilles00] C. Zilles and G. Sohi; "Understanding the Backward Slices of Performance Degrading Instructions"; International Symposium on Computer Architecture, 2000.

[Zilles01] C. Zilles and G. Sohi; "Execution-based Prediction Using Speculative Slices"; International Symposium on Computer Architecture, 2001