Designing High Bandwidth On-Chip Caches

Kenneth M. Wilson and Kunle Olukotun

Computer Systems Laboratory
Stanford University
Stanford, CA 94305
{kwilson, kunle}@ogun.stanford.edu

Abstract

In this paper we evaluate the performance of high bandwidth caches that employ multiple ports, multiple cycle hit times, on-chip DRAM, and a line buffer to find the organization that provides the best processor performance. Processor performance is measured in execution time using a dynamic superscalar processor running realistic benchmarks that include operating system references. The results show that a large dual-ported multi-cycle pipelined SRAM cache with a line buffer maximizes processor performance. A large pipelined cache provides both a low miss rate and a high CPU clock frequency. Dual-porting the cache and the use of a line buffer provide the bandwidth needed by a dynamic superscalar processor. In addition, the line buffer makes the pipelined dual-ported cache the best option by increasing cache port bandwidth and hiding cache latency.

1. Introduction

It is well known that the on-chip memory system of a microprocessor is a key determinant of microprocessor performance. Previously, the main concerns of memory designers were cache size, associativity, and line size. As the performance of microprocessors increases, instruction and data bandwidth demands lead chip designers to create more complicated on-chip memory systems. Currently, designers are considering multi-ported caches, multi-cycle caches, multi-level caches, and even caches made with on-chip DRAM. Though each of these has been covered individually in previous work [Joup94, MIPS94, Oluk92, Saul96, Shim96, Sohi91, Wils96], there are no papers that investigate the interaction of these different methods to find the organizations that work well together.

There are many options available for increasing cache bandwidth. One straightforward way of increasing cache bandwidth is to provide multiple cache ports that can be independently addressed by the processor's load/store unit in the same cycle. Ideally, each cache port operates independently, with no increase in hit time and only a small increase in chip area. Another way to increase memory bandwidth is to enlarge the primary data cache. Enlarging the primary data cache decreases the cache's miss rate, thereby decreasing the average response time of the processor's memory system. Increasing the size of the primary data cache causes the hit time of the cache to increase. Since accessing the data cache is usually on the processor's critical timing path, the hit time of larger caches may be pipelined to avoid increasing the processor's cycle time. The extra processor cycles can be partially hidden by the use of a dynamic superscalar processor. If a very large on-chip cache is desired, the designer may choose to use DRAM in place of SRAM. With improvements in DRAM technology it is now possible to include DRAM memory on-chip [Saul96][Shim96]. Caches containing DRAM can have much larger capacities than standard SRAM cache designs for equal processor die area.

One way to increase memory bandwidth and at the same time hide the increase in hit time needed by a larger cache is to add a small buffer with a one cycle access time to the processor's load/store execution unit. This level zero cache can be implemented as a small line buffer [Wils96]. A line buffer holds cache data inside the processor's load/store unit to allow a load to recently accessed data to be satisfied from the line buffer instead of from the cache. This prevents the load from occupying the primary data cache ports, allowing other loads to access the cache earlier. When a line buffer is added to a processor with a multi-cycle cache, not only does the line buffer reduce the utilization of the cache ports, but the line buffer also reduces the average latency of load instructions by returning data in a single cycle.

To understand the interaction between different high bandwidth cache organizations, we evaluate the performance of caches that employ combinations of multi-port, multi-cycle access, on-chip DRAM, and a line buffer. In the course of this evaluation we find the combination of features and cache parameters that provides the best processor performance. To determine processor performance we use an accurate model of a four issue dynamic superscalar processor with non-blocking caches. The rest of this paper is organized as follows: The high bandwidth on-chip cache designs for this investigation are described in section 2. The experimental methodology for simulating the architectures is discussed in section 3. The results that characterize the design space by looking at the interactions between different on-chip cache configurations are presented in section 4. First, the processor instructions per cycle (IPC) results show the effect of the different cache configurations on processor performance when changes in cycle time are not taken into account. Second, we combine the IPC results with the proces-

sor cycle times determined in section 2 to find the application execution times of our benchmarks for various cache size and structure. Section 5 gives our conclusions.

2. High Bandwidth Cache Design

In this section we describe alternative high bandwidth cache designs and look at their impact on cache access time. The impact on cache access time is important because we are interested in the largest cache which has an access time within the processor's cycle time. We assume that the hit time of the cache is on the critical timing path of the processor and therefore directly impacts the processor's cycle time. If this is not the case, a larger slower cache should be built to improve the processor's performance.

To measure cache access time we use the technology independent delay measurement technique called *fan-out of four* (FO4) [Horo92]. One FO4 is the delay of a signal passing through an inverter driving a load capacitance that is four times larger than its input (e.g. driving four equivalently sized gates). Circuits can then be judged by comparing their delay to the delay of the inverter with a fan-out of four. Once FO4 is used to calculate the change in delay for a new cache structure, we find the resulting cache access time by using the fact that a processor with a single-ported single-cycle 8 Kbyte primary data cache on the processor's critical timing path has a cycle time of 25 FO4 [Horo96].

A modified version of *cacti* [Wilt96] is used to quantify the relationship between cache size and cache hit time. Cacti models a 0.5um CMOS process to find the cache access time for SRAM cache sizes varying from 4 Kbytes to 256 Kbytes. Even though cacti produces cache access times for larger caches, cacti does not produce the best cache design due to restrictions on cacti's design space. Since we are interested in SRAM cache sizes of up to 1 Mbyte, we modified cacti to support larger caches by changing cacti's parameters to increase the size of its design space. For instance, we increased the possible number of cache sub-arrays from eight to thirty-two. The resulting cache access times are converted to FO4 and graphed under the label "Single-Ported Cache" in Figure 1. Note that the shape of the curve for the cache access times is as expected and is just an extension to the cache access curve found in [Wilt94]. The cacti results for eight-way banked caches are also shown in Figure 1 and will be explained in the next section.

2.1 Multi-Ported Caches

Multiple cache ports are required to satisfy the bandwidth demands of modern superscalar processors. In a four issue dynamic superscalar processor with a 32 Kbyte primary data cache, adding a second cache port increases processor performance by an average of 25% [Wils96]. As expected, there are diminishing returns in performance from increasing the number of cache ports beyond two. Increasing the number of cache ports to three and four increased processor performance by only four and one percent respectively. For this reason we concentrate on caches with two ideal ports. An ideal cache port operates independently of any other cache port, is accessible every cycle, and has a cache hit time of one cycle. In reality multiple cache ports are usually implemented by banking or duplicating the primary data cache.

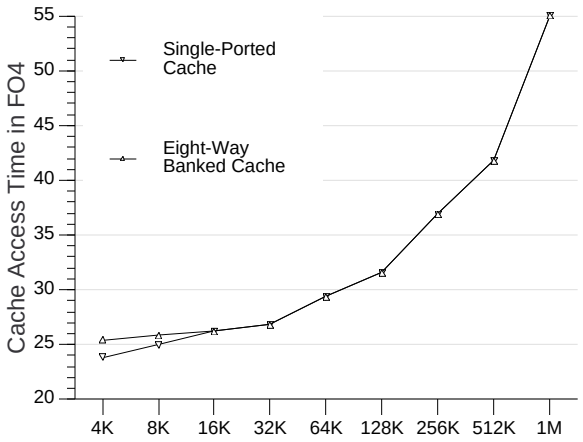

Figure 1: Access times for single-ported and eight-way banked caches.

Banked caches consist of multiple independently addressed banks [Sohi91]. Recent microprocessors have used this technique to improve memory bandwidth. The MIPS R10000 [MIPS94][Yeag96] implements a two-way banked cache. Splitting cache accesses into two banks allows two cache accesses to be overlapped if the two cache accesses are to different banks. Given an ideal cache access stream, this technique can double the bandwidth of the primary data cache. However, in practice this technique suffers from bank conflicts and may not increase performance if enough references address the same bank.

One drawback of implementing a banked cache is an increase in the number of wires required to interconnect the banks. This increases both the area consumed by the cache and the wire delay. The delay due to the increased wire lengths is offset by the decrease in delay required to address the smaller cache array. In the case of small caches, banking results in an increase in both cache access time and cache area. For larger caches banking becomes a trade-off between processor die area and cache access time. In fact the cache designs produced by cacti are already eight-way banked internally for cache sizes of 16 Kbytes and greater.

We use cacti to find the cache access times of eight-way banked caches that vary in capacity from 4 Kbytes to 1 M byte. Cacti considers cache organizations within a design space and picks the best organization. We force cacti to consider only caches that contain eight or more banks. This produces the access times for eight-way or greater internally banked SRAM caches graphed in Figure 1. As expected, the access times presented in Figure 1 for eight-way banked caches are greater than the access times for single-ported cache sizes of less than 16 Kbytes. The eight-way banked and single-ported caches have the same access times for cache sizes of 16 Kbytes and greater because single-ported caches of 16 Kbytes and greater are already at least eight-way internally banked. Assuming there is no timing penalty for changing an internally banked cache to an externally banked cache, the eight-way banked curve of Figure 1 can be used to represent the cache access times for externally eight-way banked caches.

Another way to build a multi-ported cache is to duplicate the primary data cache such as in the DEC Alpha 21164 [Edmo95]. The Alpha contains two copies of the primary data cache. The dual-ported cache of the DEC Alpha has the performance drawback that all stores must be sent to both cache ports simultaneously so that both copies of the data cache are consistent. Using both cache ports for each store should not significantly degrade performance because stores can be buffered and bypassed to allow loads to access the cache first. We make the assumption that stores can be buffered until both cache ports are not servicing load instructions. This means that stores do not degrade machine performance; therefore ideal multiple cache ports are used to model duplicate caches.

Even though duplicating the data cache causes a doubling of the die area needed for the primary data cache, the only timing cost is the added delay due to adding one more write port to the load/store buffer. Since adding an additional write port causes only a small increase in delay, we assume that the added delay can be eliminated with increased effort spent on the circuit design. This means that the access times for single-ported caches shown in Figure 1 can also be used for duplicate caches.

2.2 Pipelined Caches

With application working sets increasing and the gap between memory access time and processor cycle time expanding, some chip designers will decide to implement larger caches that require multi-cycle hit times. Using multi-cycle hit times allows the use of larger and more complicated primary caches without impacting processor cycle time. This is especially important for primary data caches. The extra hit time of a multi-cycle cache can be partially hidden by pipelining the cache [Oluk92], and using a dynamic superscalar processor [Wils95].

The size of a cache that can be accessed in two or three processor cycles can be determined from the cycle times shown in Figure 1. If a processor has a cycle time of 25 FO4, an 8 Kbyte cache can be accessed in one processor cycle, a 512 Kbyte cache can be accessed in 1.67 cycles, and a 1 Mbyte cache can be accessed in 2.20 cycles. Pipelining requires the addition of a latch with a delay of 1.5 FO4, to create a 512 Kbyte cache that can be implemented on-chip and be accessed in two processor cycles. Further, Figure 1 shows that if a 1 Mbyte cache is desired, either the processor cycle time has to be increased to make this cache fit into a two cycle hit time, or the hit time of the 1 Mbyte cache must be increased to three processor cycles. In this study we do not consider on-chip SRAM caches larger than 1 Mbyte.

2.3. Increasing Cache Port Efficiency

A line buffer is a small fully-set-associative multi-ported level-zero cache located within the processor's load/store execution unit [Wils96]. When a line buffer is added to a processor's load/store execution unit, it reduces the number of accesses to the cache ports by satisfying a large percentage of the load instructions before they access a cache port. By decreasing the number of accesses to the cache ports, the line buffer reduces the need for multiple cache ports. Another advantage of a line buffer is that it returns recently accessed cache data in a single cycle, which helps hide the additional latency of pipelined caches.

2.4. DRAM Caches

A fairly recent proposal for building an on-chip cache is to use DRAM on the processor die [Saul96][Shim96]. The main advantage of DRAM is that it allows much larger caches to be constructed on-chip. The chief disadvantage is longer cache hit times. Much of the performance effect of the longer hit times can be hidden through the addition of a small first level cache and the latency hiding abilities of a dynamic superscalar processor. The extra latency can be further reduced by using a line buffer to return data with high spatial and temporal locality in a single processor cycle.

The biggest hurdle to placing DRAM caches on the processor die is that the technologies to create DRAMs and processors are completely different. With today's technologies, it is only possible to build a low performance processor on a DRAM die; this might be useful for microcontrollers and low-end CPUs. High-end CPUs are more likely to use the option of building DRAM on the processor die. Due to the differing technologies this will make the DRAM consume about twice the die area and decrease the DRAM performance.

One advantage of using DRAM for on-chip caches is that the row buffers of the DRAM can be combined to make a first level data cache. In [Saul96], two row buffers for every DRAM bank were used to make a 16 Kbyte primary data cache with two-way-set-associativity. The disadvantage to using the row buffers as a primary data cache is that each row buffer contains 512 bytes. This means that the line size of the 16 Kbyte data cache is 512 bytes. The longer line size reduces the hit rate of the row buffer data cache, but should not have too great an impact since the miss time of the row buffers is the time to access the DRAM cache. For this study, we use a DRAM cache size of 4 Mbytes and do not use a secondary off-chip cache due to the large size of the DRAM cache.

The on-chip DRAM hit time is not completely understood at this time, but [Saul96] predicts a six cycle hit time is possible for a 200MHz processor. They also predict that using the DRAM's row buffers as a first level cache allows a cache hit time of one processor cycle if the data is in the row buffer. These hit times are very aggressive. An actual implementation of a processor in a DRAM technology [Shim96] used a separate 2 Kbyte SRAM data cache with a one cycle hit time and a miss time to DRAM of five cycles, but with a processor frequency of only 66 Mhz. It takes a DRAM cache design similar to [Saul96] to provide enough bandwidth to compete with standard on-chip SRAM caches. Due to our concern for adequate bandwidth we use a one cycle hit time 16 Kbyte two-way-set-associative row buffer cache, but vary the hit time of the DRAM cache from six to eight cycles to get an idea of how processor performance decreases as the hit time of the DRAM increases.

3. Simulation Methodology

This section describes the dynamic superscalar processor used to evaluate the different on-chip memory structures, and the SimOS [Rose95b] environment under which the processor is simulated. We also discuss the simulation methods and benchmarks used to produce the results presented in this study.

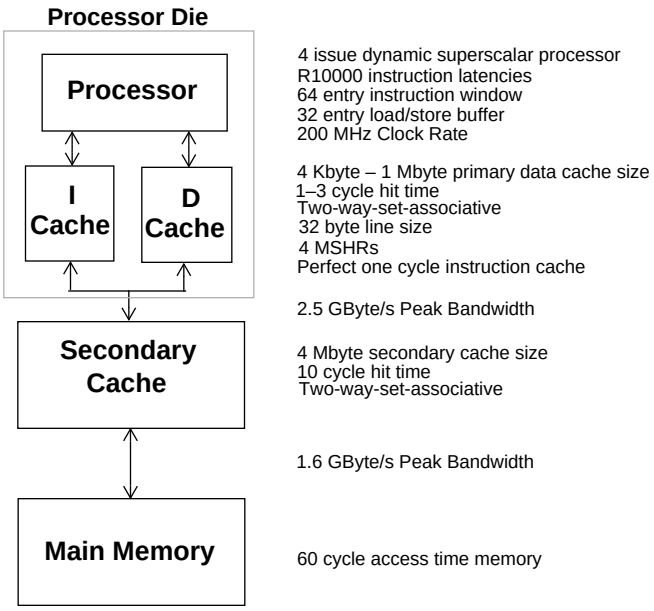

Figure 2: Processor and memory subsystem.

3.1. Processor and Memory Subsystem

The processor used for this study is a four issue dynamic superscalar processor with R10000 instruction latencies and a 200 Mhz clock rate [Yeag96, Gwen94, MIPS94]. The processor and memory subsystem are summarized in Figure 2. Unlike the R10000, our processor model has a 64 entry reorder buffer and a 32 entry load/store buffer. To focus the study on the performance of the memory subsystem, there are no restrictions on the type of instructions that can be issued each cycle.

The memory subsystem consists of separate primary instruction and data caches, a unified secondary cache, and main memory. The primary instruction cache is a perfect cache that always hits and responds in a single cycle. The primary data cache is a two-way-set-associative cache with 32 byte lines and a single cycle access time, or a fully pipelined multi-cycle access time. Note that the load latency is actually one cycle greater than the cache access time due to the load's address calculation. The data cache is lock-up-free, but the cache size is not fixed, allowing us to investigate the performance effects of various cache sizes. The second level cache is a 4 Mbyte two-way-set-associative cache with 64 byte lines and a ten cycle (50ns) access time. Four miss status handling registers (MSHR) [Fark94] implement the lock-up-free cache and are located in the primary data cache to support misses for up to four data cache lines. Main memory has a sixty cycle (300ns) access time.

For some applications high performance off-chip memory bandwidth is critical for good performance. Our memory organization supports a bandwidth of 1.6 GBytes/second peak between the second level cache and main memory, and 2.5 GBytes/second peak between the processor chip and the second level cache. Though these memory bandwidths may seem very impressive, the DEC Alpha 21264 [Kell96] supports even greater bandwidths of 2 GBytes/second from the second level cache to memory and 4 GBytes/second from the processor die to the second level cache.

3.2. SimOS Environment

SimOS [Rose95b] is a complete machine simulation environment containing models of all the hardware present on modern computer systems including the CPU, memory subsystems, and I/O devices. It can simulate the hardware with enough speed and detail to boot and run a commercial Unix-based operating system, Silicon Graphics IRIX version 5.3, and arbitrary application programs that run on IRIX.

This study uses the MXS [Benn95] CPU simulator running under SimOS. MXS is a detailed CPU simulator that simulates the machine model described earlier in this section. This CPU simulator models a dynamic superscalar processor that supports multiple instruction issue, out-of-order execution, hardware branch prediction, and lock-up-free caches. The simulator's operation is parameterized and can be configured to accurately model a variety of machines.

SPEC95 integer benchmarks

gcc	Builds SPARC code
li	LISP interpreter
compress	Compresses and decompresses file in memory

SPEC95 floating point benchmarks

tomcatv	Mesh-generation program
su2cor	Quantum physics; Monte Carlo simulation
apsi	Solves problems regarding temperature, wind, velocity, and distribution of pollutants

SimOS multiprogramming benchmarks

pmake	Two compilation processes for 17 files
database	Sybase SQL server using bank/customer transaction processing modeled after the TPC-B transaction processing benchmark [Gray93]
VCS	Simulates the FLASH MAGIC chip [Kusk94] using the Chronologics VCS simulator

Table 1: The nine benchmarks.

3.3. Benchmarks

The performance of nine realistic benchmarks is used to evaluate the high bandwidth cache designs that are proposed in section two. Table 1 shows that the nine benchmarks are made up of three integer SPEC95 benchmarks, three floating point SPEC95 benchmarks, and three multiprogramming applications. The newer SPEC95 benchmarks were chosen over SPEC92 because their larger working set sizes stress the memory system much more than the SPEC92 benchmarks. The three SimOS benchmarks were chosen because they represent realistic multiprogramming workloads from the areas of programming, database transaction processing, and engineering simulation.

Each benchmark is simulated for over 100 million instructions within an interesting portion of its execution. The breakdown of execution time spent in kernel, user, and idle for each benchmark is shown in Table 2. Idle is a result of waiting for I/O, and applications such as database spend a large portion of their execution in this mode. Since the processor is spinning in a tight loop in idle

mode, the instructions per cycle (IPC) for this mode could skew the results for benchmarks with significant idle time. For this reason, the IPC of the benchmarks during idle time is not factored into our performance measurements.

	Kernel	User	Idle	Load	Store
gcc	10.0	90.0	0.0	28.1	12.2
li	0.2	99.8	0.0	33.2	13.0
compress	8.4	91.6	0.0	34.5	8.0
tomcatv	0.4	99.6	0.0	26.9	8.5
su2cor	0.5	99.5	0.0	28.0	6.3
apsi	2.2	97.8	0.0	40.0	11.7
pmake	8.9	86.0	5.1	25.8	11.9
database	18.4	17.0	64.6	24.8	13.6
VCS	9.9	90.1	0.0	25.7	15.1

Table 2: Execution time percentages plus percentages of loads and stores in the instruction stream.

It is interesting to note that the floating point benchmarks almost never execute in kernel mode, while the integer benchmarks spend up to ten percent of their execution time in kernel mode. In addition to improving the accuracy of simulation results for integer benchmarks, the simulation of kernel mode, as well as user mode, allows the simulation of multiprogramming benchmarks such as pmake and kernel intensive benchmarks like database which spend 9% and 52% of their non-idle time in kernel mode respectively.

4. Results

To characterize the performance of the different on-chip cache designs we simulate them with cache sizes varying from 4 Kbytes to 1 Mbyte. The simulation results are shown for one representative benchmark from each of the three groups. Gcc represents the integer SPEC95 benchmarks, tomcatv the floating point SPEC95 benchmarks, and database the multiprogramming benchmarks. The results are presented in two stages. First, to understand how IPC changes with cache structure, we present results for a processor with a fixed processor cycle time. We consider the effect that pipelining has on IPC for an ideal 32 Kbyte multi-ported cache as the pipeline depth of the cache is increased from one to three cycles. We also perform the same study with banked caches and compare the results to ideal multi-ported caches. Then we show how the addition of a line buffer increases IPC by hiding memory latency. To conclude our study of IPC we look at DRAM caches. Second, using our knowledge of IPC we investigate which memory organization provides the smallest application execution time for different processor cycle times.

Before we begin looking at the high bandwidth memory organizations, we look at the miss rates per instruction of our nine benchmarks. The miss rates give an idea of the application working set size and the changes in benchmark performance with cache size. Figure 3 shows the effect on the cache misses per instruction of increasing the primary data cache size for each of the nine benchmarks used in this study. Note that the miss rates for each of our representative benchmarks are consistent with those of the other two benchmarks within their grouping.

From Figure 3 we see that the integer SPEC95 benchmarks have the lowest miss rates, the multiprogramming benchmarks have much larger miss rates, and the floating point SPEC95 benchmarks have slightly lower miss rates on average than the multiprogramming benchmarks. The integer benchmarks have the smallest miss rates because they have working sets that fit into smaller caches. We also see that for integer benchmarks changes in miss rate due to increases in cache size tend to be incremental. The multiprogramming benchmarks are integer applications with larger working sets than the integer SPEC95 benchmarks, so the change in miss rate with increasing cache size responds similarly for both; however the multiprogramming application's miss rates are greater. The floating point applications have large working sets and radical drops in miss rates at specific cache sizes. Their response to increasing cache size is different from the integer applications because floating point applications tend to access their data in more regular patterns than integer applications.

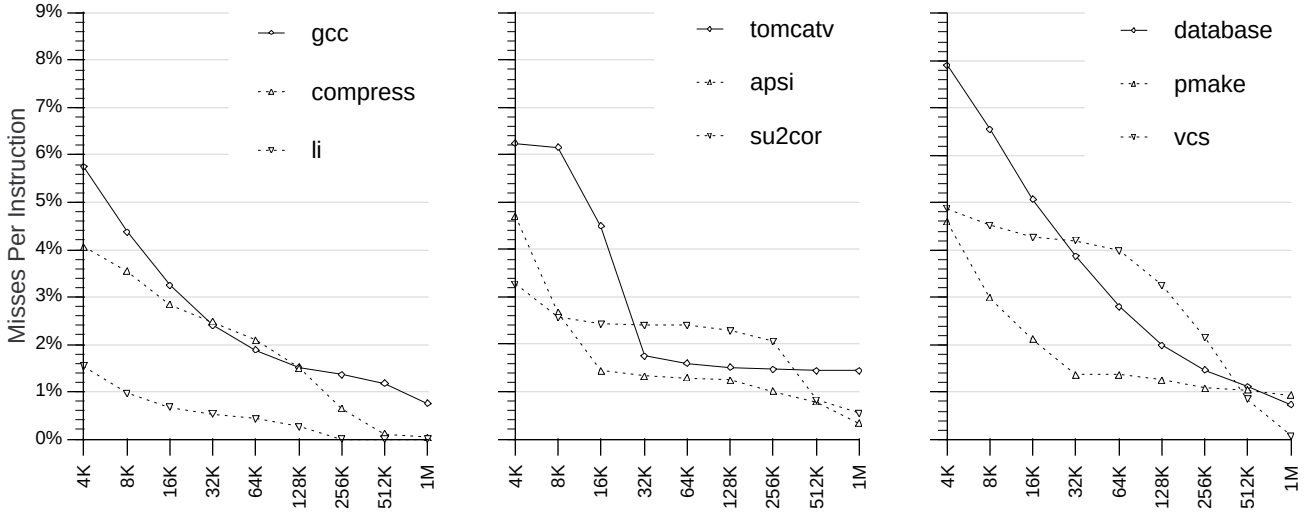

Figure 3: Benchmark miss rates in misses/instruction for single-ported caches.

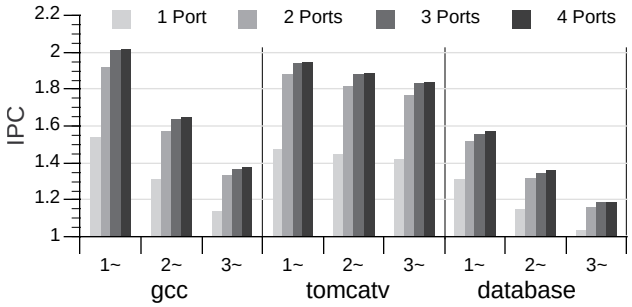

Figure 4: The performance of ideal multi-cycle multi-ported 32 Kbyte caches with a fixed processor cycle time.
A "~" is used to denote cycle

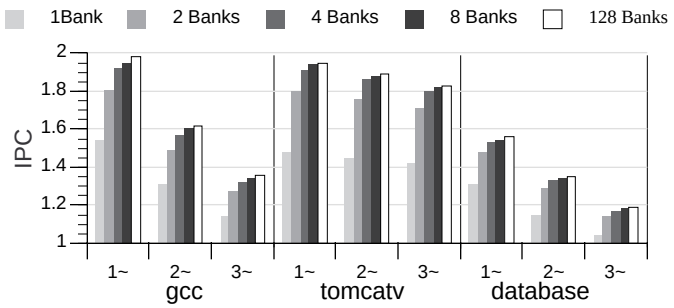

Figure 5: The performance of 32 Kbyte multi-cycle banked caches with a fixed processor cycle time.

4.1. Multi-Cycle, Multi-Ported Caches

We begin by presenting the IPC results of ideal multi-cycle caches with a fixed processor cycle time. For caches with multi-cycle hit times, a key question is: How much performance is lost as the hit time, measured in processor cycles, increases? Figure 4 shows how machine performance decreases with increasing hit time for one to four ideal cache ports with a 32 Kbyte primary data cache for gcc, tomcatv, and database. We see from Figure 4 that the decrease in performance for the benchmarks with multi-cycle hit times is almost independent of the number of ideal cache ports. In the case of integer applications, multi-cycle hit times degrade machine performance considerably. For the integer application gcc with two ideal cache ports, performance declines by 18% when the cache hit time is increased to two cycles, and performance declines another 15% when the cache hit time is increased to three cycles. Without the use of a dynamic superscalar processor there would be a larger performance decrease given the frequency of loads. The dynamic superscalar processor is able to hide a portion of the additional latency of the pipelined cache by reordering instruction execution to start the cache accesses as early as possible and rescheduling the load uses when necessary. For floating point applications, machine performance does not decrease nearly as much due to the large amount of instruction level parallelism (ILP) available. The dynamic superscalar processor uses the ILP available in tomcatv to hide a greater portion of the multi-cycle hit time than is possible for integer applications. Tomcatv shows only a 3% drop in performance for a two cycle hit time cache and another 3% drop in performance for a three cycle hit time. For database, a larger integer application, the performance decrease is 13% and 12% for two and three cycle hit times respectively. The smaller decline compared to gcc is due to database's larger working set which causes a higher miss rate and therefore reduces the performance impact due to increases in cache hit time.

Figure 5 shows the performance that results with banked caches of 1 to 128 banked ports and one to three cycle hit times for a 32 Kbyte primary data cache and fixed processor cycle time. By comparing Figure 5 with Figure 4, we see that a four-way banked cache provides lower performance than a cache with two ideal cache ports, while an eight-way banked cache provides higher performance than a cache with two ideal cache ports. The 128-way banked cache results show that the performance difference between an eight-way banked cache and a cache with a large number of banks is small.

Note that where banked caches are discussed we are interested in the number of external banks where each external bank has its own cache port. These results hold for all nine benchmarks and validate our focus on eight-way banked caches in this study.

By comparing Figure 5 to Figure 4, we see that the decrease in performance for multi-cycle hit times is about the same for an eight-way banked cache as it is for caches with two ideal cache ports. In addition, the slope of the performance decrease shown in Figures 4 and 5 is fairly constant for each benchmark as the hit time is increased from one to three cycles. The results in Figures 4 and 5 are for a fixed cache size of 32 Kbytes. In section 4.4 we investigate the performance effect of changes in cache size, processor cycle time, and cache hit time.

4.2. Line Buffer with Multi-Ported Multi-Cycle Caches

The use of a 32 64-bit entry fully-set-associative multi-ported line buffer allows a large percentage of load instructions to be satisfied from the buffer in a single cycle instead of from the data cache. Figure 6 shows the IPC of the three representative benchmarks for 32 Kbyte eight-way banked and duplicate caches of one, two, and three cycle hit times with and without a line buffer. In the legend of Figure 6, the label "LB" is added to show when a line buffer is included in the processor's load/store execution unit. In all cases the addition of a line buffer increases machine performance, but for the eight-way banked cache with a single cycle hit time the increase in machine performance is only about one-half of a percent. The reason for this small increase is that the line buffer was originally proposed to decrease the cache port access pressure when there is only one cache port. In a four issue processor with more than two cache ports, the load/store unit does not tend to get backed up with loads waiting to access the cache port; therefore, there is little performance gain due to the addition of a line buffer. The only exception is for very small caches, where the extra associativity provided by the line buffer increases processor performance.

On the other hand, a single cycle duplicate cache only contains two cache ports; therefore decreasing the cache port access pressure with a line buffer provides a greater processor performance increase of three percent. A previous study [Wils96] showed that using four ideal cache ports instead of two ideal cache ports increased performance by less than five percent for a one cycle 32 Kbyte cache. A line buffer achieves over 60% of that performance improvement without increasing the number of cache ports from two to four.

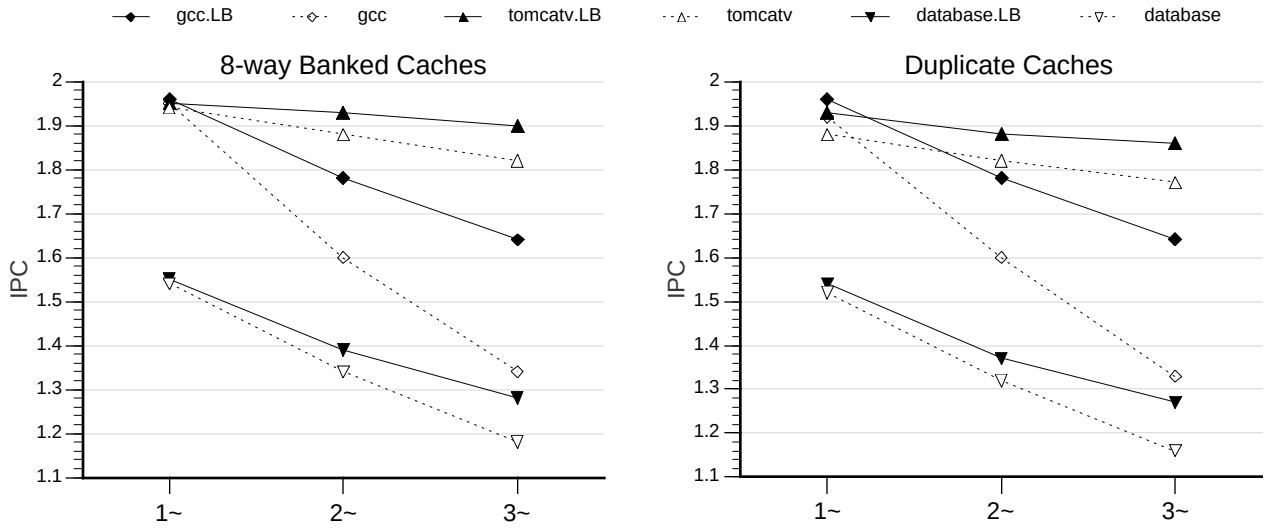

Figure 6: 32 Kbyte multi-cycle eight-way banked and duplicate caches with a line buffer and fixed processor cycle time.

Even though a line buffer is useful for multiple cache ports, it shows even more promise for pipelined caches. The line buffer hides a portion of the latency of a pipelined cache by returning data in a single cycle, thereby reducing the number of loads that need to wait for the multi-cycle hit time of the pipelined cache. As the degree of pipelining increases, the line buffer offers greater performance advantage. The use of a line buffer with an eight-way banked (duplicate) cache improves gcc's processor performance by 11% (11%) for a two cycle pipelined cache, and 22% (23%) for a three cycle pipelined cache. The database application sees a significant though smaller performance gain of 4% (4%) and 8% (9%) for two and three cycles of pipelining. Even though database is an integer application like gcc, it has a much larger working set than gcc and therefore a higher miss rate. Since the line buffer improves the performance of cache hits and not misses, we anticipate a smaller performance gain from database. Tomcatv has the smallest performance gain of only 3% (3%) and 4% (5%) for two and three cycles of pipelining. Figures 4 and 5 show that tomcatv has only a small drop in performance when a pipelined cache is used with a dynamic superscalar processor. Therefore the extra cache latency is already being hidden, providing less opportunity for the line buffer to improve processor performance. The line buffer causes a 50% reduction in the performance drop due to pipelining the cache for gcc, 50% for tomcatv, and 28% for database. Considering that the addition of a line buffer does not impact the processor's cycle time [Wils96] and machine performance is always increased, the rest of the results presented in this paper are for memory subsystems that include a line buffer.

4.3. Performance of DRAM Caches

Figure 7 presents the performance results for our dynamic superscalar processor with a 4 Mbyte DRAM on-chip data cache, a 16 Kbyte two-way-set-associative single-cycle cache created from the DRAM row buffers, and no 4 Mbyte SRAM off-chip cache. The hit time of the DRAM is varied from six to eight cycles to show how longer DRAM latencies will affect machine performance. Only the results for eight-way banked DRAM caches are presented because a 4 Mbyte cache is large and the DRAM's row buffers act as banks. The performance results for the banked DRAM cache without a line buffer are added to show how much difference a line buffer makes as the hit time of the DRAM is increased from six to eight cycles. With the database application, the performance of the banked DRAM cache with and without a line buffer is identical.

Compared to [Shim96] the hit time of the row buffers and DRAM are very optimistic. This is necessary to make a DRAM cache that could compare to on-chip SRAM caches. We find that if the row buffers have a two cycle hit time (not shown on graph), the performance of the DRAM cache does not perform well enough to merit implementation on the processor die.

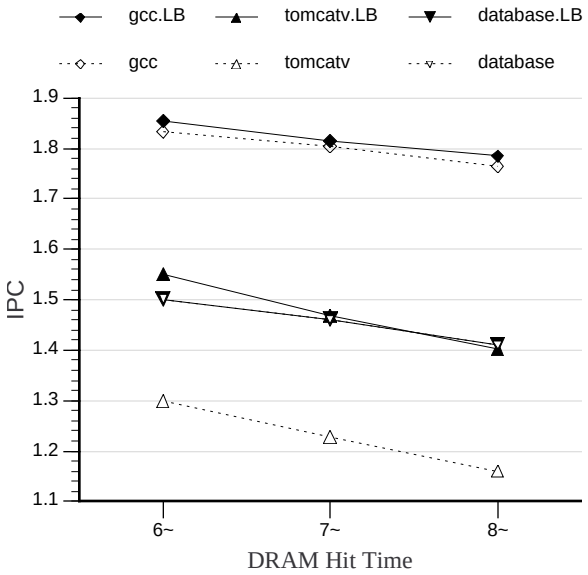

Figure 7: Multi-cycle 4 Mbyte DRAM cache with an eight-way banked 16 Kbyte row buffer cache and fixed processor cycle time.

Figure 7 shows that a line buffer provides a negligible improvement in the performance of gcc and database but drastically increases the performance of tomcatv. For an eight-way banked one cycle hit time cache, a line buffer is not really needed to reduce the access pressure on the cache ports. Therefore, we do not expect an improvement from the line buffer unless very small caches are used or there are a great number of conflict misses that can take advantage of the extra associativity provided by the line buffer. Database shows no performance increase due to the line buffer, but gcc gains more incremental performance from adding a line buffer to DRAM caches than to SRAM caches. This occurs because the use of 512 byte cache lines reduces processor performance by increasing the number of conflict misses and therefore increasing the importance of the extra associativity added by the line buffer. The line buffer increases the performance of tomcatv 18% by reducing the conflict misses that are caused both by the increase in line size, and the small size of the 16 Kbyte cache. With a line buffer, the performance cost of using the 16 Kbyte two-way-set-associative 512 byte line row buffer cache instead of an equivalent SRAM cache with 32 byte lines is 17%, 6%, and 6% for tomcatv, gcc, and database respectively.

Note that the decrease in processor performance due to increasing the hit time of the DRAM cache is not that significant because of the single-cycle hit time row buffer cache. The same amount of performance is lost with an increase in DRAM hit time of six to seven cycles as is lost for an increase in hit time from seven to eight cycles. On average for all nine benchmarks, the processor performance drops by 3% for each one cycle increase in the DRAM hit time.

4.4. Comparing Design Alternatives

To determine which on-chip memory organization provides the best IPC, we simulate the nine benchmarks with cache sizes ranging from 4 Kbytes to 1 Mbyte, and cache hit times of one to three processor cycles. As expected, the best performing organizations are single cycle access duplicate caches and eight-way banked caches with a line buffer inside the processor's load/store execution unit.

Figure 8 shows how the IPC of the processor increases with increasing cache size for gcc, tomcatv, database, and the average of the nine benchmarks. This is done for duplicate and eight-way banked pipelined caches of one to three cycles with a line buffer. The datapoint for the 4 Mbyte DRAM cache with a one cycle 16 Kbyte row buffer cache and a six cycle DRAM hit time is also shown. The curves for both gcc and database have the same characteristics as the curves for the average of the nine benchmarks. Tomcatv behaves slightly differently. For larger cache sizes, the eight-way banked cache just barely outperforms the duplicate cache. Large eight-way banked caches perform better because floating point applications, like tomcatv, are more likely than integer applications to take advantage of the extra cache ports by having more than two non-conflicting load instructions ready to access the cache in a single cycle. On the other hand, for a small cache tomcatv performs significantly better with a duplicate cache than an eight-way banked cache. The higher miss rate of the small cache causes cache lines to be replaced more frequently. For tomcatv, the banked cache tends to group cache accesses to different cache lines within a single cycle. This increases the rate of cache line replacement, lowering the cache's ability to take advantage of spatial locality and therefore increasing the miss rate. As the miss rate decreases with larger caches, the chance of replacing a cache line decreases, and the performance advantage of more cache ports allows the eight-way banked cache to outperform the duplicate cache.

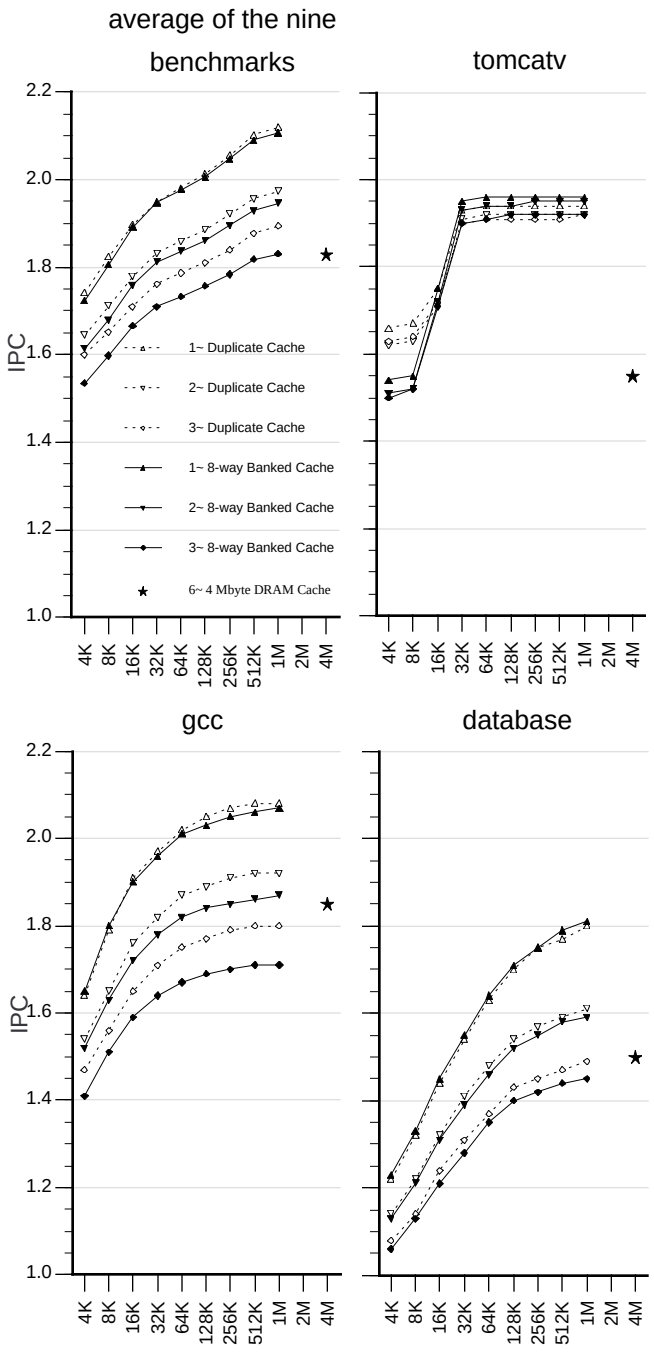

Figure 8: IPC of multi-cycle banked and duplicate caches with a line buffer and fixed processor cycle time.

In Figure 8 we see that on average the performance of a duplicate

cache is greater than that of the equivalent eight-way banked cache even though Figures 4 and 5 show us the opposite result. The change is due to the inclusion of a line buffer within the load/store execution unit of the processor. Earlier in this study we showed that without the line buffer, eight-way banked caches always outperform duplicate caches. Although an eight-way banked cache has the drawback of bank conflicts, the performance penalty is not as severe as being restricted to two cache ports. Adding a line buffer does not significantly reduce bank conflicts, but does reduce port contention; therefore, a duplicate cache with a line buffer can outperform an eight-way banked cache with a line buffer. Furthermore, the cache access time of an eight-way banked cache is always greater or equal to the cache access time of a duplicate cache of the same size. Consequently, on average a duplicate cache with a line buffer will outperform an eight-way banked cache with a line buffer. For this reason we only examine duplicate caches to find the best cache organization for a given processor cycle time.

The IPC of a six cycle hit time DRAM cache with a one cycle row buffer cache and a line buffer is also included in Figure 8. Even with the optimistic hit time simulated for the DRAM cache, on average the DRAM cache does not perform as well as a 16 Kbyte SRAM cache. The reasons for this are first, the 512 byte lines of the row buffer cache cause a significant degradation in processor performance due to conflict misses that is only partially alleviated by the use of a line buffer, and second, the 16 Kbyte row buffer cache is not large enough to make up for the six cycle latency of the DRAM. A DRAM cache can only compete with a SRAM cache if the row buffer cache is increased to 32 Kbytes and the performance degradation due to the use of 512 byte lines can be hidden.

Up to this point we have considered the processor's cycle time fixed and presented performance in terms of IPC results, whereas the real measure of performance is application execution time. To determine execution time, we use the cache access times shown in Figure 1 to find the maximum size duplicate SRAM cache that can be built with hit times of one, two, and three processor cycles for a given processor cycle time. We then use these cache sizes and cycle times to find the execution time of applications running on a processor with the corresponding dual-ported cache. However, we cannot use the IPC results shown in Figure 8 because they are from simulations of the different cache structures with a fixed level two cache hit time of ten cycles and a memory access time of sixty cycles. Different simulations using the correctly scaled level two cache hit time and memory access time are used in Figure 9 to present the execution time of our applications. Figure 9 shows how the execution times for gcc, tomcatv, database, and the average of all nine benchmarks vary with cache size and hit time for different processor cycle times. The curves for each benchmark are normalized to the execution time of the processor with a cycle time of 10 FO4 and a 32 KByte three-cycle pipelined cache.

Notice that the execution time curves for the duplicate caches have similar shapes for one and two cycle hit time caches. The curve for the three cycle cache is also comparable, but only the region from a 32 Kbyte cache to a 1 Mbyte cache is shown. The shape of the curves for gcc and database is the same as for the average of the nine benchmarks, but the slope of the curve for database is flatter than the curve for gcc. The slope of the database curve is flatter

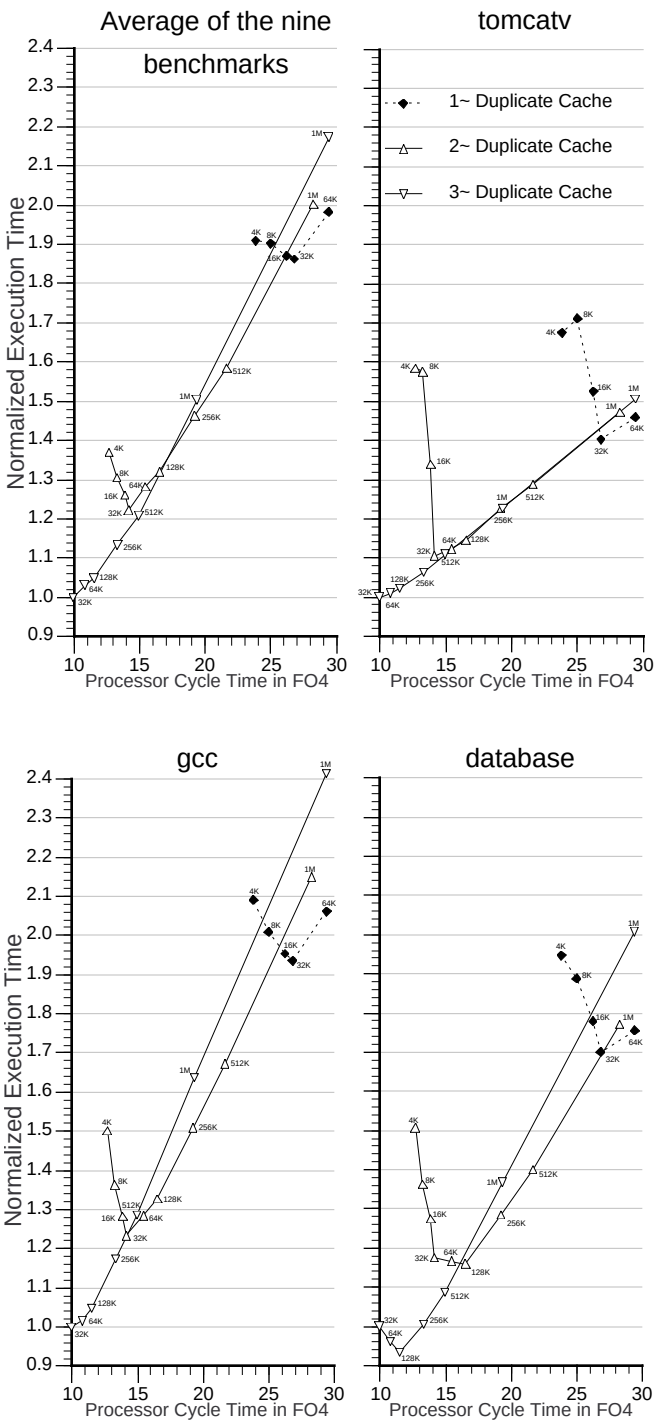

Figure 9: Normalized execution time for multi-cycle duplicate caches with a line buffer.

since the miss rates are greater; therefore the effect of increasing the frequency of the processor is less. In the case of tomcatv, we see very little difference between the performance of one to three cycle pipelined caches in the range of 32 Kbytes to 1 Mbyte because the IPC of tomcatv does not vary in this range of cache sizes, and tomcatv loses very little performance with increasing cache pipeline depth.

One thing to note from Figure 9 is that even with a fixed cache size, a given percentage reduction in processor cycle time results in a smaller percentage reduction in execution time. This is due to the fact that a significant portion of an application's execution time is spent in the memory system. For example, the performance of tomcatv only increases by a factor of 1.5, even though the processor cycle time is reduced by a factor of three. We find that tomcatv spends roughly half of its execution time in the memory system. Using a speedup of three in processor cycle time and fifty percent of the execution time spent in the processor, Amdahl's Law [Henn96] predicts an overall performance increase of 1.5 times.

The non-pipelined cache results of Figure 9 can be compared to the results of Jouppi and Wilton [Joup94] for single-level on-chip caches. They used cacti for the access times of their caches and a trace driven memory simulator to simulate the performance of 1 Kbyte to 256 Kbyte direct mapped blocking caches. We also use cacti for our cache access times, but we simulate a dynamic superscalar processor with two-way-set-associative non-blocking caches. The dynamic superscalar processor and the non-blocking caches hide part of the performance lost due to small cache sizes, and a two-way-set-associative cache will perform about as well as a direct-mapped cache of twice its size [Henn96]. These differences tend to increase the processor's performance for smaller size caches and decrease the incremental performance gained from doubling the cache size.

Our two studies overlap for gcc and tomcatv executed with single-cycle single-level caches varying in size from 4 Kbytes to 256 Kbytes. Jouppi and Wilton show that the best performance with a single level of cache is achieved for SPEC92 gcc with a 32 Kbyte direct mapped primary data cache. For SPEC95 gcc, we find that for an aggressive dynamic superscalar processor, the best performance for gcc is obtained with a two-way set-associative 32 Kbyte duplicate cache. In the case of SPEC92 tomcatv, Jouppi and Wilton found that only a 16 Kbyte data cache was required, but we find that a 32 Kbyte duplicate cache is needed to provide the best execution times. Larger set-associative caches are required due to the larger working set sizes of the SPEC95 benchmarks. Our caches are also multi-ported to support the additional cache bandwidth required by a dynamic superscalar processor.

The amount of cache pipelining that optimizes performance depends on both the working set size of the applications that are executed on the dynamic superscalar processor and the processor's cycle time. Executing with larger application working sets increases the cache miss rate, requiring a larger cache to minimize execution time. Refer to Figure 3 to see how the larger cache sizes available due to pipelining reduce miss rate. Decreasing the processor's cycle time makes it harder to build a cache large enough to minimize execution time without pipelining the cache. Therefore, the combination of a large working set size and a small processor cycle time require large pipelined caches to produce the best processor performance.

In Figure 9 we see that the best performing cache organization is indeed based upon both processor cycle time and working set size. If the processor's cycle time is large enough, a single-cycle 64 Kbyte duplicate cache will provide the best performance, but if the processor's cycle time is less than 25 FO4, a pipelined cache is always the best performer. In between these two extremes the best organization is dependent upon both cycle time and working set size. For gcc we see that if the processor can accommodate a single-cycle 16 Kbyte cache then the cache should not be pipelined; however database has a larger working set than gcc, so its results lead to a different conclusion. In fact, if we simulate only small applications with working sets that can fit in a 4 Kbyte cache, there is be no point in using pipelined caches unless the processor's frequency is so fast that a one cycle 4 Kbyte cache cannot be built. Only applications with large realistic working sets need the larger caches that are possible with pipelining.

From Figure 9 we see that a processor cycle time of 29 FO4 can accommodate a one cycle 64 Kbyte duplicate cache. Even with database's working set size, a 64 Kbyte single cycle cache outperforms any pipelined cache that can be built. For processor cycle times of less than 29 FO4, the larger pipelined cache maximizes processor performance. In the unlikely event that a dynamic superscalar processor can be built in 15 FO4, a 1 Mbyte three cycle cache provides the best performance, but a two cycle 32 Kbyte cache should be considered for its smaller die area at the cost of a small drop in performance. If a processor can be built with a cycle time of 10 FO4, then at least three cycles of pipelining are required.

Figure 9 shows the best cache size and pipelining necessary to minimize application execution time for a given processor cycle time. When the processor cycle time is not fixed, we find that on average a 32 Kbyte cache minimizes execution time for each cache pipeline depth. This shows that on average our application working sets fit well enough within a 32 Kbyte cache that increasing the cache size does not sufficiently decrease the cache miss rate to compensate for the resulting increase in cycle time.

5. Conclusion

In this study we have investigated a number of on-chip data cache organizations that include multi-ported caches, multi-cycle caches, line buffers, and DRAM caches. We have evaluated these cache organizations with simulations that include operating system references for nine realistic benchmarks with primary data cache sizes ranging from 4 Kbytes to 4 Mbytes. A dynamic superscalar processor with a lockup-free fully-pipelined cache was used to maximize the bandwidth demands on the cache. The processor cycle times were varied from a cycle time of 29 FO4, that can accommodate a 64 Kbyte one cycle single-ported primary data cache, to extremely fast processors that have a processor cycle time of only 10 FO4.

For a fixed cache size and processor cycle time, increasing the number of cache ports through cache banking and cache duplication increases processor performance, while increasing the amount of pipelining in the cache decreases processor performance. In the case of ideal cache ports, increasing the number of ports from one to two results in a 25% increase in processor performance. An increase in the number of cache ports from two to three achieves a 4% increase in performance and an increase from three to four ports increases performance by less than 1%. Duplicating the cache results in processor performance comparable to a cache with two ideal cache ports. Adding extra cache ports by banking the cache also increases processor performance, but much more slowly than adding

ideal cache ports. We used cacti to analyze an eight-way banked cache and demonstrated that banking the cache increases the hit time for small cache sizes. Larger caches, however, already use an internal banking structure. Looking at IPC and ignoring changes in cycle time, we found that a processor with a duplicate cache will always outperform a four-way banked cache but that an eight-way banked cache will always outperform a duplicate cache. For this reason we concentrate on eight-way banked caches and duplicate caches in this study. When pipelining is added to the cache with cache size and cycle time held constant, we see a 12-23% decrease in IPC per pipeline stage for integer applications but only a 3-9% decrease for floating point applications.

Since adding additional cache ports can be expensive both in die area and cache cycle time, a line buffer can be used to reduce the performance loss due to building fewer cache ports. The use of a line buffer with a duplicate cache improves processor performance by 3%, while a line buffer with an eight-way banked cache improves performance by only 0.5%. In fact, the performance improvement due to a line buffer used with a duplicate cache changed the results so that the IPC of a duplicate cache with a line buffer is on average always greater or equal to the IPC of an eight-way banked cache with a line buffer. In addition, the use of a line buffer with pipelined multi-ported caches decreases the performance drop due to pipelining by 28%-61% for integer applications and 50%-74% for floating point applications.

Combining cycle time, cache size, and cache structure to find the cache organization with the best performance on our nine benchmarks, we found that the use of a large pipelined duplicate cache with a line buffer almost always results in the best processor performance. Since the use of a line buffer always increases processor IPC without a corresponding increase in processor cycle time and the cycle time of a duplicate cache is always less than or equal to the cycle time of an eight-way banked cache, a processor performs better with a duplicate cache instead of an eight-way banked cache as long as a line buffer is included within the load/store execution unit of the processor. We also found that the use of an aggressive six cycle hit time 4 Mbyte DRAM cache with a one cycle eight-way banked 16 Kbyte two-way-set-associative primary row buffer cache and a line buffer, on average, provides less processor performance than an equivalent 16 Kbyte SRAM cache with a ten cycle hit time 4 Mbyte secondary cache.

To minimize application execution time, the pipeline depth of the cache must be determined by the processor's cycle time. For a processor with a slow cycle time of 29 FO4, a 64 Kbyte dual-ported single-cycle cache provides the best processor performance. On the other hand, for processor cycle times between 29 FO4 and 24 FO4, a two cycle cache delivers the best performance if there is enough room on the processor die for the larger two cycle cache. For processor cycle times of less than 24 FO4, the cache must be pipelined since the processor cannot support a single-cycle non-pipelined cache of even 4 KBytes.

Acknowledgments

We would like to thank Mendel Rosenblum, Steve Herrod, Edouard Bugnion, and Robert Bosch for their help with SimOS, Mark Horowitz for his help with cache design, and the reviewers for their insightful comments. This work was supported by ARPA contract DABT63-94-C-0054.

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.

[Benn95] James Bennett and Mike Flynn, "Performance Factors for Superscalar Processors", Technical Report CSL-TR-95-661, Computer Systems Laboratory, Stanford University, Feb. 1995.

[Bowh95] William J. Bowhill, et al, "Circuit Implementation of a 300-MHz 64-bit Second-generation CMOS Alpha CPU", *Digital Technical Journal*, Special 10th Anniversary Issue, Vol. 7, No. 1, 1995, pp. 100-118.

[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", *Proceedings of the 21st Annual International Symposium on Computer Architecture*, 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", *Proceedings of the 21st Annual International Symposium on Computer Architecture*, Chicago, Illinois, April 18-21, 1994, pp. 60-70.

[Edmo95] John H. Edmondson, et al, "Internal Organization of the Alpha 21164 a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor", *Digital Technical Journal*, Special 10th Anniversary Issue, Vol. 7, No. 1, 1995, pp. 119-135.

[Fark94] Keith I. Farkas and Norman P. Jouppi, "Complexity/Performance Tradeoffs with Non-Blocking Loads", *Proceedings of the 21st Annual International Symposium on Computer Architecture*, 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", *Proceedings of the 21st Annual International Symposium on Computer Architecture*, 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.

[Gray93] Jim Gray, Ed., "The Benchmark Handbook for Database and Transaction Processing Systems", Morgan Kaufmann Publishers, 1993.

[Gwen94] Linley Gwennap, "MIPS R10000 Uses Decoupled Architecture", *Microprocessor Report*, Volume 8, Number 14, October 24, 1994, pp. 18-22.

[Henn96] John L. Hennessy and David A. Patterson, "Computer Architecture a Quantitative Approach", Morgan Kaufmann Publishers, Inc, Second Edition, 1996.

[Horo92] M. Horowitz, S. Przybylski, and M. D. Smith, "Tutorial on Recent Trends in Processor Design: Reclimbing the Complexity Curve", *Western Institute of Computer Science*, Stanford University, 1992.

[Horo96] M. Horowitz, "High Frequency Clock Distribution", High Frequency Digital Logic Design and Clocking Strategies, *1996 Symposium on VLSI Circuits*, June 13-15, 1996.

[Hunt95] Doug Hunt, “Advanced Performance Features of the 64-bit PA-8000”, *Digest of papers COMPCON ‘95*, IEEE Computer Society Press, San Francisco, CA, March 5-9, 1995, pp. 123-132.

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

[Joup90] Norman P. Jouppi, “Improving direct-mapped cache performance by the addition of a small fully-associative cache and prfetch buffers”, *Proceedings of the 17th Annual Int’l Symposium on Computer Architecture*, IEEE Computer Society Press, Seattle, May 28-31, 1990, pp. 364-373.

[Joup93] Norman P. Jouppi, “Cache Write Policies and Performance”, *Proceedings of the 20th Annual International Symposium on Computer Architecture*, San Diego, California, May 16-19, 1993.

[Joup94] Norman P. Jouppi and Steven J. E. Wilton, “Tradeoffs in Two-Level On-Chip Caching”, *Proceedings of the 21st Annual International Symposium on Computer Architecture*, Chicago, Illinois, April 18-21, 1994, pp. 34-45.

[Krof81] David Kroft, “Lockup-Free Instruction Fetch/Prefetch Cache Organization”, *Proceedings of the 8th Annual International Symposium on Computer Architecture*, 1993 pp. 81-87.

[Kell96] Jim Keller, “The 21264: A Superscalar Alpha Processor with Out-of-Order Execution”, *Microprocessor Forum*, October 22-23, 1996.

[Kusk94] Jeff Kuskin, et al, “The Stanford FLASH multiprocessor”, *Proceedings of the 21st International Symposium on Computer Architecture*, pp. 302-313, April 1994.

[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.

[Naff96] Samuel Naffziger, “SP 22.5: A Sub-Nanosecond 0.5 um 64b Adder Design”, *1996 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, Volume 39, February 1996, pp. 362, 363.

[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”, *Proceedings of the 19th Annual International Symposium on Computer Architecture*, 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.

[Rose95] Mendel Rosenblum, Edouard Bugnion, Stephen Alan Herrod, Emmett Witchel, and Anoop Gupta, “The Impact of Architectural Trends on Operating System Performance”, *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, Colorado, Dec. 3-6, 1995.

[Rose95b] Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, and Anoop Gupta, “Complete Computer System Simulation: The SimOS Approach”, *IEEE Parallel and Distrubuted Technology*, Volume 3, Number 4, Fall 1995.

[Saul96] Ashley Saulsbury, Fong Pong, Andreas Nowatzyk, “Missing the Memory Wall: The Case for Processor/Memory Integration”, *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 22-24, 1996, pp. 90-101.

[Shim96] Toru Shimizu, et al, “A Multimedia 32b RISC Microprocessor with 16Mb DRAM”, *1996 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, Volume 39, IEEE Catalog Number 96CH35889, February 1996, pp. 216, 217.

[Sohi91] Gurindar S. Sohi and Manoj Franklin, “High-Bandwidth Data Memory Systems for Superscalar Processors”, *ASPLOS-IV*, Santa Clara, CA, April 8-11, 1991.

[SPEC95] SPEC, “SPEC Benchmark Specifications - 101.tomcatv”, SPEC95 benchmarks release, 1995.

[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, November 1993.

[Wilk95] Maurice V. Wilkes, “The Memory Wall and the CMOS End-Point”, *ACM Computer Architecture News*, Vol. 23, No. 4, September 1995, pp. 4-6

[Wils95] Kenneth M. Wilson and Kunle Olukotun, “High Performance Cache Architectures to Support Dynamic Superscalar Microprocessors”, Technical Report CSL-TR-95-682, Computer Systems Laboratory, Stanford University, June 1995.

[Wils96] Kenneth M. Wilson, Kunle Olukotun, and Mendel Rosenblum, “Increasing Cache Port Efficiency for Dynamic Superscalar Microprocessors”, *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 22-24, 1996, pp. 147-157.

[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, 1994.

[Wilt96] Steven J. E. Wilton and Norman P. Jouppi, “CACTI: An Enhanced Access and Cycle Time Model”, *IEEE Journal of Solid-State Circuits*, Volume 31, Number 5, May 1996, pp. 677-688.

[Witc96] Emmett Witchel and Mendel Rosenblum, “Embra: Fast and Flexible Machine Simulation”, *Proceedings of the ACM SIGMETRICS '96: Conference on Measurement and Modeling of Computer Systems*, Philadelphia, PA, 1996.

[Wulf95] Wm. A. Wulf, Sally A. McKee, “Hitting the Memory Wall: Implications of the Obvious”, *ACM Computer Architecture News*, Vol. 23, No. 1, March 1995, pp. 20-24.

[Yeag96] Kenneth C. Yeager, “MIPS R10000”, *IEEE Micro*, Vol. 16, No 2, April 1996, pp. 28-40.