High Performance Cache Architectures to Support Dynamic Superscalar Microprocessors

Kenneth M. Wilson
Kunle Olukotun

Technical Report: CSL-TR-95-682

June, 1995

This research has been supported by OTL Research Incentive Award.

High Performance Cache Architectures to Support Dynamic Superscalar Microprocessors

Kenneth M. Wilson
Kunle Olukotun

Technical Report: CSL-TR-95-682

June, 1995

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305-4055
e-mail: pubs@shasta.stanford.edu

Abstract

Simple cache structures are not sufficient to provide the memory bandwidth needed by a dynamic superscalar computer, so more sophisticated memory hierarchies such as non-blocking and pipelined caches are required. To provide direction for the designers of modern high performance microprocessors, we investigate the performance tradeoffs of the combinations of cache size, blocking and non-blocking caches, and pipeline depth of caches within the memory subsystem of a dynamic superscalar processor for integer applications.

The results show that the dynamic superscalar processor can hide about two-thirds of the additional latency of two and three pipelined caches, and that a non-blocking cache is always beneficial. A pipelined cache will only outperform a non-pipelined cache if the miss penalty and miss rates are large.

Key Words and Phrases: pipelined cache, non-blocking cache, memory hierarchies, integer benchmarks, dynamic superscalar

1. Introduction

It is well known that microprocessor speed is improving at a greater rate than memory speed, and the performance of the memory subsystem of a computer architecture is becoming the performance bottleneck of the computer system. Dynamic superscalar computer architectures further exacerbate this problem by supporting speculation and exploiting instruction level parallelism (ILP), both of which require an increase in memory bandwidth. Simple cache structures are not sufficient to provide the increased memory bandwidth, so more sophisticated memory hierarchies such as non-blocking and pipelined caches are required.

For sophisticated memory hierarchies, understanding the change in miss rate due to different memory structures is not sufficient to understand the complex interactions that make up memory performance. To quantify the components of memory performance we will use the following formulas. A well known formula for measuring processor performance [Henn90] is:

$$\text{Execution Time} = \text{CPI} * T_{cpu}$$

where CPI is clocks per instruction, and T_{cpu} is the cycle time of the processor. CPI can be further broken down into processor and memory components.

$$\text{CPI} = \text{CPI}_{processor} + \text{CPI}_{memory}$$

$\text{CPI}_{processor}$ represents the processor pipeline with an ideal single cycle memory, and CPI_{memory} is the additional CPI due to delays in the memory subsystem. For the purposes of this paper, CPI_{memory} will be calculated with the following equation:

$$\text{CPI}_{memory} = \text{CPI}_{hit(x)} + \text{CPI}_{miss} - \text{CPI}_{overlap}$$

$CPI_{hit(x)}$ is the additional CPI due to hit times of x cycles where x is greater than one, and $CPI_{overlap}$ represents the CPI gained from the overlap of misses within the memory subsystem. CPI_{miss} is the CPI due to cache misses and can be calculated as follows:

$$CPI_{miss} = MR_D * \text{miss penalty of L1 cache} + MR_{L2} * \text{miss penalty of L2 cache}$$

where MR_D and MR_{L2} are the global miss rates per one hundred instructions for the level one data cache and the level two cache respectively.

The preceding formulas can be used to describe the effects of pipelining the cache, making a cache non-blocking, and scheduling instructions. A pipelined cache is a cache where the cycle time of the cache is smaller than the latency of the cache, and there can be multiple accesses in the pipeline of the cache. An increase in cache size would normally cause a decrease in CPI_{miss} and an increase in T_{cpu} [Przy88]. With the use of a pipelined cache [Oluk92], $CPI_{hit(x)}$ of the cache increases, but the cache can be made larger without an increase in Tcpu. A pipelined cache is achieved by performing the cache access in stages, and having up to one access in any stage at a time, much like a pipelined processor. The doubling (two pipe stages) or tripling (three pipe stages) of the hit time of the cache allows the cache designer to build a much larger cache, greatly reducing the miss rate of the cache. The reduction of the cache miss rate must be great enough to offset the large increase in the cache hit time to make this cache improvement technique worthwhile.

A non-blocking cache improves performance by increasing $CPI_{overlap}$ without increasing $CPI_{processor}$, $CPI_{hit(x)}$, or CPI_{miss}. A non-blocking cache buffers cache misses, allowing the cache to be accessed even with multiple cache misses pending [Krof81]. To the processor, the overlapping of cache accesses appears as if the miss penalty is reduced. A special case of a non-blocking cache is a blocking cache with hit-under-miss. In a hit-under-miss cache, the first miss to the cache does not block the cache, and subsequent hits to the cache can still be performed. A hit-under-miss cache is easier to implement in hardware and provides the same benefits as a non-blocking cache but to a lesser extent.

Instruction scheduling is required to receive the full performance benefit of pipelined and non-blocking caches. The dynamic and static scheduling of instructions can reduce $CPI_{processor}$ and $CPI_{hit(x)}$ by moving dependencies to hide part of the hit time of the cache [Ghar92]. $CPI_{processor}$ and $CPI_{hit(x)}$ are reduced by rescheduling the use of a load away from the load instruction to overlap the load latency with instruction execution.

Non-blocking techniques can be combined with dynamic instruction scheduling to further reduce CPI by allowing the dynamic processor to hide the latencies of cache misses as well as hits. This can be performed through the use of limited blocking or a non-blocking cache. Limited blocking is where instead of the processor stalling on a cache miss, instructions that are not dependent on the load that missed are still allowed to execute, but the cache is stalled until the miss completes [Conte92]. A non-blocking cache improves processor performance by adding the overlapping of cache misses to the ability of overlapping execution with cache misses provided by limited blocking.

A summary of the effects of each individual enhancement technique discussed above is shown in Table 1. Note that the entry for dynamic scheduling in Table 1 assumes limited blocking. All of the possible combinations of the memory performance improvement techniques in Table 1 are well understood for floating point applications. While the memory performance of floating point applications has been well explored, integer applications have not received the same attention. To provide direction for the designers of general purpose high performance microprocessors, we investigate the performance tradeoffs for combinations of cache size, blocking and non-blocking caches, and pipeline depth of caches within the memory subsystem of a dynamic superscalar processor for integer applications.

Technique	CPI processor	CPI miss	CPI overlap	CPI Hit	Cycle Time
Increase Cache Size	0	-	-	0	+
Dynamic/Static Scheduling	-	0	0	-	+
Non-Blocking Cache	0	0	+	0	0
Pipelined Cache	0	0	0	+	-

Key	
+:	increase
-:	decrease
0:	no effect

Table 1: Individual effects of memory performance improvement techniques.

The conventional wisdom is that, 1) if a cache is non-blocking, the overlapping of miss penalties will be great enough that the processor will not benefit from the reduced miss rate of a pipelined cache, and 2) that pipelining may actually decrease processor performance due to the increase in hit time of the pipelined cache. This paper will show that a dynamic superscalar processor can hide most of the increased hit time of the pipelined cache, and that for a dynamic superscalar processor, pipelining the cache can still be useful even when the cache is non-blocking.

The rest of this paper is organized as follows: The computer architecture used in this investigation is described in section 2, and the experimental methodology used to simulate the architecture described in section 2 is discussed in section 3. Section 4 contains the results gathered to characterize the design space by simulating the interactions between data cache size, blocking and non-blocking caches, and the pipeline depth of the cache. In section 5 we discuss how to use miss rates to relate the results of section 4 to large integer applications, and section 6 presents the conclusions drawn from the experimental results and the preceding discussion.

2. Architectural Assumptions

This section describes the computer architectures that are simulated in this paper. A dynamic superscalar processor is used to give the benefits of instruction scheduling and to take advantage of the high memory bandwidth that is being provided by the memory system. Since the memory system is the focus of this study, we describe its datapath in detail.

2.1. Processor

Figure 1 shows the dynamically scheduled superscalar processor used in this study, and Figure 2 shows the processor's pipeline diagram. The processor uses the Tomasulo algorithm [Toma67] for the dynamic instruction scheduling, and a branch target buffer [Lee84] and reorder buffer [Smith85] for branch misprediction and recovery. The processor can issue and retire up to two MIPS instructions every cycle. All busses internal to the processor are 32 bits wide, and all busses involving memory are 64 bits wide. The branch target buffer is four-way-set-associative with 4096 entries and uses two bit saturating counters for branch prediction [Smith81]. The reorder buffer contains 32 entries and can retire up to two instructions per cycle to the register file.

To reduce the enormous number of simulation runs, parts of the design space have been fixed. Both first and second level caches have a line size of 32 bytes, and the first level data cache is always direct mapped to provide the lowest access time [Wilt94]. The organization of the instruction cache is fixed at 8KB and four-way set associativity. Since this study focuses on data references, the instruction cache was designed to be aggressive enough to provide a low instruction miss rate to isolate the performance effects of data references. The size of the off-chip secondary cache is fixed at 1MB.

2.2. Memory Subsystem

The memory subsystem is made up of the load/store unit, level one instruction cache, level one data cache, level two cache, and the pipelined memory as shown in Figure 3. The load/store unit is further divided into six reservation stations, the level one data cache controller, and the miss unit. The load/store unit handles all data memory accesses and contains a ten entry store buffer [Toma67].

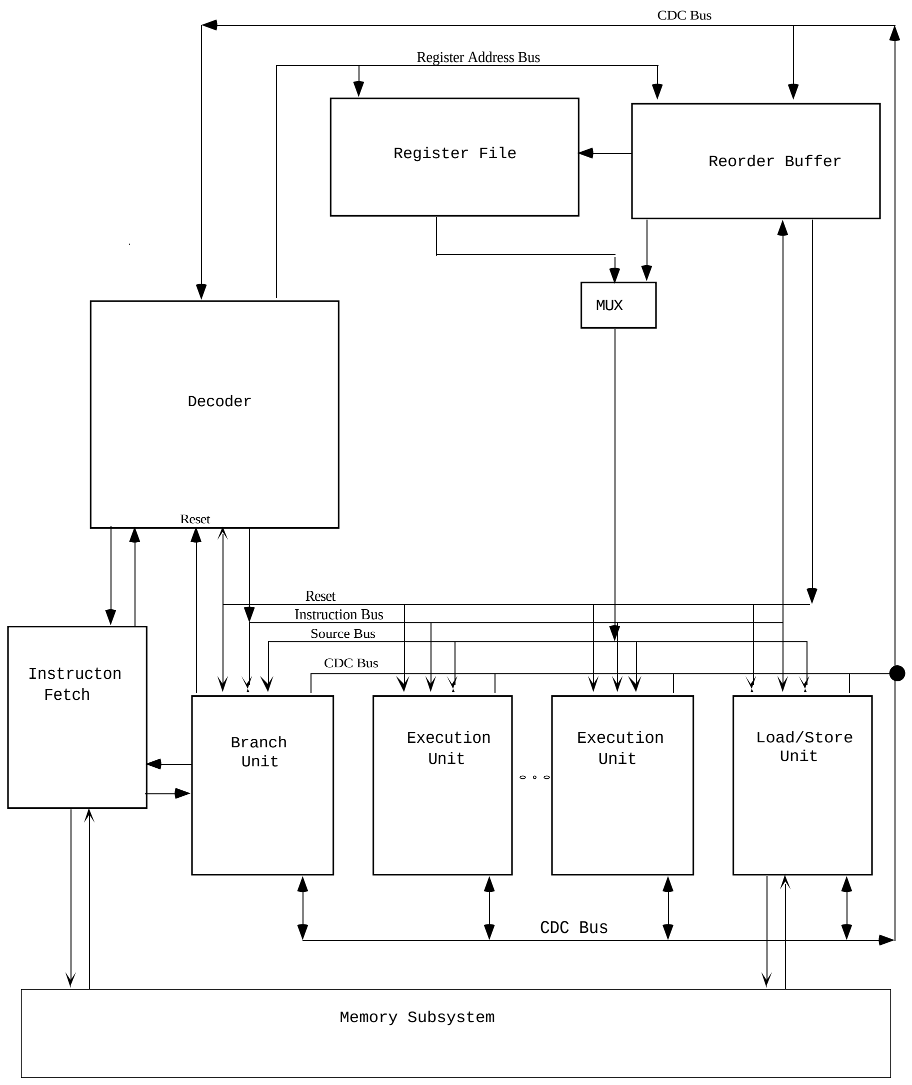

Figure 1: The dynamic superscalar processor.

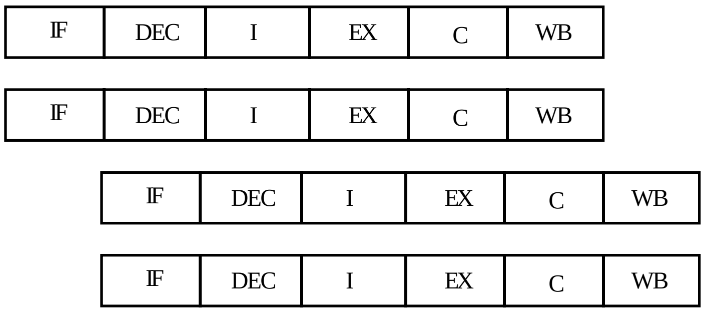

Figure 2: The dynamic superscalar processor instruction pipeline.

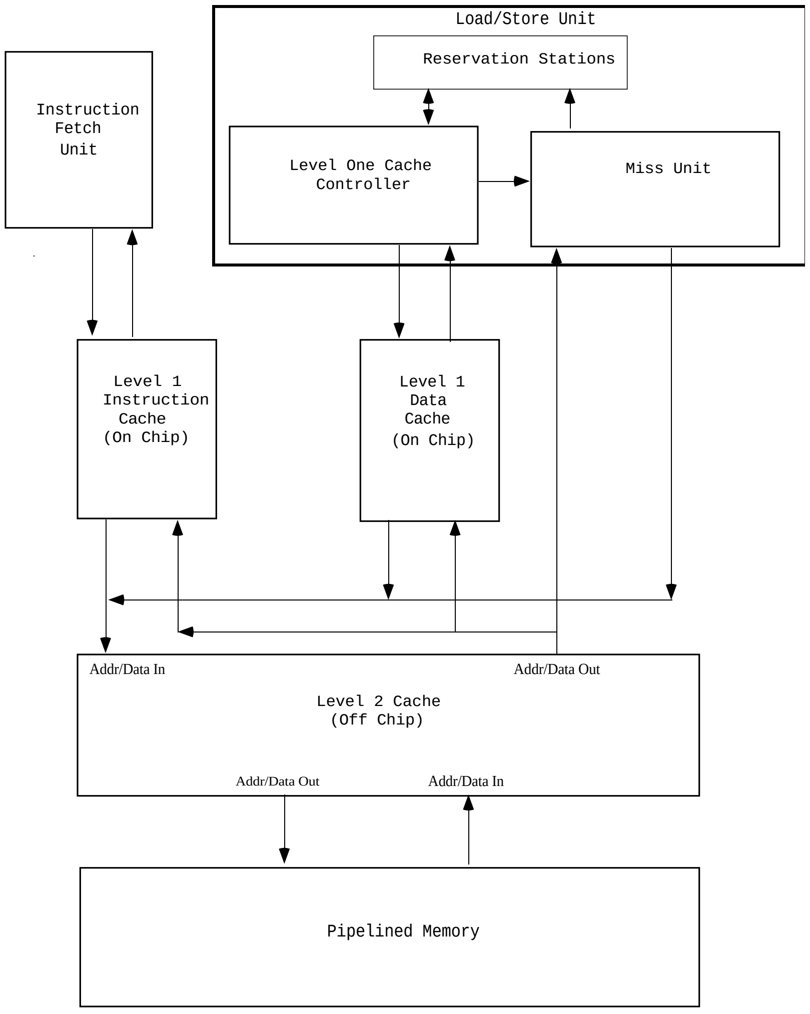

Figure 3: The memory subsystem.

Inside the load/store unit, the level one data cache controller issues all new loads and stores to the level one data cache. If the level one data cache has the data, or the access is a write, the cache accepts the access. Since the level one data cache is write-through with no allocation on a write miss, all writes are immediately sent to the level two cache, independent of whether the write was a hit or a miss. The miss unit snoops on all accesses to the level one data cache, and controls access to the second level cache on a miss. When the level two cache is not busy, the miss unit will send a read cache line command to the level two cache. After the miss penalty has elapsed, the level two cache writes the requested line to the level one data cache at the rate of eight bytes per cycle. The miss unit will be snooping the level two cache for this write, looking for data to satisfy the misses buffered in the Miss Status Handling Registers (MSHR) [Krof81]. If the required data is present, the miss unit forwards the data to the correct reservation station so that the load data can be placed on the completion bus.

The miss unit implements the non-blocking cache behavior [Chen92, Fark94], and is implemented as a modified inverted MSHR [Fark94]. The inverted MSHR scheme has been modified in two ways. First, there are only six MSHRs instead of one for every register in the processor. The use of a smaller number of MSHRs is possible because the load/store unit can have at most as many misses as there are reservation stations in the load/store unit. Note that having one MSHR per reservation station allows multiple misses to the same destination register, which was not allowed in the original inverted MSHR scheme. Second, instead of each MSHR being dedicated to a particular reservation station, each MSHR stores the number of the reservation station that contains the miss. This allows the misses to be kept in FIFO order so that if multiple misses are resolved in a single cycle, they can be retired in the order they entered the load/store unit. One example of where multiple misses are satisfied in a single cycle occurs when four load bytes from the same word in memory are stored in the MSHRs. The inverted MSHR organization described in [Fark94] did not limit the number of misses that could be completed in a single cycle, so all four of the load byte instructions would complete in the same cycle, whereas this processor can complete at most two load/store instructions per cycle.

The level two cache uses a copy-back, write-allocate, no fetch on write miss policy. The main memory pipelines all accesses from the level two cache. The main memory is also aware of any duplicate accesses to the same memory address and combines accesses to the same address. For all of the investigations in this paper, the memory delay is set at a 20 cycle memory access time and a 5 cycle memory cycle time.

3. Simulation Methodology

This section describes how we simulated the computer architecture described in the previous section. The model is described at the functional block level, and is written in the Verilog Hardware Description Language (Verilog HDL). The model accurately captures the details of the entire dynamic superscalar architecture including caches and main memory.

The use of Verilog to describe the computer architecture under simulation provides increased realism over general purpose programming languages by forcing the explicit specification of the parallel pipeline behavior of the processor and memory subsystem. The detailed specification of the processor pipeline structures in the model gives an idea of the design complexity of the hardware, and insures that all corner cases are implemented. The implementation of all corner cases is required to obtain an accurate measure of program execution time in terms of CPU clock cycles. Another advantage of Verilog is that there exist logic synthesis tools that can be used to turn the model into a gate level description. The drawback of these tools is that they take a great deal of time and effort to synthesize hardware from a large model. Currently under development are tools that can quickly and efficiently extract estimates of cycle time and silicon area from this model.

3.1. Benchmarks

The performance of four of the SPEC92 integer benchmarks are used to compare the different memory architectures used in this study. No floating point benchmarks were used, since floating point programs are usually technical applications that exhibit a great deal of parallelism. The increased parallelism can be scheduled by a compiler to hide the latency of load misses. The instruction level parallelism of the floating point SPEC92 benchmarks is so great that MIPS TFP is able to execute the floating point data directly out of the secondary cache while achieving one of the best floating point performances in the industry [Gwen93]. If TFP can store the floating point data directly in the secondary cache with a hit time of five cycles, then it is easily possible to compile for the two or three cycle hit time of a pipelined cache.

The integer SPEC benchmarks used in this study are:

1) Compress: Compresses a file using adaptive Lempel-Ziv coding
2) Eqntott: Translates a boolean equation to a truth table
3) Espresso: Minimizes boolean functions
4) gcc: compiler based on the GNU C compiler version 1.35

3.2. Simulation Speed

The fastest commercially available Verilog simulators simulate about 50 instructions per second on our Verilog model. This is too slow to execute the SPEC92 benchmarks to completion, so smaller input sets have been used than are recommended by the designers of the benchmark suite. The smaller input sets decrease the execution time of the benchmarks enough to make collecting data for the different memory configurations under simulation possible within a reasonable period of time.

The problem with using smaller input sets when executing the SPEC92 benchmarks is that this may change the results. If the input set is too small, then the percentage of time spent outside the frequently executed code of the benchmark may begin to be as large or larger than the time spent inside the frequently executed code. The benchmark with the smaller input set has to execute long enough to amortize the effects of initialization code, compulsory misses, I/O, etc., over the execution time of the benchmark. To prove that the benchmarks executed with the smaller input sets show the same execution patterns as when larger input sets are used, data characterizing compulsory misses and miss rates were sampled every ten thousand instructions. For all of the benchmarks, input sets were chosen that produced the same pattern of memory system usage over time as the larger input sets.

Another problem is the smaller data working set size. We will need to compare the results of the simulated SPEC benchmarks to applications that have larger data working sets. The solution of using smaller cache sizes was used by Gupta [Gupta91], and will be used in this study as well. Since the working sets of the benchmarks are smaller, smaller caches must be used to observe the higher miss rates expected of an application with a larger data working set. The miss rates of the smaller caches can then be used to map the SPEC benchmark performance to larger integer applications.

4. Results

To characterize the design space of this study, combinations of blocking, non-blocking, and pipelined first level data caches were simulated for cache sizes varying from 2KB to 64KB. The miss penalty of the first level data cache was also simulated at six and twelve cycles to show the effect of changes in miss penalty. The data from these simulations allows us to answer a number of key questions. First, how much improvement in performance is gained by the use of a non-blocking cache instead of a blocking cache. Second, how much of the additional hit time produced by pipelining the cache can be hidden by the dynamic superscalar processor. Lastly, after understanding the performance effects of the individual components, we study how the combinations of the three mechanisms of non-blocking, pipelining, and varying cache size affect performance.

4.1. Blocking vs. Non-Blocking Caches

Table 2 shows the CPI for a blocking cache and a non-blocking cache for each of the benchmarks. To favor the blocking cache, a 32KB first level data cache was used to produce a low miss rate, and a very small miss penalty of three cycles was used in the simulation. Note that a 32KB on-chip first level data cache is as large as any currently available commercial processor and is large enough to hold a sizable fraction of the working set of our benchmarks. The miss penalty of three cycles is lower than can be found on any modern processor for an off-chip level two cache. Even with the miss rate and miss penalty set to favor the blocking cache, the non-blocking cache outperformed the blocking cache for all of the benchmarks, achieving an average speedup of 8%.

Conte's [Conte92] study of non-blocking caches found speedups of between 1.11 (for a ten cycle miss penalty and 10% miss rate) and 1.51 (for a twenty cycle miss penalty and 10% miss rate) when using a non-blocking cache in a two issue dynamic superscalar processor. Based upon Conte's and our results we believe that a non-blocking cache is desirable for our two issue dynamic superscalar processor for three reasons. First, a non-blocking cache increased performance for every simulation produced in this study. Second, the additional complexity of adding a non-blocking cache to a dynamic superscalar processor is small. Third, the incremental cost of adding non-blocking was very small since only six MSHRs were needed and the presence of the miss unit inside the load/store unit reduced the complexity of the level one data cache. One of the advantages of using a Verilog model in this study is that the design is specified in enough detail to give us a good idea of the complexity and design cost associated with architectural options such as non-blocking caches. Due to the low incremental cost and higher memory performance of non-blocking caches, even when using an extremely low miss rate and low miss penalty, the rest of the studies in this paper assume a non-blocking cache.

Benchmark	Read Misses Per 100 Instructions	Blocking CPI	Non-Blocking CPI	Non-Blocking Speedup
compress	1.42	1.32	1.12	1.18
eqntott	0.33	1.26	1.17	1.08
espresso	0.19	1.34	1.33	1.01
gcc	0.71	1.76	1.69	1.04

Table 2: Table comparing the performance of blocking vs. non-blocking caches on the dynamic superscalar processor with a cache size of 32KB, a one cycle hit time, and a miss penalty of three cycles. Note that miss rate is the number of data cache misses per 100 instructions whereas Conte defined miss rate as the percent of loads that missed in the data cache.

4.2. Hiding Latency with Dynamic Instruction Scheduling

A dynamic superscalar processor is able to hide some of the one or two cycle additional hit time required by a pipelined cache. In order to quantify the percentage of the additional hit time that can be hidden, a first level data cache with very low miss penalties and very low miss rates was simulated. With the low miss penalties and miss rates, the performance improvement due to pipelining the cache is negligible, but the increase in hit time is not. Thus the increase in hit time that is not hidden by the dynamic superscalar processor results in an increase in $CPI_{hit(x)}$. Table 3 shows the CPI results for cache pipeline depths of one, two, and three stages with a fixed cache size of 32KB.

Benchmark	Percentage Loads	1 deep pipeline CPI	2 deep pipeline CPI	3 deep pipeline CPI	Latency Hidden 1->2	Latency Hidden 2->3
compress	22%	1.12	1.20	1.25	64%	77%
eqntott	23%	1.17	1.24	1.34	70%	57%
espresso	22%	1.33	1.41	1.49	64%	64%
gcc	21%	1.69	1.75	1.81	71%	71%

Table 3: CPI results for different levels of pipelining with a 32KB non-blocking data cache and a three cycle miss penalty.

The percentage of the additional hit time due to a pipelined cache that can be hidden by dynamic scheduling is shown in Table 3. If the processor is unable to hide any of the additional hit time of the pipelined cache, the CPI of the processor will increase by the frequency of loads executed by the processor for each cycle of additional hit time. For example, compress has a load frequency of 22%. The non-pipelined cache has a CPI of 1.12 for the compress benchmark. If the processor cannot hide any of the additional latency, the CPI observed for the two deep pipelined cache should be 1.12 + 0.22 = 1.34. Instead the CPI of the dual pipelined cache is 1.20, so the processor hides 0.14 CPI or 64% of the additional latency.

The dynamic superscalar processor hides about two thirds of the additional hit time; therefore the increased size of the cache allowed due to pipelining only needs to make up for the remaining one third of the additional hit time to break even. The results for the blocking caches are similar, as shown in Table 4. Roughly two thirds of the additional hit time is hidden by the dynamic superscalar processor for cache pipelines of two and three deep.

Benchmark	Percentage Loads	1 deep pipeline CPI	2 deep pipeline CPI	3 deep pipeline CPI	Latency Hidden 1->2	Latency Hidden 2->3
compress	22%	1.32	1.39	1.45	68%	73%
eqntott	23%	1.26	1.33	1.43	70%	57%
espresso	22%	1.34	1.42	1.50	64%	64%
gcc	21%	1.76	1.81	1.87	76%	71%

Table 4: CPI results for different levels of pipelining with a 32KB blocking data cache and a three cycle miss penalty.

Notice that for the compress benchmark, more latency is hidden with an increase of hit time from two to three cycles than with an increase of one to two cycles. This is due to the level one instruction and data cache sharing the level two cache's "Addr/Data Out" bus (see Figure 3). Since the caches have a thirty-two byte line and the data bus is only sixty-four bits, it takes four cycles to transfer an entire cache line from the second level cache to a first level cache. If the level two cache had a read access by the instruction cache one cycle, and a read access from the data cache the next cycle, then the instruction cache will see the expected miss penalty, but the data cache will see the miss penalty plus three cycles. This bus conflict does not occur for the 32KB non-pipelined cache, occurs often for the two-pipelined cache, and occurs less often for the three-pipelined cache. This effect increases the CPI of the processor with a two-pipelined cache more than with a three-pipelined cache, so the increase in CPI with pipeline depth does not progress as expected. Only in the compress benchmark does this bus conflict occur often enough to cause an appreciable effect on the CPI results.

cache size	blocking	no pipelining	two pipelined	three pipelined	CPI miss	CPI processor	CPI hit(2)	CPI hit(3)	CPI overlap
2K	1.53	1.23	1.29	1.33	0.61	0.92	0.06	0.10	0.30
4K	1.40	1.16	1.23	1.28	0.48	0.92	0.07	0.12	0.24
8K	1.37	1.15	1.22	1.27	0.45	0.92	0.07	0.12	0.22
16K	1.35	1.12	1.21	1.26	0.43	0.92	0.09	0.14	0.23
32K	1.33	1.13	1.20	1.25	0.41	0.92	0.07	0.12	0.20
64K	1.35	1.12	1.20	1.25	0.43	0.92	0.08	0.13	0.23

a) Results for compress with a six cycle miss penalty.

cache size	blocking	no pipelining	two pipelined	three pipelined	CPI miss	CPI processor	CPI hit(2)	CPI hit(3)	CPI overlap
2K	1.66	1.31	1.37	1.40	0.74	0.92	0.06	0.09	0.35
4K	1.43	1.17	1.24	1.29	0.51	0.92	0.07	0.12	0.26
8K	1.39	1.15	1.22	1.27	0.47	0.92	0.07	0.12	0.24
16K	1.35	1.12	1.20	1.25	0.43	0.92	0.08	0.13	0.23
32K	1.31	1.11	1.18	1.23	0.39	0.92	0.07	0.12	0.20
64K	1.29	1.10	1.18	1.23	0.37	0.92	0.08	0.13	0.19

b) Results for compress with a twelve cycle miss penalty.

cache size	blocking	no pipelining	two pipelined	three pipelined	CPI miss	CPI processor	CPI hit(2)	CPI hit(3)	CPI overlap
2K	1.49	1.33	1.39	1.43	0.41	1.08	0.06	0.11	0.16
4K	1.30	1.20	1.27	1.35	0.22	1.08	0.07	0.15	0.10
8K	1.27	1.17	1.25	1.34	0.19	1.08	0.08	0.17	0.10
16K	1.27	1.17	1.25	1.33	0.19	1.08	0.07	0.16	0.10
32K	1.27	1.17	1.24	1.33	0.19	1.08	0.07	0.16	0.10
64K	1.26	1.17	1.25	1.33	0.18	1.08	0.08	0.17	0.09

c) Results for eqntott with a six cycle miss penalty.

cache size	blocking	no pipelining	two pipelined	three pipelined	CPI miss	CPI processor	CPI hit(2)	CPI hit(3)	CPI overlap
2K	1.66	1.44	1.53	1.59	0.58	1.08	0.09	0.15	0.22
4K	1.34	1.23	1.30	1.37	0.26	1.08	0.07	0.15	0.11
8K	1.28	1.18	1.26	1.35	0.20	1.08	0.08	0.16	0.10
16K	1.28	1.18	1.26	1.34	0.20	1.08	0.08	0.17	0.10
32K	1.27	1.17	1.25	1.34	0.19	1.08	0.08	0.17	0.10
64K	1.26	1.16	1.25	1.33	0.18	1.08	0.08	0.16	0.10

d) Results for eqntott with a twelve cycle miss penalty.

cache size	blocking	no pipelining	two pipelined	three pipelined	CPI miss	CPI processor	CPI hit(2)	CPI hit(3)	CPI overlap
2K	1.53	1.48	1.56	1.62	0.37	1.16	0.07	0.14	0.05
4K	1.46	1.43	1.51	1.57	0.30	1.16	0.08	0.14	0.03
8K	1.42	1.40	1.48	1.56	0.26	1.16	0.08	0.15	0.02
16K	1.40	1.39	1.47	1.55	0.24	1.16	0.08	0.16	0.01
32K	1.41	1.39	1.47	1.55	0.25	1.16	0.08	0.16	0.02
64K	1.4	1.39	1.46	1.54	0.24	1.16	0.08	0.16	0.01

e) Results for espresso with a six cycle miss penalty.

cache size	blocking	no pipelining	two pipelined	three pipelined	CPI miss	CPI processor	CPI hit(2)	CPI hit(3)	CPI overlap
2K	1.74	1.66	1.74		0.58	1.16	0.07		0.08
4K	1.62	1.58	1.66	1.72	0.46	1.16	0.08	0.14	0.04
8K	1.55	1.54	1.61	1.69	0.39	1.16	0.08	0.15	0.01
16K	1.53	1.51	1.59	1.66	0.37	1.16	0.08	0.15	0.02
32K	1.52	1.51	1.59	1.66	0.36	1.16	0.08	0.15	0.01
64K	1.52	1.51	1.58	1.66	0.36	1.16	0.07	0.15	0.01

f) Results for espresso with a twelve cycle miss penalty.

cache size	blocking	no pipelining	two pipelined	three pipelined	CPI miss	CPI processor	CPI hit(2)	CPI hit(3)	CPI overlap
2K	1.95	1.87	1.91		0.81	1.14	0.04		0.08
4K	1.90	1.83	1.87	1.93	0.76	1.14	0.04	0.09	0.07
8K	1.85	1.80	1.84	1.90	0.71	1.14	0.05	0.11	0.05
16K	1.83	1.78	1.83	1.89	0.69	1.14	0.05	0.11	0.05
32K	1.83	1.78	1.83	1.89	0.69	1.14	0.05	0.11	0.05
64K	1.82	1.77	1.83	1.89	0.68	1.14	0.05	0.11	0.05

g) Results for gcc with a six cycle miss penalty.

cache size	blocking	no pipelining	two pipelined	three pipelined	CPI miss	CPI processor	CPI hit(2)	CPI hit(3)	CPI overlap
2K	2.16	2.05	2.09	2.14	1.02	1.14	0.04	0.09	0.11
4K	2.07	1.98	2.03	2.08	0.93	1.14	0.05	0.10	0.09
8K	1.99	1.93	1.98	2.04	0.85	1.14	0.04	0.10	0.06
16K	1.96	1.91	1.96	2.02	0.82	1.14	0.05	0.11	0.05
32K	1.95	1.90	1.95	2.01	0.81	1.14	0.05	0.11	0.05
64K	1.95	1.90	1.95	2.01	0.81	1.14	0.05	0.11	0.05

h) Results for gcc with a twelve cycle miss penalty.

Table 5: Simulation results for all four integer benchmarks with a six and twelve cycle miss penalty. The results in the columns blocking, no pipelining, two pipelined, and three pipelined are the CPI from their respective simulations. CPI miss is CPI_{miss} from the equations at the beginning of this paper.

4.3. Pipelined Caches

Table 5 shows the results of executing the benchmarks with cache sizes ranging from 2KB to 64KB, and level two caches with hit times of six and twelve CPU cycles. CPI_{memory} was calculated by subtracting $CPI_{processor}$ from the CPI of the processor simulated with a blocking cache. $CPI_{overlap}$ represents the decrease in CPI due to using a non-blocking cache to overlap read misses, and is calculated by subtracting the CPI results of the processor simulated with a non-pipelined cache from the CPIs for the processor with a blocking cache. Remember that all of the simulations with pipelined caches tabulated in Table 5 used a non-blocking cache, so the results from the non-pipelined cache are also the results from a non-blocking cache. Subtracting the CPI of a non-pipelined cache from the CPI of the two-pipelined (three-pipelined) cache was used to calculate $CPI_{hit(2)}$ ($CPI_{hit(3)}$).

For compress, the six cycle and twelve cycle CPI results cross with increasing cache size. This is due to the same bus conflict effect as was described for compress in section 4.2. As the cache size increases, the bus conflict has a greater effect on performance because with a larger cache a larger portion of the working set is contained in the cache, causing more reads that miss in the level one data cache to also miss in the level two cache. Since a 64KB cache contains the entire working set for the compress benchmark, the CPIs for both six and twelve cycle miss penalties are expected to be equal, but since the six cycle results are slowed down by bus conflicts, the twelve cycle results are slightly better.

By analyzing the data in Table 5 we can understand when a pipelined cache is preferable over a non-pipelined cache. Notice that the larger the miss penalty and the larger the miss rate of the first level data cache, the better the pipelined cache will perform when compared to the non-pipelined cache. A pipelined cache will be desirable if the pipelining allows a large enough increase in cache size to increase the pipelined cache's performance beyond the performance of the non-pipelined cache. For the compress benchmark, the data in Table 5 shows that a pipelined cache of two stages has the potential to outperform a non-pipelined cache if the single cycle data cache size is less than 2.6KB.

For the processor with the pipelined cache to outperform the same processor with a non-pipelined cache, the two stage pipelined processor needs to allow an increase in cache size of eight to sixteen times that of the non-pipelined data cache (2.6KB to 32KB in the case of compress). The cache cycle time evaluator, cacti [Wilt94], was used to measure the cycle time in nanoseconds of different direct mapped cache sizes for a 0.5 micron CMOS process. The results from cacti clearly showed that a sixteen fold increase in the size of the cache is possible with a doubling of hit time.

Due to the large increase in cache size with each additional pipeline stage, we only consider two or three cycle first level data cache pipelines. If pipelining allows a sixteen fold increase in cache size as the cacti results suggest, and the non-pipelined first level data cache can be as large as 32KB (processors today have on-chip first level caches of 32KB), then a two-pipelined cache could be as large as 512KB, and a three-pipelined cache could be as large as 4MB. With today's technology it is unlikely that it would be possible to provide an on-chip first level data cache as large as a three-pipelined cache would allow, although multichip module (MCM) technology might provide the potential for a cache of this size [Oluk92].

Cache Size (KB)	Compress	Eqntott	Espresso	Gcc
2	4.46	3.62	2.29	3.13
4	2.57	0.97	1.20	2.09
8	2.20	0.45	0.50	1.24
16	1.81	0.39	0.20	0.86
32	1.42	0.33	0.19	0.71
64	1.28	0.22	0.06	0.64

Table 6: Read misses per 100 instructions for cache sizes of 2KB to 64KB.

6 cycle miss penalty		**12 cycle miss penalty**	
	Miss Rate		Miss Rate
compress	3.6	compress	2.7
eqntott	1.7	eqntott	1.1
espresso	1.8	espresso	1.2
gcc	2.1	gcc	1.5
average:	2.3	average:	1.6

Table 7: For each benchmark, the miss rate per 100 instructions that gives equal performance for a two deep pipelined cache and a non-pipelined cache is entered in the table. If the miss rate per 100 instruction is greater than the miss rate in the table, then a pipelined cache would be preferred over a non-pipelined cache.

Table 6 contains the read miss rates for all four of the benchmarks for cache sizes ranging from 2KB to 64KB. The data in Tables 5 and 6 can be used to find the miss rate at which the pipelined cache becomes desirable over a non-pipelined cache for each of the benchmarks. It was found that compress with a level one data cache miss penalty of six cycles caused the pipelined cache to outperform the non-pipelined cache if the non-pipelined cache was sized at less then 2.6KB. From Table 6, we find that compress' 2.6KB non-pipelined cache has a miss rate per one hundred instructions of 3.6 (as seen in Table 7). If a processor under design is expected to execute applications with the same memory access patterns as compress, then a two-pipelined cache should be considered for this processor if simulations show that such applications have a read miss rate per one hundred instructions of greater than 3.6.

5. Discussion

SPEC92 integer benchmarks have small data working set sizes when compared to real integer applications. To collect the miss rates expected of real integer applications, smaller caches have to be simulated with the SPEC92 integer benchmarks. This study further magnifies this problem by using smaller input data sets to the SPEC92 integer benchmarks, requiring even smaller caches to produce the same miss rates. To make up for the small working set sizes of the SPEC92 integer benchmarks, the miss rates captured with small caches in our simulations, are used to relate our results for non-blocking and pipelined caches to more realistic cache sizes and benchmarks.

The results of the previous section can be used to show how a two-pipelined cache will perform on multi-user commercial benchmarks. This paper contains the miss rates at which a two-stage pipelined cache can outperform a non-pipelined cache for the SPEC92 integer benchmarks. We also have formulas and data for determining how the CPI will change when a two-pipelined cache is used instead of a non-pipelined cache for our dynamic superscalar processor. The problem is that due to small working set sizes, the miss rates for the SPEC92 integer benchmarks are much smaller than would be found in large integer applications for the same cache size. To relate our findings to large integer applications with the same dynamic instruction patterns as our SPEC92 integer benchmarks, we will use the miss rates from the executions of the SPEC integer benchmarks to predict the behavior of large integer applications with equivalent miss rates and realistic cache sizes executing on a dynamic superscalar processor with a two-pipelined cache.

The integer SPEC92 benchmarks simulated in this study are real applications that have the same branch characteristics and load frequency as larger integer applications, so we expect the same sort of dynamic scheduling behavior. The percentage of the instruction stream sent to each execution unit for compress, eqntott, espresso, and gcc are shown in Table 8. Table 9 has the branch frequencies for the multi-user commercial workloads used by Maynard [Mayn94]. Note that the branch frequencies are the same for the multi-user commercial workloads as the branch frequencies of eqntott, espresso, and gcc. Cvetanovic [Cvet94] also shows that the load frequencies for compress, eqntott, espresso, gcc, tpcb1, and tpcb2 (He executes TPC with different inputs than [Mayn94]) are similar.

	compress	eqntott	espresso	gcc
LS	38%	37%	32%	31%
SHIFT	19%	6%	10%	7%
ALU	27%	35%	39%	41%
BRN	13%	19%	17%	19%
SYS	0%	0%	0%	0%
DIV	0%	0%	0%	0%

Table 8: Percentage of instructions executed by each execution unit.

TPC-A	TPC-C	Netperf	Laddis	Kenbus	Sdet
16.9%	18.9%	18.6%	18.9%	16.3%	17.8%

Table 9: Percentage of branch instructions in commercial multi-user applications.

Commercial multi-user benchmarks from the paper [Mayn94]

By comparing the miss rates measured for larger integer benchmarks to the miss rates found for our benchmarks, the performance of blocking, non-blocking, and pipelined first level data caches can be approximated for benchmarks with large data working sets and realistic cache sizes. Figure 4 shows the miss rates found by [Mayn94] for the six commercial multi-user integer benchmarks in Table 9 for first level data cache sizes of 8KB to 1024KB. For a standard processor on-chip data cache size of 32KB, the smallest miss rate from Figure 4 is 2.3 misses per one hundred instructions. For a two-pipelined and a non-pipelined cache to have the same performance on a dynamic superscalar processor we found that an average miss rate per one hundred instructions of 2.3 was needed with a six cycle miss penalty. This means that with the miss rates shown in Figure 4, our results predict that if the commercial benchmarks were executed on our dynamic superscalar processor with a two-pipelined non-blocking cache, we would see better performance than if they were executed with a non-pipelined non-blocking cache.

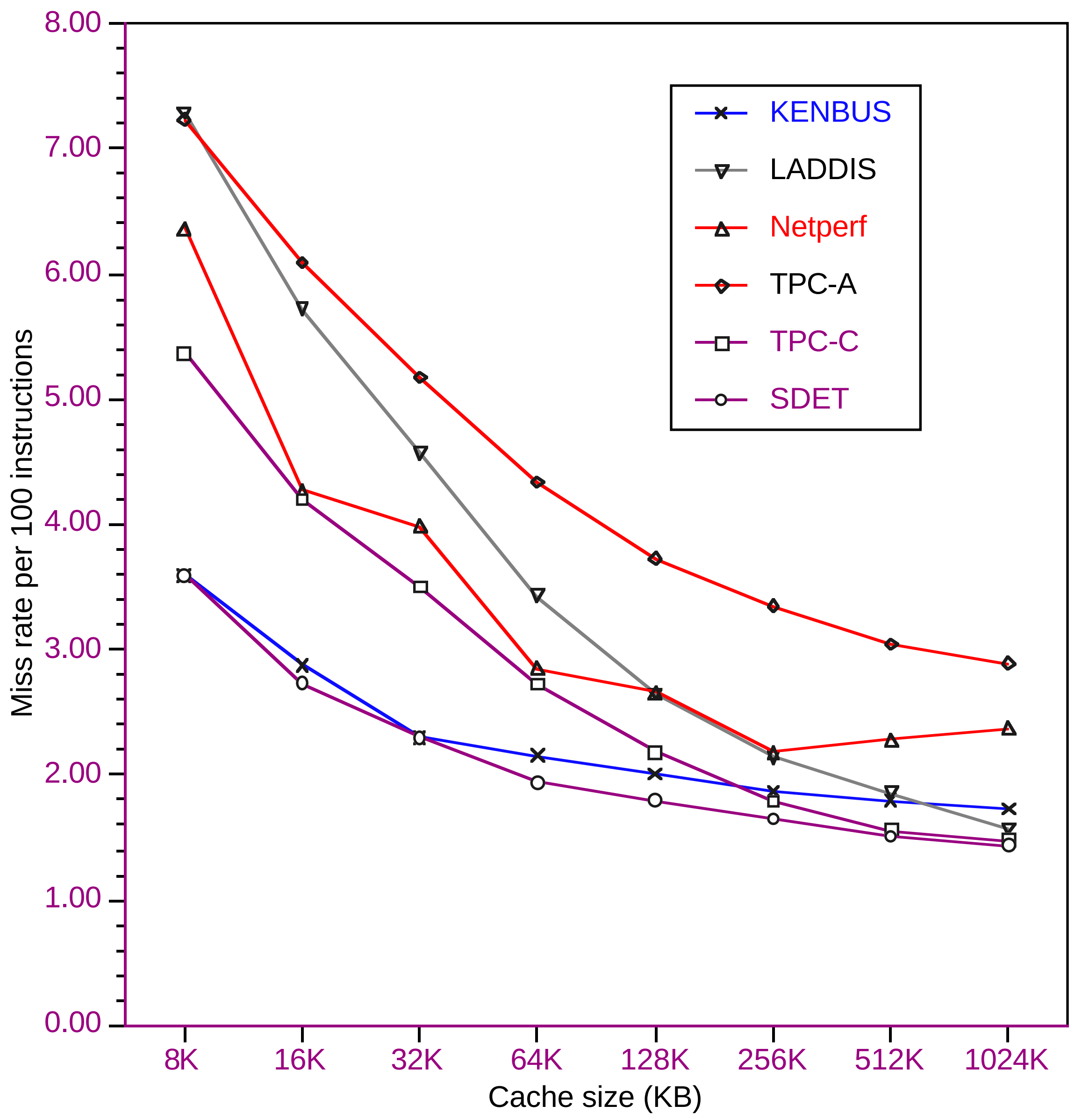

Figure 4: Cache miss rates of multi-user commercial workloads with a 32 byte line and direct mapped caches [Mayn94].

	CPI hit(2)	CPI hit(3)
compress	0.07	0.12
eqntott	0.07	0.15
espresso	0.08	0.15
gcc	0.04	0.10

Table 10: $CPI_{hit(2)}$ and $CPI_{hit(3)}$ for each benchmark due to pipelining.

6 cycle miss penalty		**12 cycle miss penalty**	
compress	51%	compress	51%
eqntott	48%	eqntott	47%
espresso	8%	espresso	6%
gcc	7%	gcc	7%

Table 11: $CPI_{overlap}$ as a percentage of CPI_{miss}.

With the use of the formulas from the introduction, the data from Table 5, and the conservative assumption of a perfect level two cache, we can estimate CPI_{memory} for the commercial benchmarks executing on our dynamic superscalar processor. The possible effects on CPI_{memory} for the dynamic processor with a two-pipelined 512KB data cache in place of a non-pipelined 32KB data cache can be shown by calculating the change in CPI_{memory} for the full range of our experimental values of $CPI_{overlap}$ and $CPI_{hit(2)}$ for miss penalties of six and twelve cycles.

Table 10 shows the values of $CPI_{hit(2)}$ and $CPI_{hit(3)}$ calculated for the two and three pipelined caches for each of our benchmarks, and Table 11 shows values of $CPI_{overlap}$ as a percent of CPI_{miss} for each benchmark. Note that a $CPI_{overlap}$ of zero is equivalent to a blocking cache. This means that espresso and gcc don't show a large benefit from a non-blocking cache, whereas compress and eqntott show an effective miss penalty of about one half of the miss penalty of a blocking cache.

Table 12 contains the changes in CPI_{memory} for miss penalties of six and twelve cycles, for $CPI_{overlap}$ values of zero and 51% of CPI_{miss}, and for $CPI_{hit(2)}$ values of 0.04 and 0.08. Note that the results shown in Table 12 are conservative because a perfect level two cache is assumed. The memory performance variation for each benchmark as miss penalty, $CPI_{overlap}$, and $CPI_{hit(2)}$ are varied is tabulated in order of increasing performance in Table 12. As expected, an increase in the miss penalty increases the performance benefits of a two-pipelined cache over a non-pipelined cache. The greater the application performance improves with a non-blocking cache, the larger the value of $CPI_{overlap}$, creating a smaller effective miss penalty, and therefore smaller performance improvements for a pipelined cache. If the dynamic processor is able to hide more of the additional hit time of the pipelined cache, $CPI_{hit(2)}$ will be smaller and pipelined cache performance will improve. Miss rate is still the most important factor in determining the usefulness of pipelining. The larger the miss rate, and the more steeply the slope of the miss rate curve declines with increasing cache size, the better the performance benefit of a pipelined cache over a non-pipelined cache.

L1 Miss Penalty	CPI overlap	CPI hit(2)	KENBUS	LADDIS	Netperf	TPC-A	TPC-C	SDET	Average
6	51%	0.08	-0.07	-0.00	-0.03	-0.02	-0.02	-0.06	-0.03
6	51%	0.04	-0.03	0.04	0.01	0.02	0.02	-0.02	0.01
12	51%	0.08	-0.05	0.08	0.02	0.05	0.03	-0.03	0.02
6	0%	0.08	-0.05	0.08	0.02	0.05	0.04	-0.03	0.02
12	51%	0.04	-0.01	0.12	0.06	0.09	0.07	0.01	0.06
6	0%	0.04	-0.01	0.12	0.06	0.09	0.08	0.01	0.06
12	0%	0.08	-0.02	0.25	0.13	0.18	0.15	0.01	0.12
12	0%	0.04	0.02	0.29	0.17	0.22	0.19	0.05	0.16

Table 12: Conservative estimates of the decrease in CPI due to using a two-pipelined cache over a non-pipelined cache for varying miss penalty, $CPI_{overlap}$, and $CPI_{hit(2)}$ ordered by performance gain.

Of the six multi-user commercial benchmarks, LADDIS performs the best with the pipelined cache, and KENBUS performs the worst. For LADDIS, a pipelined cache improves memory performance for all but the worst case conditions for pipelined cache performance (see Table 12). Even though LADDIS does not have the highest miss rate, its does have the largest improvement in miss rate for a 32KB cache compared to a 512KB cache (see Figure 4). This shows that having a large miss rate is not enough as there must also be a decrease in miss rate with an increase in cache size to get any performance improvement from the larger cache sizes possible with pipelining. At the other extreme, KENBUS performance actually decreases with a pipelined cache in all but the best case. The low performance improvement data for KENBUS is due to a low miss rate and the fact that the miss rate barely decreases with an increase in cache size. This means that the miss rate characteristics exhibited by KENBUS would not suggest using a pipelined cache in place of a non-pipelined cache.

6. Conclusion

The performance of blocking, non-blocking, and pipelined on-chip first level data caches using integer applications with data cache sizes from 2KB to 64KB has been evaluated. We have shown through simulations on a detailed processor simulator that the components of $\text{CPI}_{\text{memory}}$ due to cache misses, hit time, and overlapping of cache misses can be isolated and used to study the effects of non-blocking and pipelined caches used with a dynamic superscalar processor.

Our simulations showed that non-blocking caches will always improve the performance of a dynamic superscalar processor, and that the incremental cost and design complexity of adding a non-blocking cache to a dynamic superscalar processor is very small. We also showed that dynamic instruction scheduling can be used to hide a large portion of the additional hit time of a pipelined cache to make pipelined caches desirable at lower miss rates. Our results show that for integer applications, roughly two thirds of the additional hit time of a pipelined cache can be hidden by dynamic instruction scheduling for two and three deep pipelined caches.

Pipelined caches allow a trade-off of longer hit times for large cache size, and can be used in place of small direct mapped data caches with single cycle hit times. A pipelined cache should only be used if the processor has a large miss penalty, or if the applications executed on the processor have high miss rates. The reduction in miss rate needs to be great enough to make up for the increase in hit time due to the pipelined cache. Since commercial multi-user workloads have high miss rates and the dynamic superscalar processor can hide up to two-thirds of the additional hit time of a pipelined cache, superscalar processors executing commercial workloads are good candidates for pipelined caches.

Our results using integer SPEC92 benchmarks suggested miss rates at which pipelined caches improve processor performance. Comparing these miss rates to the miss rates for commercial multi-user applications presented by Maynard [Mayn94], our miss rates implied that a dynamic superscalar processor with a two-pipelined cache running the commercial benchmarks should break about even with a miss penalty of six cycles and improve performance with a miss penalty of twelve cycles. When we calculated the change in CPI from executing the commercial benchmarks with a 512KB two-pipelined cache instead of a 32KB non-pipelined cache, we found with what conditions the multi-user commercial workloads would execute faster on our dynamic superscalar processor with the two-pipelined cache.

7. References

[Aspr93] Tom Asprey, Gregory S. Averill, Eric DeLano, Russ Mason, Bill Weiner, and Jeff Yetter, "Performance Features of the PA7100 Microprocessor", IEEE Micro, June 1993, pp. 22-35.

[Chap91] Terry I. Chappell, Barbara A. Chappell, Stanley E. Schuster, James W. Allen, Stephen P. Klepner, Rajiv V. Joshi, and Robert L. Franch, "A 2-ns Cycle, 3.8-ns Access 512-kb CMOS ECL SRAM with a Fully Pipelined Architecture", IEEE Journal of Solid-State Circuits, Vol. 26, No. 11, November 1991, pp. 1577-1585.

[Chen92] Tien-Fu Chen and Jean-Loup Baer, "Reducing Memory Latency via Non-blocking and Prefetching Caches", ASPLOS-V, Boston, Massachusetts, October 12-15, 1992.

[Chen94] Chung-Ho Chen and Arun K. Somani, "A Unified Architectural Tradeoff Methodology", ISCA-21, Chicago, Illinois, April 18-21, 1994, pp. 348-357.

[Conte92] Thomas A. Conte, “Tradeoffs in Processor/Memory Interfaces for Superscalar Processors, Proceedings of the 25th Annual International Symposium on Microarchitecture, Portland, Or 1992.

[Cvet94] Zarka Cvetanovic and Dileep Bhandarkar, “Characterization of Alpha AXP Performance Using TP and SPEC Workloads, The 21st Annual International Symposium on Computer Architecture, April 18-21, 1994, pp. 60-70.

[Fark94] Keith I. Farkas and Norman P. Jouppi, "Complexity/Performance Tradeoffs with Non-Blocking Loads", ISCA-21, Chicago, Illinois, April 18-21, 1994, pp. 211-222.

[Farr94] Mathew Farrens, Gary Tyson, and Andrew R. Pleszkun, "A Study of Single-Chip Processor/Cache Organizations for Large Numbers of Transistors", ISCA-21, Chicago, Illinois, April 18-21, 1994, pp. 338-347.

[Gee93] Jeffrey D. Gee, Mark D. Hill, Dionisios N. Pnevmatikatos, and Alan Jay Smith, "Cache Performance of the SPEC92 Benchmark Suite", IEEE Micro, August 1993, pp. 17-27.

[Ghar92] Kourosh Gharachorloo, Anoop Gupta, and John Hennessy, “Hiding Memory Latency using Dynamic Scheduling in Shared-Memory Multiprocessors”, ISCA-20, San Diego, California, May 16-19, 1993, pp. 23-33.

[Gupta91] Anoop Gupta, John Hennessy, Kourosh Gharachorloo, Tod Mowry, and Wolf-Dietrich Weber, "Comparative Evaluation of Latency Reducing and Tolerating Techiques", ISCA-19, Toronto, Canada, May 27-30, 1991, pp. 254-263.

[Gwen93] Linley Gwennap, “TFP Designed for Tremendous Floating Point”, Microprocessor Report, Vol. 7, No 11, August 23, 1993, pp. 9-13.

[Henn90] John L. Hennessy and David A. Patterson, “Computer Architecture a Quantitative Approach”, Morgan Kaufmann Publishers, Inc, 1990.

[John91] Mike Johnson, "Superscalar Microprocessor Design", Prentice-Hall Inc, 1991.

[Joup93] Norman P. Jouppi, "Cache Write Policies and Performance", ISCA-20, San Diego, California, May 16-19, 1993.

[Krof81] David Kroft, "Lockup-Free Instruction Fetch/Prefetch Cache Organization", ISCA-8, 1993 pp. 81-87.

[Lee84] J. K. F. Lee and A. J. Smith, “Branch Prediction Strategies and Branch Target Buffer Design”, IEEE Computer Magazine, Vol. 17 (January 1984), pp. 6-22.

[Mayn94] Ann Marie Grizzaffi Maynard, Colette M. Donnelly, and Bret R. Olszewski, "Contrasting Characteristics and Cache Performance of Technical and Multi-User Commercial Workloads", ASPLOS-VI, San Jose, CA, October 4-7, 1994.

[McLe93] Edward McLellan, "The Alpha AXP Architecture and 21064 Processor", IEEE Micro, June 1993, pp. 36-47.

[MIPS94] MIPS Technologies, Incorporated, "R10000 Microprocessor Product Overview", MIPS Open RISC Technology, MIPS Technologies, Incorporated, October 1994.

[NEC94] NEC Corporation, "16M bit Synchronous DRAM, preliminary data sheet", NEC Corporation, March 1994.

[Oluk92] Kunle Olukotun, Trevor Mudge, and Richard Brown, "Performance Optimization of Pipelined Primary Caches", ISCA-19, Gold Coast, Australia, May 19-21, 1992, pp. 181-190.

[Przy88] Przybylski, S., M. Horowitz, and J. Hennessy, “Performance Tradeoffs in Cache Design”, Proceedings of the 15th Annual International Symposium on Computer Architecture, June 1988, pp.290-298.

[Rau93] B. Ramakrishna Rau and Joseph A. Fisher, "Instruction-Level Parallel Processing: History, Overview, and Perspective", Journal of Supercomputing, 7, 1993, pp. 9-50.

[Sezn93] Andre Seznec, "A case for two-way skewed-associative caches", ISCA-20, San Diego, California, May 16-19, 1993.

[Smith81] Smith, James E. “A study of Branch Prediction Strategies.”, Proceedings of the 8th Annual International Symposium on Computer Architecture, 1981, pp. 135-147.

[Smith85] James E. Smith and Andrew R. Pleszkun, “Implementation of Precise Interrupts in Pipelined Processors”, ISCA, 1985, pp. 36-44.

[Toma67] Tomasulo, R. M., “An Efficient Algorithm for Exploiting Multiple Arithmetic Units.”, IBM Journal of Research and Development, Vol. 11 (January 1967), pp. 25-33.

[Uht86] Uht, A. K., “An Efficient Hardware Algorithm to Extract Concurrency from General Purpose Code”, Proceedings of the Nineteenth Annual Hawaii International Conference on System Sciences, 1986, pp. 41-50.

[Upto94] Michael Upton, Thomas Huff, Trevor Mudge, and Richard Brown, "Resource Allocation in a High Clock Rate Microprocessor", ASPLOS-VI, San Jose, CA, October 4-7, 1994, pp. 98-109.

[Wall93] David W. Wall, "Limits of Instruction-Level Parallelism", WRL Research Report 93/6, Western Research Laboratory, 250 University Ave., Palo Alto, CA, 94301, November 1993.

[Wilt94] Steven J. E. Wilton and Norman P. Jouppi, "An Enhanced Access and Cycle Time Model for On-Chip Caches", WRL Research Report 93/5, Western Research Laboratory, 250 University Ave., Palo Alto, CA, 94301.