The Performance Impact of Vector Processor Caches*

Jeffrey D. Gee
Alan Jay Smith

Department of Electrical Engineering and Computer Science
Computer Science Division
University of California
Berkeley, CA 94720

ABSTRACT

Cache memories have not been used for vector supercomputers, as far as we know, because of a belief that program behavior in relevant workloads was such as to preclude efficient cache operation. It has been possible to make efficient use of such machines by carefully programming around the resulting long memory delays, although unmodified, "dusty-deck" code usually performs poorly. In related research, we have found that hit ratios are high for large caches in processors with vector workloads. In this paper, we address the specific issue of the direct effect of cache memory on vector processor performance.

The issue in processor design is machine performance, of which the hit ratio of the cache is only one determinant. In this paper, we simulate three vector processors, the designs for which are derived from expected technology changes applied to the Ardent Titan. Our simulator is an accurate timing model incorporating the necessary aspects of the design of the cache and memory system. We find that current trends in memory and processor performance will lead to increasingly severe memory speed and bandwidth limitations. Either of two designs using large cache memories (2MB, 4MB) on the average double processor performance relative to the design without a cache. Hit ratios for almost all of the programs used for trace driven simulation, drawn from real Ardent workloads, are over 99%. Based on the work presented here and elsewhere, we recommend that future supercomputers incorporate cache memories.

December 4, 1990

* The material presented here is based on research supported in part by the National Science Foundation under grant MIP-8713274, by NASA under grant NCC2-550, by the State of California under the MICRO program, and by Philips Laboratories/Signetics, the International Business Machines Corporation, Digital Equipment Corporation, and Apple Computer Corporation.

1. Introduction

Cache memories [Smit82] are a well known and important part of CPUs. At first, cache memories, for reasons of cost, were found on only the largest mainframes. They are now found on all machines except low-end microprocessor based machines, and (curiously) supercomputers. The only vector supercomputer with a cache is the IBM 3090-VF; the machines from Cray, NEC, Fujitsu and Hitachi do not use cache for vector references. We believe that the reason for this is an (incorrect, at least at this time) assumption that reference patterns to data in such machines would yield poor cache performance. Such a reference pattern could occur for one of several reasons: (a) Vector strides could exceed the length of a cache line, thus causing a miss for every memory reference. (b) Scatter/gather operations could have no locality, thus also causing a miss for every memory reference. (c) Supercomputer workloads tend to be superlarge, and might have working sets much larger than feasible cache sizes, thus yielding very high data miss ratios.

Independent of the accuracy of the beliefs noted above, it has been possible to obtain good performance from supercomputers by careful programming. By heavy use of vectorization and careful coding to allow long delays between the issuance of load instructions and the use of the requested data, many programs could be written to perform well on supercomputers. Nevertheless, there are three severe weaknesses to the current, no cache, approach: (a) The cost of coding to match the machine architecture is enormous, and is justified only by the super-high prices of supercomputers; the advent of very high performance RISC-type 32- and 64-bit microprocessors is already creating significant price pressure on this class of machines. (b) Programs which have not been specially coded run quite poorly on supercomputers. The IBM RS/6000 is faster than some models of the Cray on non-vectorized workloads, while costing around 1% as much. (c) Processor performance has been increasing much more rapidly than memory speed [HePa90] and should continue to do so. This trend is illustrated in Table 1, consisting of data primarily from [Neve89]; missing numbers are not public, although they could be determined experimentally. As may be seen, memory latencies, in terms of processor cycles, have been increasing over time. Thus the difficulty of programming around the increasingly long memory latencies will increase.

Caches can also play a role in the so-called mini-supercomputer and personal supercomputer market. Machines of these types (Stardent, Convex, Alliant) offer a respectable fraction of supercomputer performance at a small fraction of the cost. For example, the first generation Ardent Titan was introduced in 1988 with 16 Mflops peak performance, a 16 Mhz MIPS R2000 scalar processor, an 8 Mhz vector unit, and two 16 Mhz system busses providing 256 Mbytes/second peak memory bandwidth. A follow on machine, the Stardent ST3000, was introduced in 1989 with double the processing power of the Titan. The 32 Mflop ST3000 has a 32 MHz MIPS R3000 and a 16 MHz vector unit, but no corresponding improvement in bus speed, memory bandwidth, or memory latency. Vector caches are an appealing means to match improving processor performance without a large increase in cost.

Another factor supporting the use of vector caches is the increasing use of multiprocessing in vector machines. All current Cray supercomputers and at least one new Japanese supercomputer can be configured with multiple processors, all sharing a global memory. Alliant, Convex, and Stardent also market multiprocessor systems. In shared memory systems, competing vector streams cause conflicts for memory banks and memory access paths. These conflicts can reduce memory system performance substantially [Cheu84, Cala88]. Contention for main memory is dramatically reduced if each processor has

Machine Characteristics					
Computer	Year	Clock Cycle (ns)	Peak MFLOPS	Mem. Latency (cycles)	Max CPUS
Cray 1	1976	12.5	160	11	1
Cray X-MP	1984	8.5	233	14	4
Fujitsu VP-200	1984	7.0	533	31-33	1
Hitachi S-810/20	1984	14.0	630		1
NEC SX-2	1984	6.0	1300		1
Cray 2	1985	4.1	488	35-50	4
Fujitsu VP-400	1986	7.0	1700	31-33	1
Cray Y-MP	1988	6.0	333	17	8
Hitachi S-820/80	1988	4.0	3000		1
Fujitsu VP-2600	1989	4.0	4000	>30	1
NEC SX-3	1989	2.9	5500	60-70	4
Cray 3	-	2.0	1000		16

Table 1: Machine Characteristics

This table contains some performance characteristics for a number of vector machines. Cycle times are in nanoseconds, and memory latencies are in processor cycles. Peak MFLOPS refers to the peak performance of a single processor machine. Max CPUS refers to the maximum number of processors in a multiprocessor configuration.

its own private vector cache.

Interest in vector caches has resulted in at least three recent studies ([So88a], [So88b], [Gee90]). These studies examined locality in vector references and measured cache miss ratios over a range of applications. Miss ratios are typically less than 2% for caches larger than 128 Kbytes, and fall below 1% for cache sizes above 1 Mbyte. The extensive study in [Gee90] showed that there was substantial locality in vector workload reference patterns, and gave substantial evidence that caches would aid the performance of vector machines.

Much more convincing than miss ratio figures are the results of CPU timing simulators. In our research we have developed timing simulators for hypothetical vector cache machines, enabling us to estimate the performance impact of vector caches. The hypothetical machines are based on the Ardent Titan architecture, and assume that processor speeds will continue to increase dramatically, with little or no increase in main memory speeds. We developed three simulators, each containing an identical timing model for processors and main memory. Two of the simulators contain different vector cache models, while the third assumes that all vector data is referenced directly from memory. All simulators accurately measure execution time, as well as time independent metrics such as miss ratios and bus traffic ratios. Long (500M reference) traces from a large number of real applications were used to drive the simulator. The programs came from areas such as computational fluid dynamics, linear analysis, and computational chemistry. Our results indicate that applications run up to five times faster with a vector cache, and twice as fast on average.

1.1. Previous Work

There are only a limited number of previous studies of caches for vector workloads. Two papers by So and Zecca [So88a,88b] analyzed traces for the IBM 3090 VF, and observed that low miss ratios (less than 4%) could be obtained, but with the use of larger cache and line sizes than would be needed for a scalar workload. In [Gee90], a much larger sample of Cray X-MP and Ardent Titan traces were collected to explore reference locality and cache behavior in vector applications; many of these traces are used for the results presented in this paper. Temporal and spatial locality were characterized by measuring miss ratios, working set sizes, interreference intervals, and changes in cache miss ratio with increasing block size. The study found vector references to contain more spatial locality, but less temporal locality than scalar references. Cache miss ratios fell below 1% for caches as small as 128 Kbytes, and below 0.5% for a 1 Mbyte cache.

[Gee90] also estimated the effect of caches on time performance by measuring average access delays over a range of memory system parameters. For cache machines, the average access delay is the product of the cache miss ratio and the time to service a cache miss. For machines without cache, the average access delay is measured by assigning delays to each instruction branch, to the first element of each vector reference, and to each scalar reference. Using a simple pipelined model of execution, [Gee90] estimated that a large vector cache can improve the time performance of an application by at least 50%, and possibly more, depending on the memory latency in the system.

[Abu86] provided timing results for the Alliant FX/8, a vector multiprocessor which caches vector data. Vectorized kernels from the Los Alamos benchmarks were timed, with vector lengths ranging from 1 to 100,000. Observed performance improvements showed speedups of 1.35 to 3.67 over a system with no cache. The kernels cease to fit in the cache for vector lengths beyond a few thousand elements, however, and performance drops quickly once these lengths are exceeded. The results, while interesting, mislead somewhat by suggesting that caches lose effectiveness once the problem size exceeds the cache size. This is not the case in real environments, as large numerical applications can be coded to execute efficiently in smaller memories [McKe69], [Triv77], [Agar86].

1.2. Organization

The remainder of this paper is divided into the following sections. Section 2 discusses the simulation environment. Section 3 discusses the machine and vector cache models. Section 4 describes the benchmarks and presents some of their characteristics. Simulation results are presented in section 5. Section 6 presents the conclusions of our work.

2. Simulation Environment

2.1. Simulation Platform: The Ardent Titan

The timing simulator used for this research is developed for and runs on the Ardent Titan [Died88]. The Titan is a graphics supercomputer which combines the scalar performance of a MIPS R2000 general purpose processor and a custom floating point vector unit built partially with commodity parts. The machine architecture is a combination of MIPS scalar instructions and Ardent-defined floating point operations. The MIPS processor handles instruction fetch and executes integer operations, while the floating point unit executes both scalar and vector floating point operations. Floating point unit instructions are specified in the instruction stream by a MIPS load or store to reserved locations in the address

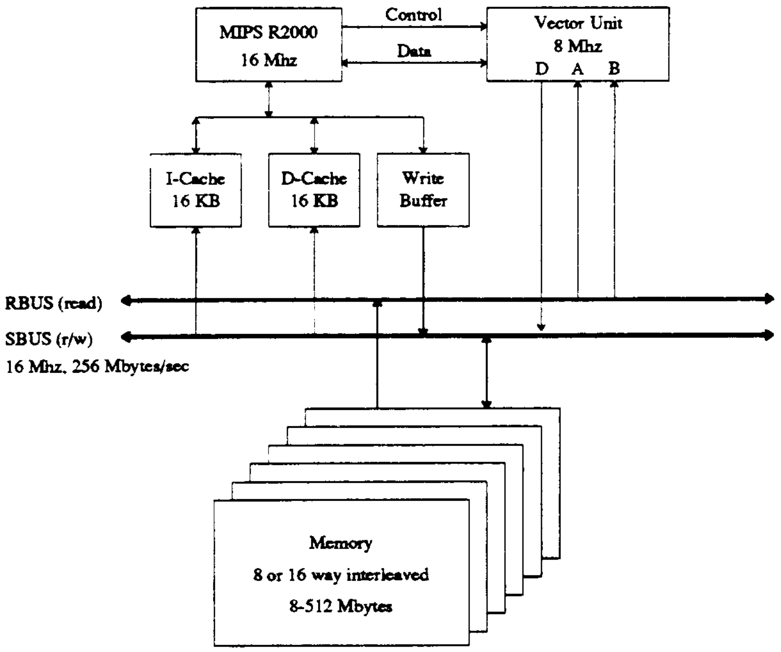


Figure 1: Ardent Titan block diagram

This figure shows a block diagram of the Ardent Titan. The Titan consists of a 16 MHz MIPS R2000, an 8 MHz custom vector unit, and a 256 MByte/sec memory bus. Instruction and data caches for MIPS references are provided, while the vector unit references all data from memory through one store and two load pipelines.

space. Each reserved address corresponds to a different floating point operation. A block diagram of the Titan is shown in figure 1.

Vector Operation Timings	
Operation	Clock Cycles
FP add,sub	6 + VL
FP mul	6 + VL
FP single div	7 + 4 * VL
FP double div	7 + 5 * VL
load	8 + VL
store	2 + VL

Table 2: Ardent Titan Operation Timings (VL: vector length)

In the Titan, the MIPS R2000 integer unit operates at 16 Mhz, and the floating point vector unit operates at 8 Mhz with a peak performance of 16 Mflops. Timings for some typical floating point vector operations are listed in table 2. The MIPS processor fetches instructions and integer data through separate 16 Kbyte instruction and data caches. Each cache can be accessed in a single cycle, and load data is available to the MIPS processor after a one cycle delay. The integer data cache is a write through cache, with a four deep write buffer to keep write stalls infrequent. The vector unit references all data from memory. It has three independent memory pipelines for scalar and vector floating point references: two 8 Mhz pipelines for loads, and one 16 Mhz pipeline for stores. The total memory bandwidth is 256 Mbytes/second, available from one 64 bit read only bus (RBus) and one 64 bit read/write bus (SBus), both of which are clocked at 16 Mhz. The memory system can be configured with 8 to 512 Mbytes of memory, interleaved 8 or 16 ways.

2.2. Simulation Process

The simulation process is as follows. A benchmark is compiled, and a postloader translates the resulting object code into a profiled version, inserting calls to simulation entry points before each MIPS load, store, and branch. Simulator calls inserted in front of MIPS loads and stores pass the address and data associated with that load or store. If the address corresponds to an Ardent-defined vector instruction, the simulator maintains sufficient state to reconstruct and simulate the vector operation.

The simulator is implemented in Unix System V shared libraries, which the operating system attaches to the profiled code at runtime. The profiled code is then executed on an Ardent Titan. Each call to the simulator specifies an event, either the completion of a basic block, a MIPS load or store, or a vector unit operation. The simulator contains a timing model for the machine, and updates internal cycle counts based on the particular event. At the end of the simulation run, cycle counts and cache metrics are deposited to an output file.

The timing simulator accurately models the entire vector unit, vector cache, and memory system. Memory bank, vector register, and bus conflicts inherent to multiple pipeline processors are handled precisely. A scoreboard resolves data dependencies and correctly serializes operations which require common vector registers. Modeling of the MIPS scalar processor is approximated by incrementing the clock for each executed MIPS instruction. For efficiency, details of the MIPS/Integer cache part of the system are not accurately simulated, since they are common to all three models studied. In particular, memory activity by the integer processor, including integer cache misses, is not modeled.

2.3. Limitations and Justification

Although timing results are preferable to miss ratios in assessing the performance impact of cache designs, there are inherent limitations to timing simulation. First, timing simulators tend to be slow, and our simulator is complicated by fact that multiple vector pipelines and resource conflicts must be modeled exactly. This limits the design space that can be explored, as well as the amount of real execution time that can be simulated. Programs with our simulator attached execute approximately 1000 times slower than unprofiled programs, limiting our simulations to a few minutes of real time. In addition, the results produced are highly dependent on the machine architecture, memory system, and timing parameters characteristic of one particular machine. Reported results, while highly accurate for that machine, may not apply to machines built on a different architecture or technology.

One limitation of our simulator is that since the simulator and tracer run as coroutines, the simulator runs only on Ardent Titans, and not on any other machine. Since we no longer have access to an Ardent Titan, we are unable to further extend the results that we report here.

3. Simulation Models

In this section, we describe the three models studied. We first present the common architecture of the machines, which is essentially the Ardent Titan architecture, as described earlier, and shown in figure 1. The timing parameters of the machine are based partially on technological trends and partially on targeted performance levels for a future generation machine. Two of the models contain vector caches. The third model has no cache and was developed mainly for comparison purposes. The vector caches utilize different techniques to achieve the high cache bandwidth critical for multiple pipeline architectures. One cache is interleaved with a crossbar switch connecting interleaves to memory access pipelines. The other cache consists of several independent caches, each assigned to a different memory access pipeline, with consistency maintained in hardware. Both caches are direct-mapped with a block size of 64 bytes.

3.1. Machine Architecture

As noted and described above, the core of our three models is an Ardent Titan. The major design decisions are (1) determining memory and operation latencies, in clock cycles, and (2) selecting clock rates for the integer unit, vector unit, and memory bus. Since this work is based on technological upgrades to the Ardent Titan and its successor (the Stardent ST3000), we assume that pipeline organizations remain the same, i.e. latencies for floating point operations are unchanged. Similarly, the access time to main memory, in bus cycles, also remains the same as in the Titan.

We determined cycle times by examining current trends in technology. The original Ardent Titan is a 16 Mflops peak machine, with a 16 MHz integer processor, an 8 MHz vector unit, and a 16 MHz, 256 Mbyte/second bus. The Stardent ST3000, a successor to the Titan, is a 32 Mflops peak machine, with a 32 MHz integer processor, a 16 MHz vector unit, and the same 16 MHz, 256 Mbyte/second bus. At the time of this study (summer 1989), integer unit clock speeds were expected to double again to 64 MHz (60 MHz MIPS R6000 processors are available in 1990), with little, if any, decrease in memory access times. It was also believed that a vector unit could be implemented at integer unit speeds, unlike earlier Ardent machines, which have vector units running at half the integer unit clock rate. These designs were explicitly based on design alternatives under consideration at Ardent and are thus considered to be highly realistic.

Our next generation machine model thus has the following timing characteristics. The integer and vector units both run at 64 MHz, yielding a peak performance of 128 Mflops. The memory latency is four processor cycles plus nine bus cycles, identical to earlier Ardent and Stardent machines. The bus rate and aggregate bus bandwidth are 16 MHz and 256 Mbytes/second, four times slower than processor clock rate, and unchanged from previous machines.

3.2. No Cache Model

The base configuration is identical to figure 1, except for the improvement in integer and vector unit clock rates. With the slow bus and main memory, this model essentially evaluates performance in a memory limited environment. The three memory access pipelines arbitrate for the read-only R bus and writable S bus. Writes have priority for the S bus, while the read that issued earliest has priority for the R bus. The bus bandwidth and clock rate of 256 Mbytes/second and 16 Mhz are both low for a 128 Mflops machine. Interleave conflicts at main memory are not modeled. The latency for a memory access is four processor plus nine bus cycles, for a total of 40 processor cycles.

3.3. Interleaved Cache Model

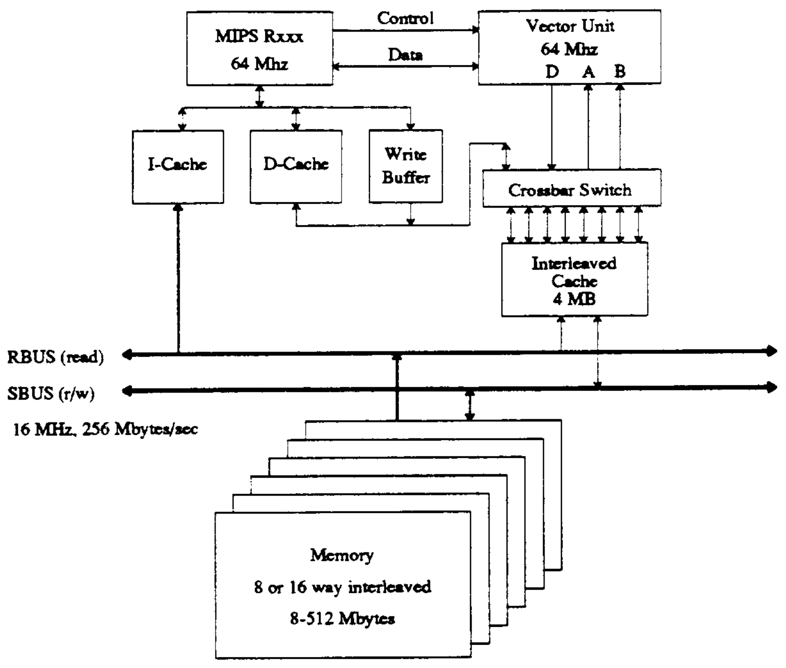


Figure 2: interleaved cache model

This figure shows the interleaved cache model, which consists of a 4 MByte, eight way interleaved cache with a two cycle access time. A crossbar switch connects the cache with the three vector access pipelines.

Because of the high memory bandwidth requirements of vector machines, arising from several data access pipelines running in parallel, a traditional, single-ported cache architecture is a bottleneck. One solution, as shown in figure 2, is to borrow from existing memory system techniques and interleave the cache N-ways; this increases the potential bandwidth by a factor of N. It also allows multiple memory access ports to run in parallel, given a suitable interconnect (e.g. a crossbar switch) between the ports and the separate cache interleaves.

In our *interleaved cache* model, the cache size is four Mbytes and the bank cycle time is two cycles. This was considered to be the maximum feasible cache size for a next generation Stardent machine. The two cycle access time includes transit time through the cache interconnect. For a given bank cycle time, the degree of interleaving is generally chosen to match bandwidth needs. We interleaved the cache eight ways, yielding a total cache bandwidth of 2 Gbytes/sec, sufficient for all three memory pipelines running at the full 64 MHz rate. A crossbar switch connects each interleave to each memory access pipeline.

The cache is interleaved by eight-byte word. The cache block size is 64 bytes and consists of a slice across all eight interleaves. In the simulator an arbitration scheme based on operation issue time is used to resolve interleave conflicts, with a two cycle delay between accesses to the same bank.

As shown in figure 2, the interleaved vector cache is distinct from the MIPS integer caches. There are several reasons why we did not simply eliminate the MIPS data cache and have the MIPS unit reference data directly from the vector cache. This would reduce the amount of cache bandwidth available to the vector pipelines, and delay either the integer unit or the vector unit in the case of conflicts. Conflict levels could be very high, as both the vector and integer units make heavy use of the cache, at least in spurts.

Instead of unifying the vector and integer caches, a better option is to use the vector cache as a large secondary cache for MIPS data references. The MIPS processor only accesses the vector cache on integer cache misses and write throughs, taking up only a small fraction of vector cache bandwidth. Cache consistency is easily maintained by making the contents of the MIPS cache a strict subset of the vector cache. Writes by the MIPS processor update both the MIPS and vector cache, while vector unit writes invalidate blocks from the MIPS cache.

On an interleaved cache miss, all interleaves are temporarily blocked. Each word retrieved from memory fills one of the interleaves, and frees that interleave to process additional cache references. The interleave where the miss occurred is serviced first, and the remaining interleaves are serviced in wraparound fashion at the rate of one interleave per bus cycle. The write policy in the interleaved cache is copy back, and copy backs are assumed to take zero time. This assumes that some sort of write buffering mechanism exists, i.e. a buffer containing several cache lines. Attributing no time to copy backs is reasonable, as replaced lines can usually be moved into the buffer while the processor is waiting for the line which caused the miss [Smit79,82]. We also assume that there is no bus or memory queuing when servicing a cache miss. Measured bus transfer ratios (presented later in this paper) are low, thus memory and busses should be free to service the occasional cache miss.

3.4. Partitioned Cache Model

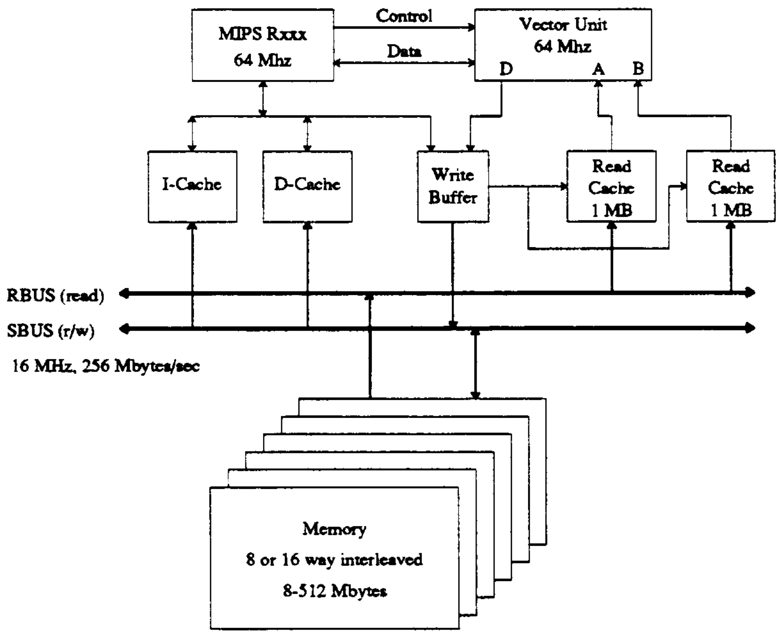


Figure 3: partitioned cache model

This figure shows the partitioned cache model, which consists of two 1 Mbyte read only caches and a deep write buffer. The read only caches are connected to the two vector read pipelines, and can be accessed in one cycle. Access time for the write buffer is also a single cycle. Consistency is maintained via hardware updates; each write to the buffer stalls both read caches for one cycle.

In addition to the interleaved design, it was considered desirable to consider at least one other cache alternative. The constraints of developing a low latency, high bandwidth cache model limits the number of feasible alternatives. For a machine with N memory ports, one could conceivably fill the requirements by (1) building N large caches, (2) building a single cache that is N times faster than each memory port, or (3) partitioning a large cache into N separate, smaller caches. The first and second options are impractical, the first due to the vast amount of cache required, and the second due to the excessive cost of building a cache N times faster than processor speed. The third option, which we explore, operates by *partitioning* a large cache into separate, smaller caches, each of which is assigned to a different memory port. The multiple caches provide sufficient bandwidth for all memory access ports.

The partitioned cache model is shown in figure 3. The two read pipelines are connected to separate read-only caches. The write pipeline feeds a write buffer common to both integer and vector units. We did not implement a write-only cache because such a cache made little sense. The write-only cache simply buffers writes, which we felt could be done more efficiently with a very deep write buffer. The buffer uses a write through mechanism to channel writes between the vector unit and main memory.

The combined size of the two caches shown is only 2MB, not the 4MB used for the interleaved design. This choice stems from an original plan (not followed) to add a third read port to the cache, with an additional 1MB cache, thus yielding a total of 3MB, close to the original four. In retrospect, we would choose to also study a design with two 2MB caches, but the facilities of Ardent Computer Corporation, now the West Coast branch of Stardent Computer Corporation, are being dismantled. We no longer have the facilities to do new simulations or repeat old ones. As will be seen below, we have been careful in our analysis to separate out the effect of cache capacity differences from other changes between the two models.

Each one Mbyte read cache is direct mapped with a 64 byte line size. As previously mentioned, the write through buffer never stalls. The cycle time of both read caches and the write buffer is one processor cycle, allowing each memory access pipeline to potentially complete one reference per cycle.

The read caches are independent, and duplicate copies of data may exist in both caches. In order to maintain consistency, a write through, write update protocol is used, by which all writes sent to the write buffer (for transmission to main memory), are also used to update any corresponding data found in the read caches. These updates stall any pending read accesses until the next cycle. Initial analysis found that updates are necessary for virtually all writes, thus the simulator assumes the worst case and blocks all read caches for one cycle when a write occurs.

Maintaining consistency between the integer and vector caches is quite simple. By using a shared write buffer, we guarantee that integer unit writes are seen by both the integer and vector cache. Using write through at all levels of the cache hierarchy eliminates the need for the integer to be a strict subset of the vector cache. To maintain consistency, vector unit writes must only invalidate the integer cache if the block is present there.

The miss penalty in the read caches is again four processor plus nine bus cycles. Valid bits are not maintained for each word within a block. Instead, the entire block is assumed valid after the first word in the block is fetched. Timing results for the partitioned cache model will thus tend to be slightly optimistic, as data assumed valid at a certain time may not actually be available until several cycles later.

4. Workload

Choosing the benchmark set is one the crucial steps in any performance study. This is particularly true for this work, as the performance of a vector cache is considerably more application and algorithm sensitive than the performance of a scalar cache. Our goal in the selection process has been to gather *real* applications which are typically used in a vector processing environment. We sought out large real programs, to stress the cache, and selected them across as wide a range of applications as possible. The applications gathered from real workloads at Ardent Computer include linear analysis programs such as *abaqus* and *ansys*, the computational chemistry module *born*, computational fluid dynamics programs such as *arc3d*, *dyna2d*, *flo57*, *flo82*, and *wake*, linear equation solvers *linpack*, *linpeak*, and *lapack*, and the molecular orbital applications *ampac*, *mopac*, and *nekton*.

Table 3 displays some vector unit characteristics of the programs. The memory references listed in the table do not include instruction and data references made by the MIPS integer unit. All statistics were gathered from the first 500 million vector unit memory references of each application, which, due to time constraints, was the stopping point for each simulation.

A column in table 3 lists the average number of memory references per floating point operation; the overall average is 1.4 memory references per floating point operation. A simple calculation shows that these memory references will be the performance bottleneck in future machines. The Stardent ST3000 has a maximum bus bandwidth of 32 million word references per second on the R and S busses. Using an average of 1.4 references per floating point operation, we calculate a peak in-memory performance of 21 Mflops for the ST3000. This is considerably less than the 32 Mflops potential peak performance of the machine. In our hypothetical machine, the vector unit is four times faster than the ST3000 vector unit, providing a theoretical peak performance of 128 Mflops. However, with memory and bus performance unchanged, the maximum performance executing from memory remains at 21 Mflops. Without some improvement in memory access rates, it is clear that vector unit memory references will be the performance bottleneck.

The working set sizes in table 3 are fairly large, on the order of one megabyte for a time quantum of one million references; see [Gee90] for a detailed study of vector reference behavior. We note that the working sets of many applications fit into our two proposed cache models. This is not a coincidence, as our choice of cache size is partially based on the sizes of typical applications, and partially based on the cost of the cache.

Table 4 is a breakdown of vector unit memory references into scalar reads, scalar writes, vector reads, and vector writes. As the Titan vector unit executes all floating point operations, scalar floating point references occasionally occur. The scalar categories also include *scatter/gather* references, which are vector references without a constant *stride*, or distance, between elements. They are included in the scalar categories because the Titan implements scatter/gather references as a series of scalar references. The programs are heavily vectorized, as over 80 percent of all floating point references are made in vector mode. The ratio of reads to writes is approximately two to one for both scalar and vector references, similar to results found in [Smit85].

Benchmark Characteristics						
Program	Dataset	Flops (millions)	Mem Refs (millions)	Refs/Flop	Data Space (Mbytes)	Working Set (Mbytes)
abaqus	t2-4-6	329.0	500.0	1.52	11.7	0.3
abaqus	t3-4-6	335.0	500.0	1.49	13.2	0.3
abaqus	t4-4-6	344.4	500.0	1.45	13.0	0.5
abaqus	t6-4-6	257.6	500.0	1.94	0.8	0.3
ampac	testbig.dat	412.4	412.3	1.00	1.2	0.4
ansys	m1	33.3	60.2	1.81	1.0	0.3
ansys	m2	313.0	500.0	1.60	2.8	1.2
ansys	m3	298.6	500.0	1.67	2.7	1.1
ansys	s1	292.9	500.0	1.71	2.0	1.1
ansys	s2	248.4	500.0	2.01	6.2	1.1
ansys	s3	194.0	500.0	2.58	1.3	0.2
arc3d	high.in	603.1	500.0	0.83	2.0	0.5
born	demo5	568.5	500.0	0.88	1.1	0.1
born	therm	398.5	500.0	1.25	0.8	0.2
dyna2d	dyna.2	354.9	500.0	1.41	0.5	0.3
FFT	1k x 1k	38.4	25.2	0.66	16.0	12.5
flo57	in.data	336.6	500.0	1.49	10.6	1.7
flo82	rae2822	325.3	500.0	1.54	2.1	0.8
lapack	1000x1000	330.5	500.0	1.51	7.7	2.6
linpack	100x100	18.6	28.8	1.54	0.3	0.2
linpeak	1000x1000	333.4	500.0	1.50	7.8	3.3
mopac	testbig.dat	508.3	500.0	0.98	0.9	0.2
wake	input.dat2	430.4	500.0	1.16	0.6	0.4
Mean		317.6	435.9	1.37	4.6	1.3
Median		330.5	500.0	1.50	2.0	0.4

Table 3: Characteristics of the Benchmark Programs

This table lists some characteristics of the benchmark programs, including counts of floating point operations and vector unit memory references. The data space size was measured using a one kilobyte page size. The working set size was computed in real time using a time quantum of one million references and a one kilobyte page size.

Vector Unit Memory Reference Breakdown					
Program	Dataset	Scalar Read %	Scalar Write %	Vector Read %	Vector Write %
abaqus	t2-4-6	9.0	4.1	57.8	29.1
abaqus	t3-4-6	7.9	3.5	59.2	29.4
abaqus	t4-4-6	5.1	3.7	61.8	29.4
abaqus	t6-4-6	28.6	14.1	36.6	20.7
ampac	testbig.dat	18.2	8.3	63.8	9.7
ansys	m1	10.3	2.5	59.3	27.9
ansys	m2	1.8	0.6	63.7	33.9
ansys	m3	3.2	1.0	61.4	34.4
ansys	s1	3.1	0.7	61.7	34.5
ansys	s2	5.8	1.1	57.0	36.1
ansys	s3	8.5	1.6	51.9	38.0
arc3d	high.in	14.3	6.2	57.1	22.4
born	demo5	28.5	11.1	37.7	22.7
born	therm	38.6	15.4	28.0	18.0
dyna2d	dyna.2	22.8	13.0	37.7	26.4
FFT	1k x 1k	8.4	8.4	41.6	41.6
flo57	in.data	5.4	2.2	62.7	29.7
flo82	rae2822	3.4	2.0	67.3	27.3
lapack	1000x1000	0.4	0.3	66.1	33.2
linpack	100x100	2.0	1.8	64.3	31.9
linpeak	1000x1000	0.2	0.2	66.4	33.1
mopac	testbig.dat2	25.3	6.1	53.7	14.8
wake	input.dat2	7.9	2.6	60.2	29.3
averages		11.8	4.8	55.5	27.8

Table 4: Breakdown of Vector Unit Memory References

This table provides a breakdown of vector unit memory references into four categories: scalar reads, scalar writes, vector reads, and vector writes. Values in the table are percentages of all vector unit memory references.

5. Results

5.1. Time Performance

This section contains timing results for the three simulation models: the model without a vector cache, the interleaved cache model, and the partitioned cache model. We briefly note that the results were generated using prototype compilers and simulators. The performance of these hypothetical machines are not meant to be indicative of any current or future products from Stardent or any other supercomputer vendor.

Performance numbers for the benchmark suite are presented in table 5. The table also contains performance ratios normalized to the *no cache* model, as well as arithmetic and geometric averages for each

Timing Simulation Results							
Program	Dataset	Megaflops			Ratios		
		int	part	none	int	part	none
abaqus	t2-4-6	10.7	10.2	5.8	1.8	1.8	1.0
abaqus	t3-4-6	15.2	14.4	7.0	2.2	2.1	1.0
abaqus	t4-4-6	16.1	15.6	3.8	4.2	4.1	1.0
abaqus	t6-4-6	14.0	10.3	2.6	5.4	4.0	1.0
ampac	testbig.dat	15.8	16.3	13.0	1.2	1.3	1.0
ansys	m1	6.3	6.2	4.4	1.4	1.4	1.0
ansys	m2	20.5	14.7	9.6	2.1	1.5	1.0
ansys	m3	11.3	9.3	6.7	1.7	1.4	1.0
ansys	s1	15.0	11.8	7.9	1.9	1.5	1.0
ansys	s2	10.6	6.1	4.5	2.4	1.4	1.0
ansys	s3	3.4	3.4	2.6	1.3	1.3	1.0
arc3d	high.in	17.2	17.6	10.2	1.7	1.7	1.0
born	demo5	17.2	17.0	4.7	3.7	3.6	1.0
born	therm	12.8	12.8	2.9	4.4	4.4	1.0
dyna2d	dyna.2	9.5	9.7	4.7	2.0	2.1	1.0
FFT	1k x 1k	4.2	8.5	16.5	0.3	0.5	1.0
flo57	in.data	12.5	12.1	7.9	1.6	1.5	1.0
flo82	rae2822	20.7	21.6	8.7	2.4	2.5	1.0
lapack	1000x1000	43.3	20.4	14.7	2.9	1.4	1.0
linpack	100x100	35.5	27.8	11.2	3.2	2.5	1.0
linpeak	1000x1000	15.3	19.5	13.0	1.2	1.5	1.0
mopac	testbig.dat	15.2	17.0	7.6	2.0	2.2	1.0
wake	input.dat2	19.9	21.1	8.9	2.2	2.4	1.0
Means (w/ FFT)							
arithmetic		15.7	14.1	7.8	2.0	1.8	1.0
geometric		13.6	12.7	6.8	2.0	1.9	1.0
Means (w/o FFT)							
arithmetic		16.3	14.3	7.4	2.2	1.9	1.0
geometric		14.4	13.0	6.5	2.2	2.0	1.0

Table 5: Performance Results for the Various Cache Models

This table contains the performance results for the benchmark suite. Performance was measured over the first 500 million vector unit references made by each application. The columns *int*, *part*, and *none* represent the simulated performance of the interleaved, partitioned, and no cache models, respectively.

column. The results show that both cache architectures improve performance dramatically on nearly all applications. The only negative result is the 1k x 1k FFT [Temp83], which has a large (16 Mbyte) dataset and poor locality of reference. We believe that the poor performance of this FFT is largely attributable to its algorithm, which was not designed for memory hierarchies; we discuss this topic further later. In the meantime, we list performance averages both including and excluding the FFT. Without the FFT, average performance improves by 5 to 10 percent.

The interleaved cache improves time performance from 20% to over 400% compared to a system without a cache. Arithmetic and geometric averages indicate that the interleaved cache doubles performance. This improvement arises from the speedup of both scalar and vector references. Scalar data references require 40 processor cycles without a cache, but only 2 processor cycles with a hit in the interleaved cache. Similarly, uncached vector references have an initial latency of 40 processor cycles, and the memory bus can only support two vector references every four processor cycles. The interleaved cache removes the initial latency and supports all three memory pipelines running at the full 64 MHz clock rate.

The partitioned cache improves performance by 30% to over 300% percent, with an average performance improvement of 80%. The amount of improvement is less than that for the interleaved cache model for most benchmarks, which we attribute to several factors. First, each write by the vector unit reduces read bandwidth while the independent read caches are updated. Second, each 1 MB read partition is only one-fourth the size of the 4 MB interleaved cache. Applications with working sets larger than a read partition but smaller than the interleaved cache will perform much better in the interleaved cache model. Finally, as discussed earlier, the combined cache size of the two read partitions is only half that of the interleaved cache. The *effective* combined cache size of the two read partitions is even lower, due to data duplication, and is probably closer to 1 MB than 2 MB. Size factors are probably responsible for the results on *ansys (m2)*, *ansys (m3)*, *ansys (s1)*, *ansys (s2)*, and *lapack*. Each of these applications has an average working set larger than 1 MB, but smaller than 4 MB. For the *ansys* programs, the interleaved cache model outperformed the partitioned cache model by 20% to 70%. In *lapack*, the largest of these programs, performance with the interleaved cache is double that with the partitioned cache.

Timing Simulation Results: Small Applications					
Program	Dataset	Ratios			Working Set Size (Mbytes)
		int	part	none	
abaqus	t2-4-6	1.8	1.8	1.0	0.3
abaqus	t3-4-6	2.2	2.1	1.0	0.3
abaqus	t4-4-6	4.2	4.1	1.0	0.5
abaqus	t6-4-6	5.4	4.0	1.0	0.3
ampac	testbig.dat	1.2	1.3	1.0	0.4
ansys	m1	1.4	1.4	1.0	0.3
ansys	s3	1.3	1.3	1.0	0.2
arc3d	high.in	1.7	1.7	1.0	0.5
born	demo5	3.7	3.6	1.0	0.1
born	therm	4.4	4.4	1.0	0.2
dyna2d	dyna.2	2.0	2.1	1.0	0.3
linpack	100x100	3.2	2.5	1.0	0.2
mopac	testbig.dat	2.0	2.2	1.0	0.2
wake	input.dat2	2.2	2.4	1.0	0.4
Means					
arithmetic		2.3	2.2	1.0	0.3
geometric		2.4	2.3	1.0	0.3

Table 6: Performance Ratios for Small Benchmarks

This table lists the performance ratios from table 5 and working set sizes from table 3 for programs with working set sizes smaller than either cache model.

The size differences between the interleaved and partitioned caches make it somewhat difficult to evaluate the relative merits of each model. To factor out these effects, we examine the performance of programs with relatively small working set sizes. Normalized performance ratios for these programs are listed in table 6, along with their average working set sizes. The largest of these programs has an average working set size of half a megabyte, and the average across these programs is only 0.3 megabytes. All of these programs clearly fit into both cache designs.

The average performance improvement for this subset of applications is 15 to 20% greater than the average improvement for the entire benchmark suite, and roughly 10% better than the benchmark suite minus the FFT. As expected, caches are more effective on programs which fit into the cache. The interleaved cache remains the better alternative, although the performance gap between the two alternative cache models has decreased from 11 percent for the full benchmark suite to 5 percent for this subset. By factoring out cache size, bandwidth reducing effects, i.e. interleave conflicts in one model and write updates in the other, are the major difference between the two models. Although we have no direct measurements, it appears that write updates in the partitioned cache have a more negative effect on performance than conflicts in the interleaved cache. (The model for the partitioned cache also omits accounting for the delay in the fetch of the remainder of a line, and also for write stalls in the write buffer, so small differences between various results may not be significant.)

We had earlier mentioned that the 1k x 1k FFT is the only negative result for both cache based models. This is a two dimensional FFT algorithm presented in [Temp83], where transforms are performed in parallel, with each vector containing one element from each different transform. Each individual transform is stored in memory as a 1k strip of contiguous elements, and the 1k different transforms are stored one after another. Consequently, a vector containing one element from each transform has a vector stride of 1k words, and since the dataset size of the 1k x 1K FFT is 16 Mbytes, each reference causes a miss. The partitioned cache model performs better than the interleaved cache model only because writes always complete in one cycle using the partitioned cache write buffer. In the interleaved cache, writes often miss in the cache, triggering a block fetch.

This particular FFT algorithm is not designed for memory hierarchies, but does improve processor utilization, as vector lengths increase by performing transforms in parallel. Alternative FFT algorithms do exist which account for memory hierarchies and attempt to minimize cache misses. [So88b] simulated a 1k x 1k FFT which uses a library routine written especially for the 64 Kbyte cache in the IBM 3090 VF [Agar86]. The routine operates on vectors with unit stride, and utilizes data blocks as often as possible once they are cache resident. The miss ratio of this FFT on the small 3090 cache is only 3 percent.

Our discussion of the FFT algorithm has been illustrative of an important point - programs can be rewritten to perform well in a cached system, just as they can be rewritten (in some cases, and in our opinion, with more difficulty) to work well in a vector machine.

As a final note, programs making a large number of scatter/gather references, such as *born*, benefit more than other programs from the addition of a vector cache. This occurs because scatter/gather references were implemented as a sequence of scalar loads and stores in the first generation Ardent Titan. Scalar loads and stores are not pipelined, thus each scatter/gather reference takes 40 processor cycles to complete without a cache. The second generation Stardent machines can vectorize these references.

5.2. Cache Metrics

Cache miss and bus traffic ratios from the simulations are shown in table 7. The *bus traffic ratio* is defined as the ratio of memory traffic with a cache (due to the reading and writing of cache blocks) to memory traffic without a cache. Low cache miss ratios reduce bus traffic ratios well below unity. By reducing bus traffic, contention for main memory decreases and the scalability of multiprocessor systems improves. Table 7 contains three sets of averages, one for all programs, one for all programs minus the FFT, and one for the subset of small programs (boldface entries in table 7). Each set lists three types of averages: (1) weighted averages, which are arithmetic averages weighted by the number of references in each application, (2) unweighted arithmetic averages, and (3) unweighted geometric averages. We have most confidence in the weighted averages. Simple arithmetic averages are skewed by poor performing benchmarks (i.e. the FFT), while the geometric averages are optimistic where low values equal better performance.

From table 7, bus traffic ratios for the partitioned cache are much higher than traffic ratios for the interleaved cache. This is due to the write through mechanism used in the partitioned cache write buffer, which forces bus traffic ratios to be at least as large as the fraction of writes in each trace.

Cache miss ratios range from as low as a few hundredths of a percent to over 40 percent for benchmarks with poor locality, such as the FFT. Miss ratios for the partitioned cache are generally larger than miss ratios for the interleaved cache, as the partitioned cache is smaller and allows data duplication.

The smaller effective size of the partitioned cache is most noticeable in the miss ratios for the *ansys* datasets *m2*, *m3*, *s1*, and *s2*, and the *flo57*, *lapack*, and *mopac* benchmarks. Partitioned cache miss ratios for these benchmarks are nearly an order of magnitude larger than interleaved cache miss ratios. With the exception of *mopac*, all of these benchmarks have working sets in excess of 1 Mbyte but less than 4 Mbytes. As noted in the timing results, most of these benchmarks also execute significantly faster in the interleaved cache model.

For the small benchmarks, miss ratios for the two cache models are in good agreement except in *mopac*. The *mopac* result is probably due to set mapping conflicts in the smaller partitioned read caches, as all caches are direct-mapped. *Mopac* contains a large fraction of scalar reads, which may be contending with vector reads for cache sets.

5.3. Comparisons

We have shown how vector caches can dramatically increase the performance of a vector processor. This increase is most noticeable when comparing the performance of our hypothetical machines to existing Ardent and Stardent products. The Ardent Titan, with a 16 MHz integer unit, 8 MHz vector unit, and 16 MHz bus, runs the *100x100 linpack* benchmark at 6 Mflops. The Stardent ST3000, with a 32 MHz integer unit, 16 Mhz vector unit, and identical 16 MHz bus, runs 100x100 linpack at 10 Mflops. Our base configuration without cache, but with 64 MHz integer and vector units and the same 16 MHz bus, runs 100x100 linpack at only 11.2 Mflops. The addition of vector caches allows us to extract a good fraction of the extra performance. The interleaved cache model runs 100x100 linpack at 35.5 megaflops, while the partitioned cache model runs the benchmark at 27.8 megaflops.

Based solely on our results, the interleaved cache appears to be the better choice of the two cache models. In addition to the results, other factors tend to favor the interleaved cache model. Its performance is less sensitive to the working set size of an application, as each memory pipeline can access the entire

Cache Miss and Bus Traffic Ratios					
Program	Dataset	Interleaved Cache 4 megabytes copy back		Partitioned Cache 2 megabytes infinite write buffer	
		miss ratio %	traffic ratio	miss ratio %	traffic ratio
abaqus	**t2-4-6**	**0.12**	**0.018**	**0.14**	**0.343**
abaqus	**t3-4-6**	**0.14**	**0.022**	**0.14**	**0.340**
abaqus	**t4-4-6**	**0.16**	**0.025**	**0.17**	**0.344**
abaqus	**t6-4-6**	**0.10**	**0.016**	**0.14**	**0.358**
ampac	**testbig.dat**	**0.17**	**0.022**	**0.37**	**0.210**
ansys	**m1**	**0.08**	**0.011**	**0.11**	**0.312**
ansys	m2	0.23	0.036	2.12	0.526
ansys	m3	0.20	0.031	2.02	0.515
ansys	s1	0.20	0.031	1.89	0.503
ansys	s2	0.25	0.038	1.49	0.491
ansys	**s3**	**0.04**	**0.006**	**0.03**	**0.398**
arc3d	**high.in**	**0.23**	**0.027**	**0.40**	**0.318**
born	**demo5**	**0.05**	**0.007**	**0.17**	**0.351**
born	**therm**	**0.04**	**0.006**	**0.08**	**0.341**
dyna2d	**dyna.2**	**0.06**	**0.008**	**0.07**	**0.400**
FFT	1k x 1k	45.60	5.255	41.40	3.813
flo57	in.data	1.50	0.177	4.47	0.355
flo82	rae2822	0.10	0.016	0.26	0.331
lapack	1000x1000	0.58	0.053	3.66	0.364
linpack	**100x100**	**0.02**	**0.003**	**0.04**	**0.342**
linpeak	1000x1000	3.47	0.552	4.27	0.674
mopac	**testbig.dat**	**0.11**	**0.014**	**1.05**	**0.297**
wake	**input.dat2**	**0.03**	**0.004**	**0.03**	**0.321**
Means (w/ FFT)					
weighted		0.50	0.068	1.25	0.399
arithmetic		2.33	0.277	2.81	0.532
geometric		0.18	0.025	0.42	0.412
Means (w/o FFT)					
weighted		0.39	0.055	1.15	0.390
arithmetic		0.36	0.051	1.05	0.383
geometric		0.14	0.020	0.34	0.372
Means (small progs)					
weighted		0.10	0.014	0.23	0.337
arithmetic		0.10	0.014	0.21	0.334
geometric		0.08	0.011	0.13	0.331

Table 7: Miss and Traffic Ratios from the Simulations

This table lists cache miss and bus traffic ratios for the benchmark suite, again measured over the first 500 million vector unit references. Averages are provided for a) all programs, b) all programs minus the FFT, and c) the small programs from table 6. Small programs are in boldface.

four megabytes of cache. The interleaved cache does not waste cache space due to data duplication, unlike the partitioned cache. The interleaved cache generates less bus traffic, since it is a copy back cache, whereas the partitioned cache uses a write through buffer. Finally, the interleaved cache appears to be more feasible to implement. It assumes two cycle technology to access the cache through the crossbar switch. The partitioned cache assumes single cycle technology, an infinite write buffer, and a single cycle update mechanism from the write buffer to the read caches.

The partitioned cache does have certain advantages. There is no need for a crossbar switch between memory ports and cache interleaves. A certain level of cache bandwidth is guaranteed through the separate read caches and write buffer, whereas interleave conflicts can reduce interleaved cache bandwidth. We note that interleave conflicts do not appear to be a problem, as evidenced by our timing results and earlier work in [Bask76]. We also note that partitioned cache bandwidth can also be reduced due to write updates. An interesting advantage of the partitioned cache is its write buffer, which makes the performance of vector writes insensitive to the vector stride. Each write takes one cycle to place in the buffer, and the total amount of bus traffic due to writes is equal to the total number of bytes that are written. In the interleaved cache model, write buffering can be used to eliminate the time penalty of a cache miss, but bus traffic remains excessive if few words in a block are modified before the block is written back.

The results from this work compare favorably with results from earlier studies. Our timing results show that a vector cache improves the performance of a 128 Mflop machine by 80 to 100%, in agreement with the 30 to 130% improvement seen in [Abu86] for the 94 Mflop Alliant FX/8. For memory limited applications, [Gee90] estimated that caches can improve performance by at least 50% for vector machines with a short memory latency, i.e. on the order of 10 to 15 processor cycles. For machines with a memory latency on the order of 50 cycles, [Gee90] predicted that caches can improve performance by over 150%. The memory latency in our model is 40 processor cycles, and the 150% performance improvement found for *100 x 100 linpack* correlates well with the predictions in [Gee90].

5.4. Extensions

Although our cache models and timing parameters appear reasonable, we believe that it is important to project our results over a wider range of cache sizes, access times, and memory latencies. In this section, we fit the timing results to a very simple performance model which accounts for some variation in these parameters. We then use this model to extend our results over a larger design space.

To keep the modeling effort tractable, we make the following assumptions: (a) performance is limited by the memory access rate of vector data, (b) Amdahl's Law applies to the performance effect of a vector cache, i.e. some fraction of program execution will not improve at all with a vector cache, and (c) this fraction is approximately 25%, corresponding to a maximum speedup of four. This maximum speedup was observed in several entries in table 5. Our model is summarized in equations (1) - (3).

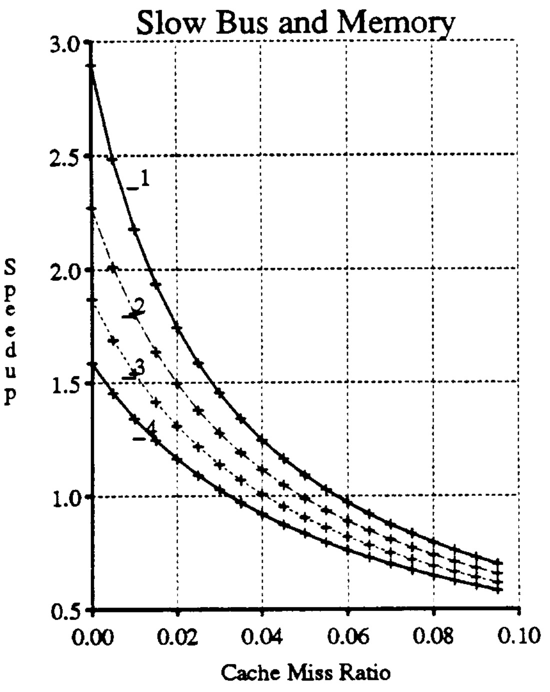
$$(1) \qquad S = \frac{1}{.25 + .75\frac{R_m}{R_c}} \quad \text{(Amdahl's Law)}$$

$$(2) \qquad R_m = \frac{W * VL}{L + BR * VL}$$

$$(3) \qquad R_c = \frac{3}{A + 3 * M * MS}$$

In the model, S is the speedup from a vector cache, R_m is the reference rate (elements/cycle) of data from memory, while R_c is the reference rate out of cache. In equation 2, W is the bus width in words, VL the maximum vector length, L the memory latency in cycles, and BR the bus rate in processor cycles. After initial latency, memory can supply W * VL words every BR * VL processor cycles. In equation (3), A is the *effective* cache access time, M is the miss rate, and MS is the miss service time. A three-ported vector cache should supply *three references every A cycles* if no misses occur. Misses increase the effective access time by a factor of 3 * M * MS.

Our timing results for the interleaved cache architecture fit reasonably well into this model. For the design we studied, W = 2, VL = 32, L = 40, and BR = 4, yielding a peak R_m of .38 elements/cycle. The interleaved cache model has a bank access time of 2 cycles, but the effective A is 1 because the three memory ports can initiate references every cycle, and the eight banks combined can support this reference rate. M is 0.005, and MS = L = 40 cycles, since the cache can begin processing references as soon as the missed word is read from memory. These parameters yield a R_c of 1.88 references/cycle. The predicted speedup from the model is 2.5, which is close to the observed speedup of 2.2.



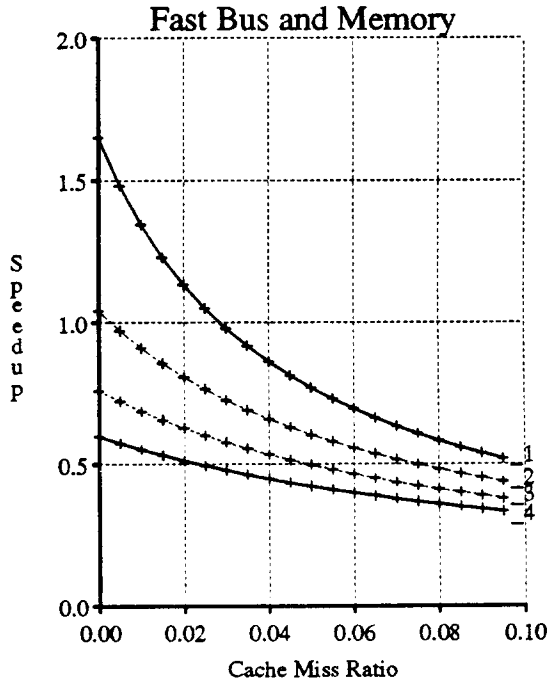


Figure 4: Performance Model Results

This figure shows speedup due to a vector cache in (a) a conservative implementation (1 bus cycle = 4 processor cycles, memory latency = 40 processor cycles), and (b) an aggressive implementation (1 bus cycle = 1 processor cycle, memory latency = 13 processor cycles). Curves are parameterized by effective cache access time, which is the number of processor cycles to complete three vector references (one per memory port).

We can now use the model to explore varying cache access times, cache miss ratios, and memory latencies. Figure 4 contains two plots, one for the fairly slow bus and memory system that we simulated, and another assuming a very aggressive bus and memory system. Each plot shows curves for a range of effective cache access times. From the first plot, we see that vector caches are extremely effective when paired with a slow memory system. A cache with a long access time of four processor cycles improves performance until the miss ratio exceeds 3%. Faster caches remain effective up to miss ratios of 4% to 6%. These break even points correspond roughly to cache sizes between 64 and 256 kilobytes [Gee90, Smit87]. For a miss ratio of 2%, which corresponds roughly to a one megabyte cache, speedup ranges from 15% to 75%.

As expected, caches are less effective when memory is fast enough. In our very simple model, speedup only exists when the cache access time is one cycle. Slower caches fail to improve or reduce performance, for the fast memory system, even if miss ratios are zero. Despite these results, slower caches can still be beneficial on applications using short vectors, as the access rate from main memory decreases significantly. Slower caches may also increase performance in multiprocessor systems, where memory contention also reduces access rates. Our model currently does not account for these effects.

6. Conclusions

To measure the performance impact of cache memory in a vector processing environment, accurate timing simulators were developed for two cache models grafted onto the Ardent Titan machine architecture. The simulator is implemented via profiled code and shared libraries, and can reasonably simulate several minutes of actual program execution. The timing parameters in our models are chosen to represent possible next generation Stardent machines, and reflect current trends in processor and memory technology. These trends indicate that performance is increasingly bus and memory limited, making the performance of our cache models all the more critical.

Our two cache models represent alternative methods to derive sufficient cache bandwidth in the presence of several pipelined memory access streams. Both models appear feasible to implement in current technology. One cache architecture utilizes proven interleaving techniques, while the second cache is partitioned across the two read and one write memory pipelines. A third timing model without caches was simulated for purposes of comparison.

Results on the benchmark suite indicate that average performance doubles when an interleaved cache is included in the system. The performance increase over a system without caches ranges from 20% to over 400% on some benchmarks. The performance improvement due to the partitioned cache is slightly less. Our hypothetical machine runs 100x100 linpack at 11.2 megaflops without a cache, 27.8 megaflops with a 2 Mbyte partitioned cache, and 35.5 megaflops with a 4 Mbyte interleaved cache. In comparison, the 16 MHz Ardent Titan runs linpack at 6 Mflops and the 32 MHz Stardent ST3000 runs linpack at 10 Mflops.

Measured cache miss ratios are low, averaging approximately 0.5 percent for the interleaved cache and 1.25 percent for the partitioned cache. The low miss ratios translate into low bus traffic ratios for the copy back interleaved cache. The partitioned cache uses a write through buffer, thus traffic ratios cannot be smaller than the fraction of writes in the trace. This factor is important when bus saturation in a multiprocessor system is an issue.

Of the two alternatives, the interleaved cache model appears most promising, mainly because the entire four megabytes of cache is available to all memory access pipelines. In the partitioned cache model, the cache is a combination of several independent, smaller caches, whose performance begins to deteriorate as working set sizes approach the size of each independent cache. Partitioned cache performance is also affected by the need to maintain consistency between the independent caches. We evaluated this effect separately by examining the performance of smaller applications, to factor out cache size differences between the two models. Over these smaller applications, the interleaved cache outperforms the partitioned cache by 5 percent, compared to 11 percent over all applications.

Both caches will reduce performance on applications with large vector strides and poor spatial locality. On a naive FFT, the partitioned cache model is two times slower and the interleaved cache model four times slower than the model without a cache. A properly written FFT can be shown to perform well [So88a,b]. We note that applications which perform poorly with caches are rare, and could be recoded to improve locality and exploit the presence of a cache.

An obvious limitation to the simulations is that results were only provided for a single set of cache and timing parameters. Although these parameters were chosen to be as realistic as possible, we would like to evaluate the performance effect of a vector cache over a wider range of implementations. To address this issue, we developed a simple model to evaluate the speedup provided by a vector cache. After verifying this model with our timing results, we extended our performance estimates over a large design space. For a slow memory system, we found that vector caches improve performance over a wide range of miss ratios and cache access times. Vector caches are less effective given an aggressive memory system implementation. Short cache access times are necessary, unless memory system bandwidth is decreased due to short vector lengths or contention from other processors.

Acknowledgements

The authors would like to thank Glen Miranker, John Sanguinetti, and Mike McNamara at Stardent Computer for providing ideas for the cache models and resources to run the simulations.

Bibliography

[Abu86] Abu-Sufah, W., and Malony, A.D., "Experimental Results for Vector Processing on the Alliant FX/8," CSRD Rpt. No. 539, University of Illinois, Urbana, IL, 1986.

[Agar86] Agarwal, R.C., and Cooley, J.W., "Fourier Transform and Convolution Subroutines for the IBM 3090 Vector Facility," *IBM Journal of Research and Development*, vol. 30, no. 2, 1986, pp. 145-162.

[Bask76] Baskett, F., and Smith, A.J., "Interference in Multiprocessor Computer Systems with Interleaved Memory," *Communications of the ACM*, vol. 19, no. 6, 1976, pp. 327-334.

[Cala88] Calahan, D.A., "Performance Evaluation of Static and Dynamic Memory Systems on the CRAY-2," *Proceedings of the 1988 International Conference on Supercomputing*, July, 1988, St. Malo, France, pp. 519-524.

[Cheu84] Cheung, T., and Smith, J., "An Analysis of the CRAY X-MP Memory System," *Proceedings of the 1984 International Conference on Parallel Processing*, August, 1984, Bellaire, MI, pp. 499-505.

[Died88] Diede, T., Hagenmaier, C., Miranker, G., Rubinstein, J., and Worley, W., "The Titan Graphics Supercomputer Architecture," *Computer*, September, 1988, pp. 13-30.

[Gee90] Gee, J., and Smith, A.J., "Vector Processor Caches," paper in preparation, 1990.

[HePa90] Hennessy, J.L., and Patterson, D.A., *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, 1990.

[McKe69] McKellar, and Coffman, E.G. Jr., "Organizing Matrices and Matrix Operations for Paged Memory Systems," *Communications of the ACM*, vo. 12, no. 3, 1969, pp. 153-165.

[Neve89] Neves, K.W., "Supercomputers: The Next Generation," *Scientific Information Bulletin*, 14(4), 1989, pp. 77-95.

[Smit79] Smith, Alan Jay, "Characterizing the Storage Process and its Effect on the Update of Main Memory by Write-Through", Journal of the ACM, 26, 1, January, 1979, pp. 6-27.

[Smit82] Smith, A.J., "Cache Memories," *ACM Computing Surveys*, vol. 14, no. 3, September, 1982, pp. 474-529.

[Smit85] Smith, A.J., "Cache Evaluation and the Impact of Workload Choice," *Proceedings of the 12th International Symposium on Computer Architecture*, June, 1985, Boston, MA, pp. 64-73.

[Smit87] Smith, A.J., "Line (Block) Size Choices for CPU Cache Memories," *IEEE Transactions on Computers*, vol. C-36, no. 9, September, 1987, pp. 1063-1075. (See correction IEEETC, 38, 6, June, 1989, p. 927.)

[So88a] So, K., and Zecca, V., "Cache Performance of Vector Processors," *Proceedings of the 15th Annual Symposium on Computer Architecture*, May, 1988, Honolulu, HI, pp. 261-268.

[So88b] So, K., and Zecca, V., "Program Locality of Vectorized Applications Running on the IBM 3090 with Vector Facility," *IBM Systems Journal*, vol. 27, no. 4, 1988, pp. 436-452.

[Temp83] Temperton, C., "Self-Sorting Mixed-Radix Fast Fourier Transforms," *Journal of Computational Physics*, vol. 52, no. 1, October, 1983, pp. 1-23.

[Trev77] Trevidi, K.S., "On the Paging Performance of Array Algorithms," *IEEE Transactions on Computers*, vol. C-26, no. 10, October, 1977, pp. 938-947.