

ABSTRACT

EKER, ABDULAZIZ. Characterization of Context Switch Effects on L2 Cache. (Under the direction of Dr. Yan Solihin.)

Multitasking is common in most systems. In order to use the processor resources efficiently, a multitasking system schedules processes to run for certain intervals by switching (saving and restoring) their contexts. However, since processes bring their own data to the cache when they are running, context switching causes each process to suffer from more cache misses. The behavior of L2 cache misses due to context switches activities under different cache configurations, working-set sizes, and process priorities has not been sufficiently investigated. Analysis of this behavior will give insights about the reasons and ways to mitigate these misses.

The first contribution of this paper is the characterization of how context switch misses at the L2 cache is affected by process priorities. The paper also characterizes the context switch effect with various cache configurations, including the size and associativity of the cache. Finally, it defines two types of misses that occur due to context switches. *Replacement context switch misses* occur when a process' working set is replaced by an interfering process. *Reorder context switch misses* occur due to reordering of lines by an interfering process, i.e. moving lines from more recently used to less recently used position. Through the characterization, we found that the number of context switch misses significantly increases with lower priorities. On average, on a Linux operating system, a process with the lowest priority suffers 15.4× more L2 cache misses due to the context switch effect compared to when time-sharing is not used, while the process with a relatively higher priority suffers only 1.2× more misses. In addition, we observed that the impact of context switch is affected more by the priority of the process itself, rather than the priority of the interfering process. We also observed that reorder context switch misses increase with higher associativity. Finally, the context switch effect is strongest when a process' working set size is close to the cache size.

Characterization of Context Switch Effects on L2 Cache

by

Abdulaziz Eker

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

COMPUTER ENGINEERING

Raleigh, NC

2007

Approved By:

Dr. Ed Gehringer

Dr. Suleyman Sair

Dr. Yan Solihin
Chair of Advisory Committee

Dedication

Dedicated to my family ...

Biography

Abdulaziz Eker was born on May 30, 1982, in Samsun, Turkey. In May 2005, he received his Bachelors degree in Computer Engineering from Purdue University.

He enrolled at North Carolina State University in August 2005, to begin work on his Masters degree in Computer Engineering. He joined Dr. Yan Solihin's research group in January 2006.

Acknowledgements

First of all, I would like to thank my parents, Idris and Sabiha Eker, and my brother Ahmet. Their endless support and guidance both in life and in my academic career have been such a blessing to me. They have made so many things so easy for me, and I could never fully express what they have meant to me. Without their help, none of this would have been possible for me.

I would like to thank Dr. Yan Solihin for giving me the opportunity to work under his guidance and support. I have learned countless things in the past year while doing research, including much about computer architecture, how to conduct research effectively and efficiently, and how to more effectively give oral presentations. The concern and time that Dr. Solihin puts into preparing his students for their future is always evident, and I want to thank him for that as well.

I would like to thank Dr. Suleyman Sair for his great support, for always being helpful to me, and for agreeing to be on my advising committee. I would like to thank Dr. Ed Gehringer for his suggestions and help during my teaching assistantship with him, and for agreeing to be on my advising committee.

I would like to thank Seongbeom Kim, Xiaowei Jiang, Abhik Sarkar, Fei Guo and Brian Rogers who were never too busy to offer advice or suggestions to problems that I encountered during the past year. I would also like to thank Ganghee Jang, Fang Liu, and Siddhartha Chhabra for their help and support.

Finally, I would like to thank Zuleyha Emirza for being such a special part of my life over the past year, and for helping to keep me focused and motivated on my work.

Contents

List of Figures	vi
List of Tables	vii
1 Introduction	1
2 Characterization Environment	4
2.1 Simulation Environment	4
2.2 Applications	4
2.3 Evaluation Methodology	6
3 Characterization	10
3.1 Impact of Process Priorities	10
3.2 Sensitivity to Cache Parameters	14
3.2.1 Cache Size	14
3.2.2 Cache Associativity	17
4 Summary and Conclusions	19
Bibliography	20

List of Figures

2.1	Illustration of types of context switch misses. (a) shows the timeline when process A runs alone, and (b) shows the timeline when process A is time-sharing the cache with process B.	7
2.2	Structure of the simulated model.	8
3.1	Number of L2 cache misses of each primary process broken down into context switch (CS) misses, capacity and conflict misses, and cold misses, for different timeslice durations, normalized to the 4m timeslice duration case.	11
3.2	The cumulative number of L2 misses over time when vpr (primary process) is run with parser (secondary process), with 92ms timeslice (a) and with 44ms timeslice (b). The context switch boundaries are indicated by the vertical lines, and at each boundary the cumulative cache miss counter is reset.	12
3.3	The number of context switch misses as a function of the priorities (timeslice durations) of the primary and secondary benchmarks. The timeslice duration is shown as x ms- y ms, where x is for the primary benchmark and y is for the secondary benchmark.	13
3.4	Number of L2 cache misses of each primary process broken down into context switch (CS) misses, capacity and conflict misses, and cold misses, for L2 cache sizes ranging from 256KB to 4MB, normalized to the 256KB case.	14
3.5	The fraction of all L2 cache misses that are context switch misses.	15
3.6	Miss Rates for Different Cache Sizes. Each figure also shows the working set size point where the slope of consequent points decreased significantly in circle.	16
3.7	Number of L2 cache misses of each primary process broken down into reorder context switch misses, replacement context switch misses, capacity and conflict misses, and cold misses, for L2 cache associativities ranging from 2-way to 32-way, normalized to the 2-way case.	18

List of Tables

2.1	Simulation Parameters	5
2.2	Spec2k Benchmark Categorization	5

Chapter 1

Introduction

Multiprogramming or multitasking is common in modern Operating Systems (OS) [10]. In order to use the processor resources efficiently, a multitasking system allows multiple processes to time-share a processor by scheduling them to run for certain intervals and switching (saving and restoring) their contexts between intervals. While multitasking enables higher processor utilization, it also incurs several overheads such as the execution of the scheduler, latency of saving and restoring the processor state, and interference of multiple processes' working sets in the caches. Among these overheads, the most significant one is the cache interference because of the growing processor-memory speed gap [12], and the growing size of working sets [4] and lowest level on-chip caches. When a process is switched out by another process, the working set that belongs to the former is likely to be replaced or nearly replaced by the latter because the latter also brings its own working set into the cache. When the former process is switched back and it tries to access its data, it suffers extra cache misses, either because its data has been prematurely replaced by the interfering process, or because its data is less *fresh* (closer to the LRU position in the cache). We refer to this type of cache misses as *context switch misses*.

Understanding the effect of context switch is important for several reasons. First, growing on-chip cache sizes implies that more data can be stored on-chip, but also implies that more data may be lost due to context switch cache interference. Second, it takes longer for the application to re-warm up a larger cache when the application resumes execution. Third, this cache re-warming may incur excessive off-chip bandwidth consumption, while off-chip bandwidth is already a very limited resource in a multi-core chip. Fourth, with growing processor-memory speed gap, the cache re-warming will take longer time to com-

plete. Finally, application working set size is growing, making it likely for an application to need every bit of cache space available to it.

Past studies by Agarwal et al. [2], Smith [9], and Mogul and Borg [8], have attempted to characterize the context switch effect on caches. However, because these studies were performed more than 15 years ago, they used assumptions that are no longer realistic in current systems, such as not simulating the OS layer, using offline traces rather than dynamic execution traces, assuming a single-level cache, and assuming an older processor model. A more recent study was performed by Koka and Lipasti [6], which analyzed how context switch misses vary with cache sizes. The study, however, does not investigate the effect of OS priorities on context switch misses. In addition, the methodology for counting context switch misses only counts extra cache misses due to data that is replaced by an interfering process. This ignores another source of context switch misses that is caused by data that is not replaced from the cache but has become less fresh in the cache (closer to the LRU position in the cache). Less fresh cache blocks have an elevated chance to be replaced by the process itself when it resumes execution.

The first contribution of this paper is the characterization of how context switch misses at the L2 cache is affected by process priorities. The paper also characterizes the context switch effect with various cache configurations, including the size and associativity of the cache. Finally, it defines two types of misses that occur due to context switches. *Replacement context switch misses* occur when a process' working set is replaced by an interfering process. *Reorder context switch misses* occur due to reordering of lines by an interfering process, i.e. moving lines from more recently used to less recently used position. Unlike the study by Koka and Lipasti [6], we consider all sources of cache misses, while they only consider replacement context switch misses. Since the Operating System (OS) priority of a process is the key information used in allocating processor resources, it is important to understand how priorities affect context switch misses. To the best of our knowledge, this has not been done before. Through the characterization, we found that the number of context switch misses significantly increases with lower priorities. On average, on a Linux operating system, a process with the lowest priority suffers $15.4\times$ more L2 cache misses due to the context switch effect compared to when time-sharing is not used, while the process with a relatively higher priority suffers only $1.2\times$ more misses. In addition, we observed that the impact of context switch is affected more by the priority of the process itself, rather than the priority of the interfering process. We also observed that reorder context switch

misses increase with higher associativity. Finally, the context switch effect is strongest when a process' working set size is close to the cache size.

The rest of the paper is organized as follows. Section 2 describes the characterization setup, Section 3 presents the characterization results and discusses observations, and, Section 4 summarizes the findings and concludes.

Chapter 2

Characterization Environment

2.1 Simulation Environment

The characterization is performed using Simics [7], a full-system simulator that can boot and run an actual OS. The simulated system has a 4 GHz processor with x86 Instruction Set Architecture, running Fedora Core 5 [1] Linux distribution that has a Linux 2.6.15.1 kernel. We perform cycle-accurate execution-driven timing simulation by attaching a processor model which simulates an in-order architecture and a cache model which is based on Simics' `g-cache` module. Over the base processor and cache models, we added statistics collection and other instrumentation code. Table 2.1 shows the parameters used for each component of the architecture. The L2 cache configuration used for experiments is 2MB 8-way associative, unless mentioned otherwise. In this paper, we would like to focus on cache interference between applications, so the OS code is emulated (so its functional effect on thread scheduling is retained) but not simulated (so the timing impact of its execution is excluded).

2.2 Applications

To characterize the context switch effect on the L2 cache, we use benchmark pairs from SPEC2K benchmark suite [11] using reference input sets. We first categorized 17 benchmarks from SPECint and SPECfp based on their miss and hit frequency. Miss (or hit) frequency is measured as the number of L2 cache misses (or hits) per 1 ms (4 million cycles in a 4GHz processor). The L2 cache configuration used is 2MB and 8-way associativity.

Table 2.1: Simulation Parameters

Processor	In-order, 4GHZ, x86 ISA	256-KB	8-way
L1 Instruction	WT, 16 KB, 2 way, 64-B line size, 1 cycle hit latency, LRU replacement	512-KB	8-way
		1-MB	8-way
		2-MB	8-way
L1 Data	WT, 16 KB, 4 way, 64-B line size, 2 cycles hit latency, LRU replacement	4-MB	8-way
		2-MB	2-way
		2-MB	4-way
L2	WB, 64-B line size, 8 cycles hit latency, LRU replacement	2-MB	8-way
		2-MB	16-way
Memory	256 MB, 200 cycles latency	2-MB	32-way

(a) Parameters of the Simulated Architecture

(b) L2 Cache Configurations

Table 2.2: Spec2k Benchmark Categorization

Miss Frequency	Hit Frequency	Benchmarks
High	Low	<i>art</i> , <i>mcf</i> , <i>equake</i>
Low	High	<i>crafty</i> , <i>gzip</i> , <i>gcc</i>
Low	Low	<i>parser</i> , <i>vpr</i> , <i>bzip2</i> , <i>twolf</i> , <i>applu</i> , <i>gap</i> <i>vortex</i> , <i>ammp</i> , <i>apsi</i> , <i>swim</i> , <i>gap</i> , <i>perlbmk</i>

Miss frequency shows how frequently a benchmark brings data to the cache, while hit frequency shows how frequently it reuses its data. The reason for having both miss and hit frequencies is that previous research has shown that cache interference is determined by both frequencies [5]. The results of this categorization are shown in Table 2.2. A benchmark is considered to have a high miss frequency if the number of misses per 1ms is greater than 8,000, and a high hit frequency if the number of hits per 1ms is greater than 400,000. Note that we did not find any SPEC2K benchmarks that have simultaneously high hit and miss frequencies. From each category, we pick a few representative benchmarks (*art*, *bzip2*, *crafty*, *mcf*, *parser*, *twolf*, and *vpr*) and pair non-identical benchmarks. As a result, we have a total of 42 pairs of benchmarks. The L2 cache miss rates for these applications on

the standard cache configuration of 2MB 8-way associative are 6.67% (art), 0.50% (bzip2), 0.01% (crafty), 33.13% (mcf), 0.23% (parser), 0.01% (twolf), and 0.01% (vpr).

We simulate each benchmark pair such that both benchmarks time-share the system. We distinguish between the benchmark for which we collect context switch miss statistics (*primary* benchmark), from the interfering (or *secondary*) benchmark. To avoid simulating the initialization phase of both benchmarks, we first skip 4 billion instructions of the secondary benchmark, then start the primary benchmark and skip its first 4 billion instructions. Then we simulate 100 million instructions of the primary benchmark to warm-up the L2 cache without collecting statistics. After that is completed, we collect statistics for 1 billion instructions of the primary benchmark. Note that the same portion of the primary benchmark is simulated regardless of how many instructions the secondary benchmark executes.

2.3 Evaluation Methodology

We define the context switch misses of a process (thread) as misses that would only occur if the process time-shares the cache with other processes (threads). We further define two types of context switch misses: reorder and replacement context switch misses. Figure 2.1 illustrates the types of context switch misses. Figure 2.1 (a) shows the timeline when a process is not time-sharing the cache, and (b) shows when it is time-sharing the cache with an interfering process. If a process encounters a cache miss in time-shared system, but does not encounter a cache miss when no time-sharing is employed, that miss is called a context switch miss. In (b), process A brings a line into the cache. If that line is *replaced* by the interfering process B before process A reaccesses it, a *replacement* context switch miss occurs. If the line is changed from more recently used to less recently used by process B, and then replaced by process A before its access, a *reorder* context switch miss occurs.

In order to identify types of context switch misses, we use two L2 caches with the same configuration during each simulation. Figure 2.2 shows the structure of simulated model. One L2 cache (time-shared) receives all the transactions while other (private) receives only transactions of the primary process. A filter object is placed between L1 and L2 level caches. If a cache access results in a miss in a time-shared cache, but in a hit in a private cache, this is considered as a context switch miss. We added three flags to each line to track the type of context switch miss. The process ID (PID) flag tells which process

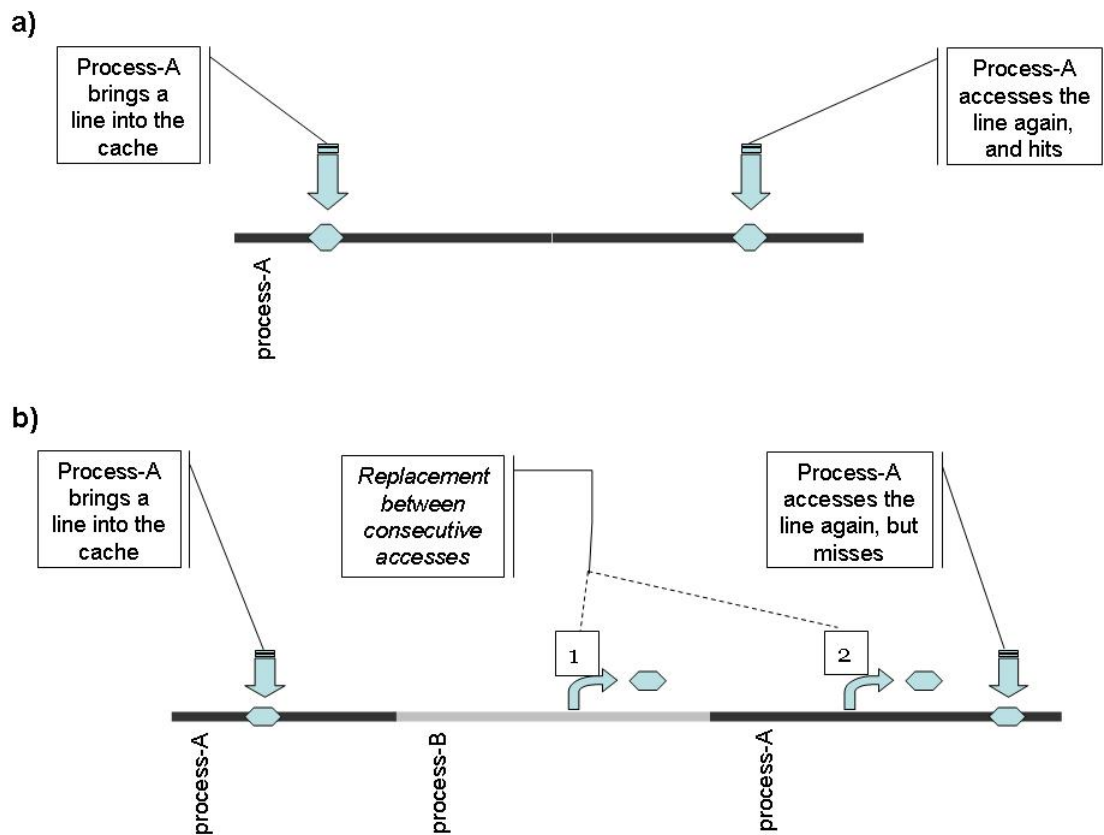


Figure 2.1: Illustration of types of context switch misses. (a) shows the timeline when process A runs alone, and (b) shows the timeline when process A is time-sharing the cache with process B.

the line belongs to. It is set based on the running process when the line is brought into the cache. The shared-only-removal (SOR) flag is set if a line is removed from shared cache, but not from private cache. The self-kill (SK) flag is set with the SOR flag if the line is removed by the owning process

When a line is removed from time-shared cache, the filter checks the private cache. If the line exists in the private cache, the filter sets the SOR flag of the line denoting that the line has only been removed from the time-shared cache. In addition, if the removed line's PID flag matches that of the running process, the SK flag of the line is set in the private cache denoting that the line has been removed by the owning process. Later in time, if a line is placed in the time-shared cache, the filter checks the same line in the private cache. If the line exists in the private cache and its SOR flag is set, the "context switch miss" counter is incremented. If the line's SK bit is also set, the "reorder context switch

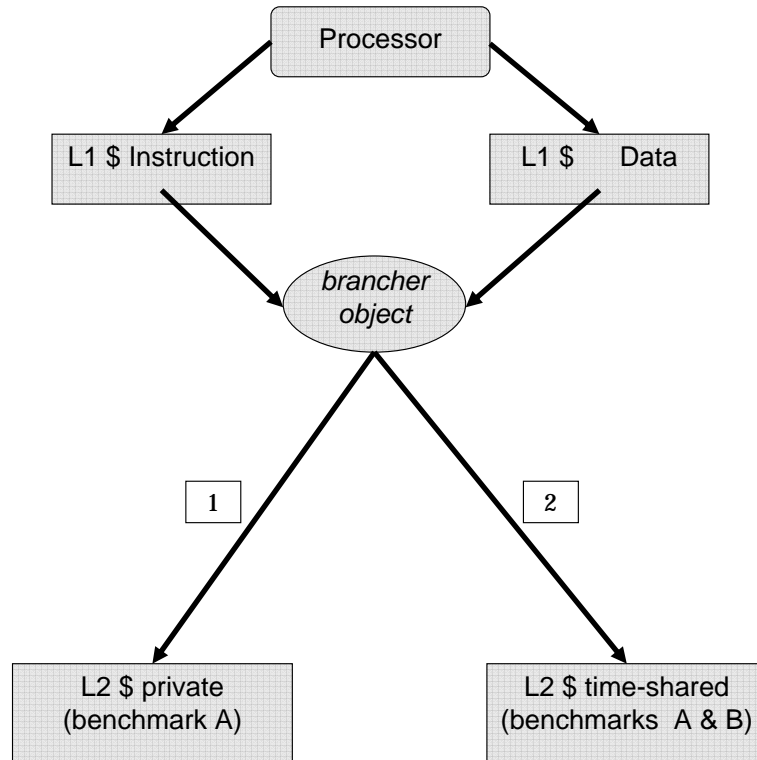


Figure 2.2: Structure of the simulated model.

miss” counter is incremented as well.

Due to the sharing of the L1 cache between the private and the time-shared caches we observed a few cases where the private L2 cache model has different numbers of misses when run with different priorities. This is because the L2 access stream is slightly changed. With the changed priority, the memory transactions received by L1 changes. Therefore, the misses encountered by the L1 cache change, resulting in a different access sequence seen by the private L2 cache.

To characterize the context switch effect with varying priorities, we assign 5 different priorities to the benchmarks. The priority of a process can be voluntarily *lowered* using the `nice` command (raising priorities requires administrator privilege so it is rarely performed for user programs). In this paper, we use 5 different nice values (1, 5, 11, 15, and 19). Linux 2.6 kernel determines the timeslice duration of a process solely based on its priority [3]. The timeslice duration is computed as $(20 - NiceValue) \times 5ms$, rounded down

to the nearest multiple of the timer tick (the default timer tick in Linux is 4ms). Hence, the nice value of 1, 5, 11, 15, and 19 correspond to timeslice duration of 92ms, 72ms, 44ms, 24ms, and 4ms, respectively. Note that a process may not fully utilize its assigned timeslice for a variety of reasons such as blocking for I/O operations.

Chapter 3

Characterization

In this section, we first present the characterization of the context switch effect relating to the priorities of processes, and relating to various cache configurations.

3.1 Impact of Process Priorities

Figure 3.1 shows the breakdown of three types of cache misses of the primary process (context switch, capacity and conflict, and cold) when the priority of both the primary and secondary processes is simultaneously varied. The timeslice duration affected by the priority is shown in the x-axes, while the y-axes shows the number of L2 cache misses normalized to the 4ms case. Each bar shows the number of misses of each primary process, which is an average of number of misses over all cases when the primary benchmark is paired with all other benchmarks. The figure shows that the total number of cache misses is strongly affected by the timeslice duration, in which shorter timeslices yield significantly higher misses than longer timeslices. The traditional cache misses (cold, capacity and conflict) are mostly constant across different timeslice durations, hence the total cache misses is mainly affected by the number of context switch misses. The figure also shows that for a very short timeslice of 4ms, the fraction of cache misses due to context switches is very high (more than 40% for bzip2, crafty, twolf, parser, and vpr on the 4ms case). Even for benchmarks that have very high base miss rates (33.13% mcf and 6.67% art), the fraction of context switch misses also become quite significant (20% for art and 10% for mcf). Overall, the increase in cache misses due to context switches can be computed as the ratio of the context switch misses over all other misses. For the shortest timeslice, this

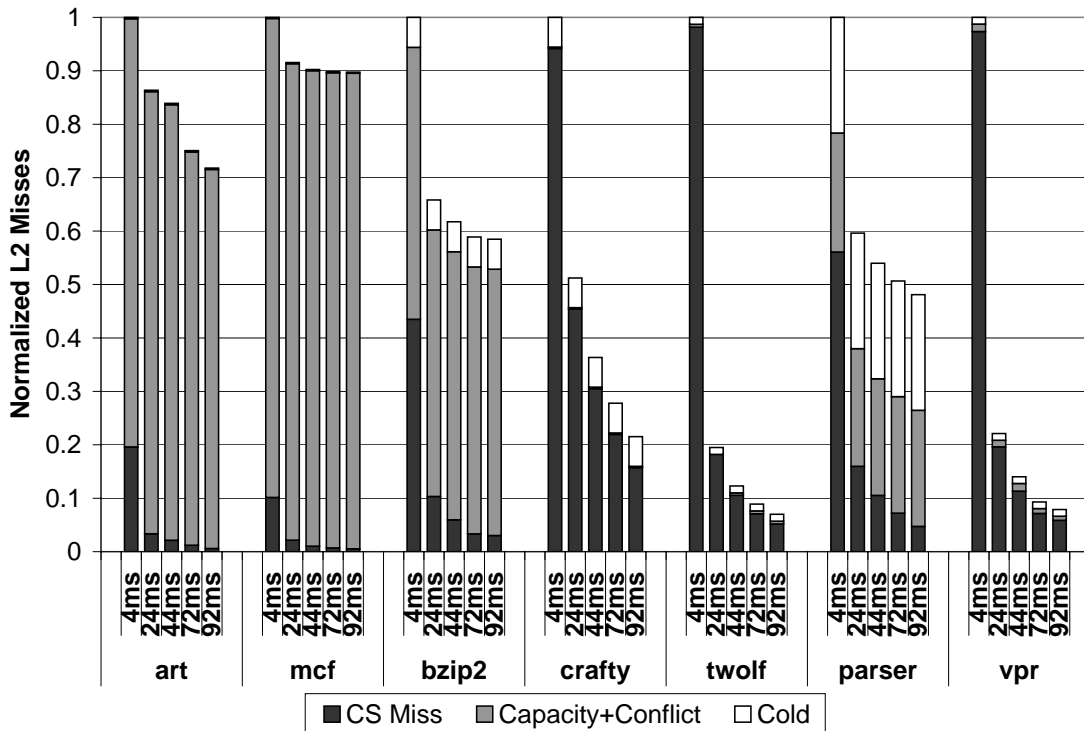
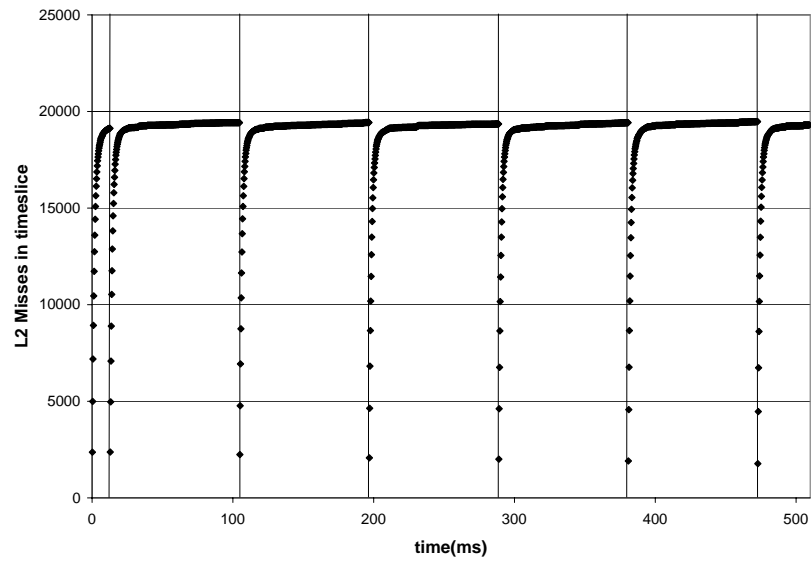


Figure 3.1: Number of L2 cache misses of each primary process broken down into context switch (CS) misses, capacity and conflict misses, and cold misses, for different timeslice durations, normalized to the 4m timeslice duration case.

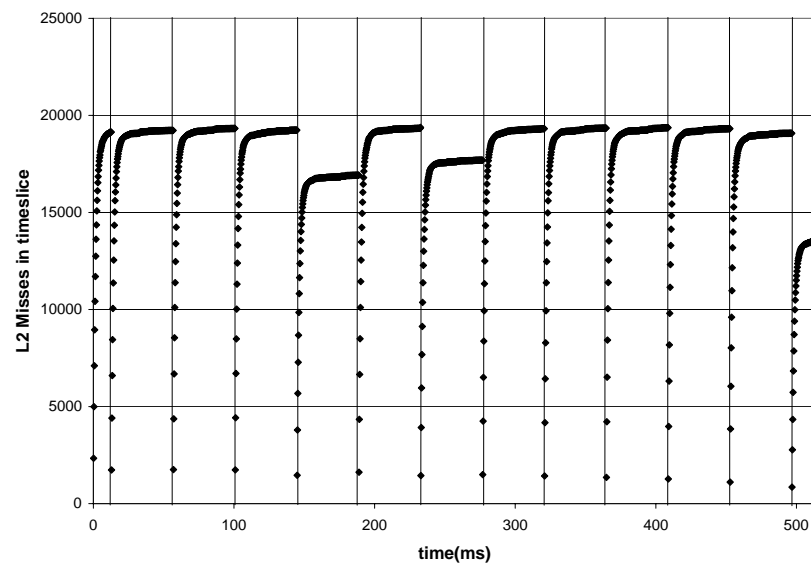
ratio's average is $15.4\times$, and ranges from 1.1 (mcf) to 53.3 (twolf). For the longest timeslice (92ms), the average ratio is much smaller at $1.2\times$.

The number of context switch misses is affected by two opposing factors. The first factor is that a shorter timeslice implies that, to execute the same number of instructions, the primary benchmark executes over more timeslices and suffers from more cache re-warmings. The opposing factor is that because the secondary benchmark also has shorter timeslices, it replaces fewer cache blocks that belong to the primary benchmark, allowing the primary benchmark to have fewer context switch misses when it resumes. Figure 3.1 clearly shows that the latter factor (reduction of context switch misses per timeslice) is much smaller compared to the former factor (increase of number of timeslices).

To illustrate this issue further, we show in Figure 3.2 the cumulative number of L2 cache misses over time (reset to zero at each context switch boundary), for vpr running as the primary benchmark and parser as the secondary benchmark. The figure shows that at each timeslice boundary, vpr would start to suffer from a very high number of misses,



(a)



(b)

Figure 3.2: The cumulative number of L2 misses over time when vpr (primary process) is run with parser (secondary process), with 92ms timeslice (a) and with 44ms timeslice (b). The context switch boundaries are indicated by the vertical lines, and at each boundary the cumulative cache miss counter is reset.

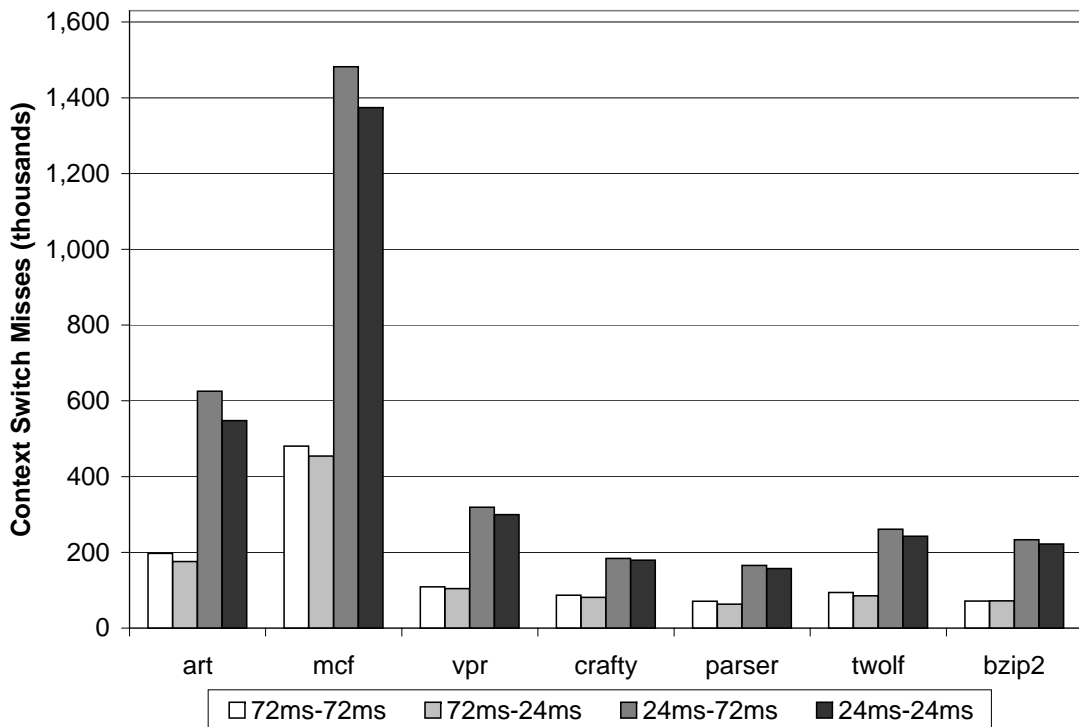


Figure 3.3: The number of context switch misses as a function of the priorities (timeslice durations) of the primary and secondary benchmarks. The timeslice duration is shown as x ms- y ms, where x is for the primary benchmark and y is for the secondary benchmark.

up until the number of misses starts to stabilize. These misses are due to vpr re-warming the cache after each context switch. The re-warming period is roughly 4ms for not only the 92ms timeslice, but also about the same for the shorter 44ms. Hence, the reduction of context switch misses per timeslice due to a shorter timeslice is minor compared to the increase in the number of timeslices it causes.

To examine the effect of asymmetric priorities, Figure 3.3 compares the number of context switch misses affected by the priorities (timeslice durations) of the primary and secondary benchmarks. For each primary benchmark, the number of L2 context switch misses is averaged over all secondary benchmarks. The figure shows that the context switch misses of the primary benchmark is largely determined by its priority as opposed to the priority of the secondary benchmark, as evident in the similar number of misses between 72ms-72ms and 72ms-24ms, as well as between 24ms-72ms and 24ms-24ms. However, the figure also shows that given the same priority for the primary benchmark, lower priority in the secondary benchmark causes slightly fewer context switch misses on the primary

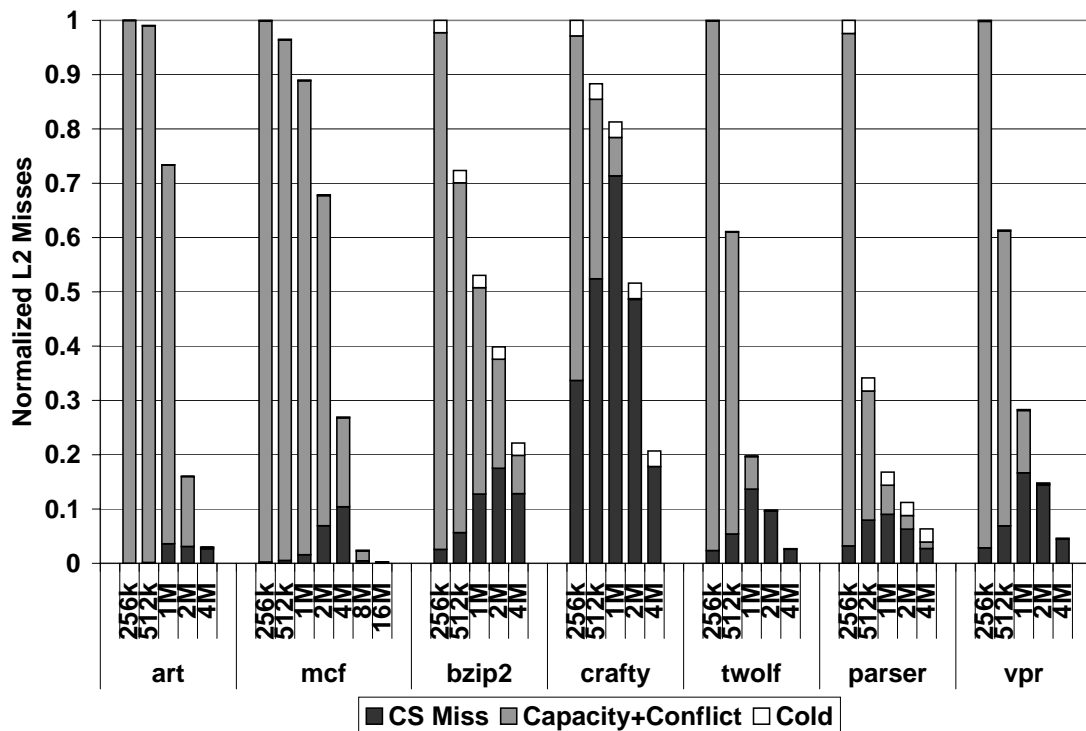


Figure 3.4: Number of L2 cache misses of each primary process broken down into context switch (CS) misses, capacity and conflict misses, and cold misses, for L2 cache sizes ranging from 256KB to 4MB, normalized to the 256KB case.

benchmark, as evident in the 72ms-24ms bars being slightly lower than the 72ms-72ms bars, as well as 24ms-24ms bars being slightly lower than the 24ms-72ms bars. The reason for this is that the secondary benchmark runs for a smaller amount of time each time, and has less chance of displacing the primary benchmark's working set in the cache.

3.2 Sensitivity to Cache Parameters

3.2.1 Cache Size

Figure 3.4 shows the breakdown of three types of cache misses of the primary process (context switch, capacity and conflict, and cold) when the L2 cache sizes are varied from 256KB to 4MB, normalized to the 256KB case. The timeslice duration is 4ms for both the primary and secondary benchmarks. As expected, the total number of L2 cache misses decreases with increasing L2 cache size as more capacity and conflict misses are eliminated.

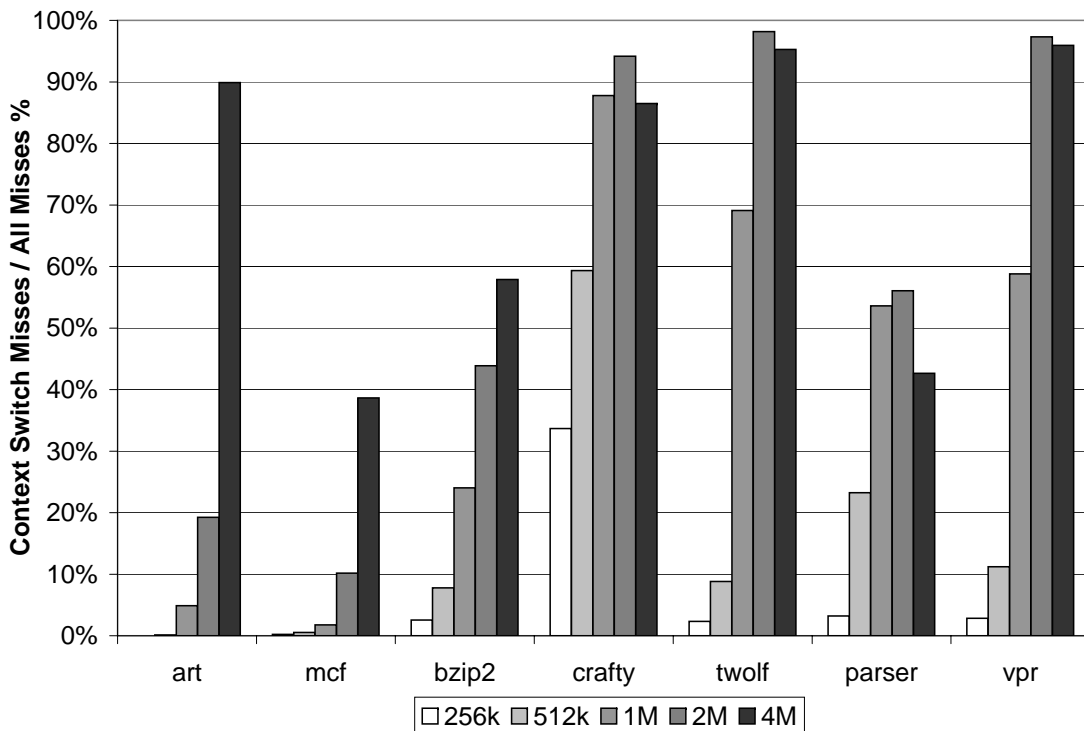


Figure 3.5: The fraction of all L2 cache misses that are context switch misses.

However, the trend of the context switch misses is very different. In absolute term, they increase as the L2 cache size increases, then after a certain cache size, they start to decrease. In relative term (fraction of all misses that are context switch misses), however, they mostly keep on increasing even until the cache size is 4MB (Figure 3.5). This is because as the cache size is increased, the number of cache misses due to capacity and conflict is falling significantly faster than the fall in the number of context switch misses. The exceptions are *crafty*, *twolf*, and *parser*, which at 2MB mostly only have cold misses. Since cold misses do not decrease as the cache size is enlarged further to 4MB, while the number of context switch misses still decrease, their relative fraction also decreases. Overall, the figures show that in large L2 caches, context switch misses become the main source of L2 cache misses.

Next, we hypothesize that the increasing-decreasing trend in the number context switch misses in the absolute term is caused by the interaction of the working set sizes of the primary and secondary benchmarks. With a small L2 cache size, each working set overflows the cache, and since the maximum number of context switch misses per timeslice is the cache size, they are quite small. A medium L2 cache size is large enough to hold the

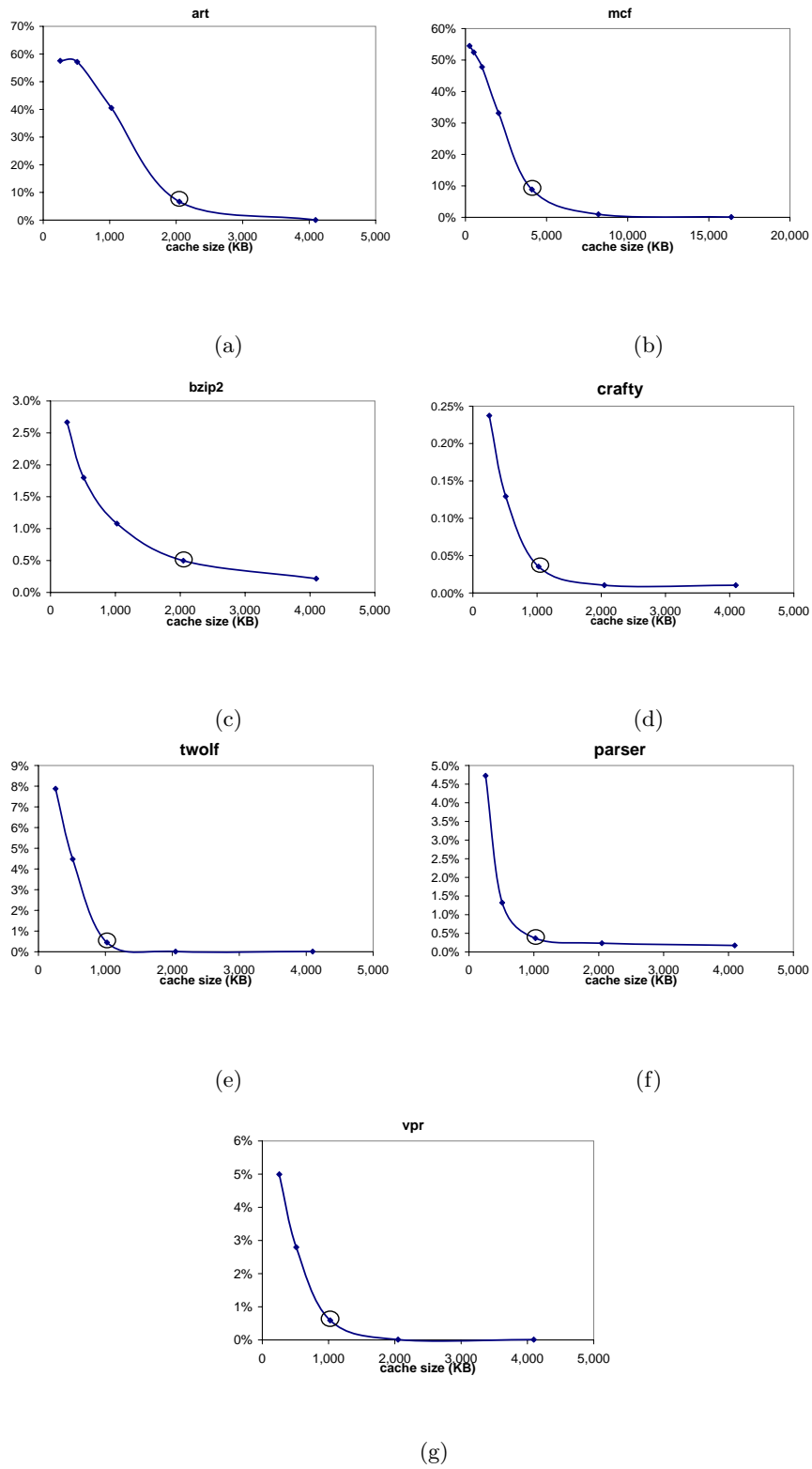


Figure 3.6: Miss Rates for Different Cache Sizes. Each figure also shows the working set size point where the slope of consequent points decreased significantly in circle.

working set size of only one benchmark (either the primary or the secondary). In addition, the maximum number of context switches per timeslice grows with the cache size. This combination creates a situation in which at each context switch, one benchmark replaces and reorders a significant portion of the working set of the other benchmark, causing a large number of context switch misses for the latter benchmark. However, eventually when the L2 cache size is large enough to hold the working set sizes of both benchmarks, the context switch misses start to go down.

To test the hypothesis, we analyze the dominant working set size of each benchmark. To do that, we run each benchmark individually with various cache sizes (256KB through 4MB) in order to plot their miss rate curve. From the curve, we pick up a knee point corresponding to a certain cache size in the curve which shows a dramatic reduction in miss rates, and beyond this point larger caches only reduce the miss rates slightly. Figures 3.6(a) to (g) show how miss rate changes with different cache sizes. Each figure shows miss rate for one benchmark when time-sharing is not employed, and the circled points are knee points. Using method, we estimate the size of the working set of *crafty*, *twolf*, *parser*, and *vpr* as between 512KB-1MB, *bzip2* and *art* as between 1MB-2MB, and *mcf* as between 2MB-4MB. The upperbound of the working set sizes for the benchmarks (1MB for *crafty*, *twolf*, *parser*, and *vpr*; 2MB for *bzip2* and *art*; and 4MB for *mcf*) almost always coincides with the cache sizes in which each benchmark suffers from the most context switch misses in Figure 3.4. This confirms the hypothesis that there is a strong relationship between the working set size of a benchmark and the magnitude of context switch misses that it suffers from, in particular, a benchmark suffers the most from context switch activities when its working set size is close to the cache size.

3.2.2 Cache Associativity

Figure 3.7 shows the breakdown of four types of cache misses of the primary process (reorder context switch, replacement context switch, capacity and conflict, and cold) when the L2 cache associativities are varied from 2-way to 32-way, normalized to the 2-way case. The L2 cache is 2MB, and the timeslice duration is 4ms for both the primary and secondary benchmarks. The impact of associativities on context switch misses does not have a specific trend that is valid for all benchmarks. This is because each benchmark has a different access distribution over the cache sets. On the other hand, reorder context switch misses usually

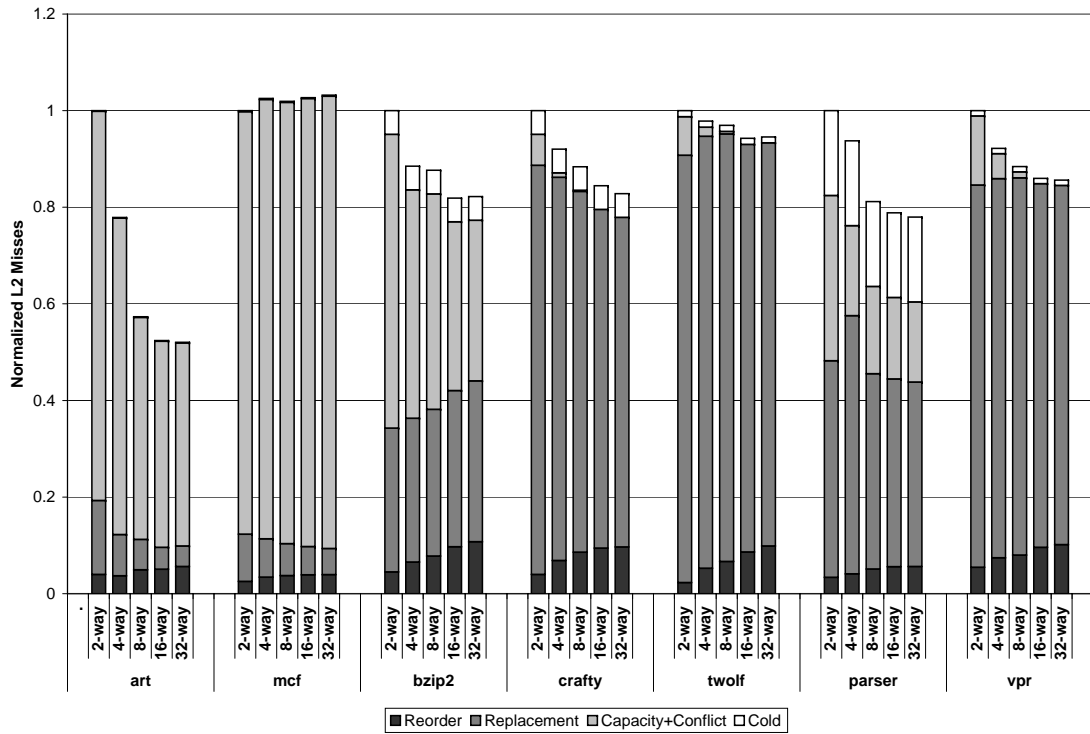


Figure 3.7: Number of L2 cache misses of each primary process broken down into reorder context switch misses, replacement context switch misses, capacity and conflict misses, and cold misses, for L2 cache associativities ranging from 2-way to 32-way, normalized to the 2-way case.

increase with higher associativity. This is because when the cache associativity is small, the interfering process replaces the primary process' lines due to conflicting accesses. But with higher associativity, the interfering process only moves those lines closer to the LRU. In other words, some of the replacement context switch misses become reorder context switch misses with higher associativity.

Chapter 4

Summary and Conclusions

In this study, we characterized the context switch effect based on different scheduling priorities and various cache configurations using a full system simulator and a recent version of Linux Operating System.. We have also defined two types of context switch misses. We observed that a process with lower priority or shorter timeslice suffers significantly more context switch misses than with higher priority due to the need to frequently rewarm the L2 cache after each context switch. Cache interference with another benchmark with lower priority is slightly less damaging compared to high priority. For applications running with low priorities (short timeslice duration), the number of context switch misses increases with larger L2 cache sizes up to a certain L2 cache size, before starting to decrease. We found that this inflection point coincides roughly with the working set size of the application. In relative term (fraction of all L2 cache misses that are context switch misses), however, context switch misses become a more dominant factor of L2 cache misses as the cache size becomes larger. Finally we found that, although L2 cache associativity does not have a particular effect on the total number of context switch misses, it increases the number of reorder context switch misses. This is because the replacement context switch misses caused by interfering process change, with higher associativity, to reordering the line in LRU stack, thus become reorder context switch misses.

Considering that cache sizes are increasing in the future, the total amount of data cached on-chip that may be interfered due to context switches would increase. Hence, techniques that mitigate context switch misses would become increasingly needed.

Bibliography

- [1] Fedora Project. *<http://fedora.redhat.com>*, 2006.
- [2] A. Agarwal, J. Hennessy, and M. Horowitz. Cache performance of operating system and multiprogramming workloads. *ACM Transactions on Computer Systems*, November 1988.
- [3] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel*, pages 258–294. O’Reilly, 2005.
- [4] C. Cascaval and D. A. Padua. Estimating cache misses and locality using stack distances. In *Proc. 174th Annu. Int. Conf. Supercomputing*, pages 150–159, San Francisco, 2003.
- [5] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting the Impact of Inter-Thread Cache Contention on a Chip Multiprocessor Architecture. In *Proc. of the 11th International Symposium on High Performance Computer Architecture*, pages 340–351. IEEE Computer Society, 2005.
- [6] P. Koka and M. H. Lipasti. Opportunities for cache friendly process scheduling. *Workshop on Interaction between Operating System and Computer Architecture*, October 2005.
- [7] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform. *IEEE COMPUTER*, 35(2):50–58, February 2002.
- [8] J. C. Mogul and A. Borg. The effect of context switches on cache performance. *Proceedings of ASPLOS-IV*, April 1991.

- [9] A. J. Smith. Cache memories. *ACM computer surveys*, 14(3):473–530, September 1982.
- [10] W. Stallings. *Operating Systems: Internals and Design Principles*. Prentice Hall, 5th edition, 2005.
- [11] Standard Performance Evaluation Corporation. Spec benchmarks. <http://www.spec.org>, 2000.
- [12] W.A. Wulf and S.A. McKee. Hitting the Memory Wall: Implications of the Obvious. *Computer Architecture News*, 23(1):20–24, March 1995.