Comparing the Memory System Performance of the HP V-Class and SGI Origin 2000 Multiprocessors using Microbenchmarks and Scientific Applications*

Ravi Iyer† Nancy M. Amato‡¶ Lawrence Rauchwerger§¶ Laxmi Bhuyan†
{ravi,amato,rwerger,bhuyan}@cs.tamu.edu
Department of Computer Science, Texas A&M University
College Station, TX 77843-3112, USA

Abstract

As processor technology continues to advance at a rapid pace, the principal performance bottleneck of shared memory systems has become the memory access latency. In order to understand the effects of cache and memory hierarchy on system latencies, performance analysts perform benchmark analysis on existing state-of-the-art multiprocessors. In this study, we present a detailed comparison of the memory system of two recent commercial ventures, the HP V-Class and the SGI Origin 2000. Our goal is to compare and contrast design techniques used in these multiprocessors to tolerate the effect of memory latency. Our experimental methodology uses microbenchmarks as well as scientific applications to characterize the user-level performance.

1 Introduction

Shared memory multiprocessors have gained widespread acceptance as a means to provide powerful parallel computing. The design and evaluation of shared memory multiprocessors has been the topic of much research. To build scalable shared memory multiprocessors, the use of small nodes connected through a powerful scalable interconnect has been shown to be quite attractive. Recent shared memory systems such as the SGI Origin 2000 [7] and the HP V-Class [5] multiprocessors are excellent examples of this trend.

With recent advances in processor technology, the performance bottleneck for current and future shared memory multiprocessors has become the latency of memory access. With the use of deeper cache/memory hierarchies, it is difficult to characterize the effect of memory access on application performance using a theoretical approximation. Recent work [1, 2, 6, 10] in this area mainly uses microbenchmarks and typically focuses on understanding the performance of a single multiprocessor. In this paper, we focus on an in-depth understanding of the cache/memory performance of two state-of-the-art multiprocessors, the HP V-Class and the SGI Origin 2000. Our evaluation methodology employs both microbenchmarks and scientific applications.

We use several microbenchmarks to illustrate the impact of technological advancements in processor design (such as multiple outstanding misses) and system protocols (such as optimizations to cache coherence protocols) on the latencies of uniprocessor access patterns and multiprocessor sharing patterns found in large-scale applications. We also employ scientific applications to provide an overall perspective on user-level performance. Using these experiments, we determine relative advantages and disadvantages of design techniques used in the two systems and evaluate the effectiveness of their cache and memory systems for various types of workloads.

Our uniprocessor experiments show that current processors supporting multiple outstanding cache misses are capable of reducing the average access latency by a significant factor over single outstanding cache miss execution. When comparing the two systems, we find that the uniprocessor load and store latencies measured on the HP V-Class tend to be 40 to 45% lower than those on the SGI Origin 2000. Using multiprocessor benchmarks, we study the impact of sharing degree on the access latency for dominant sharing patterns. We find that an increase in the sharing degree has relatively no impact on the HP V-Class latency, whereas significantly increases the average latency experienced on the SGI Origin 2000. Using scientific applications, we observe that while the SGI Origin 2000 experiences lower execution time, the HP V-Class generally exhibits better speedups. Such findings can help system designers gain a better understanding of existing system bottlenecks and provide a performance prediction mechanism for large-scale application developers.

The rest of the paper is organized as follows. Section 2 presents an overview of the two systems used in this study. Section 3 presents a detailed description of the uniprocessor microbenchmark suite and the results obtained on each system. Section 4 presents the multiprocessor benchmark suite and results for dominant multiprocessor sharing patterns. Section 5 uses scientific applications for a user-level performance comparison of these memory systems. Finally, Section 6 summarizes our results.

*Supported in part by a Hewlett-Packard Equipment Grant.
†Supported in part by NSF Grants MIP 9622740 and CCR 9810205.
‡Supported in part by NSF CAREER Award CCR-9624315 and NCSA Grant CCR970010N.
§Supported in part by NSF CAREER Award CCR-9734471 and NCSA Grant ASC980006N.
¶Supported in part by NSF Grant ACI-9872126 and DOE ASCI ASAP (Level 2 Program) Grant B347886.

To appear in the *Proceedings of the 13th ACM-SIGARCH International Conference on Supercomputing*, June 1999, Rhodes, Greece.

2 System Overview

In this paper, we study the performance of two commercial shared memory multiprocessors, the HP V-Class [5] and the SGI Origin 2000 [7]. While these systems scale up to or beyond 512 processors, we chose a 16-processor configuration due to its availability [9, 12].

2.1 System Architectures

A block diagram of the 16-processor HP V-Class server architecture is shown in Figure 1. The symmetric multiprocessor design utilizes Exemplar Processor Agent Controllers (EPAC), 8 Exemplar Memory Access Controllers (EMAC) and 8 Exemplar PCI Interface Controllers (EPIC). These controllers are connected centrally by a HP Hyperplane crossbar comprised of four ERACs (Exemplar Routing Attachment Controllers). Each processor agent (EPAC) is connected to two HP PA-8200 processors, that are based on the RISC Precision Architecture (PA-2.0). The PA-8200 is a 4-way out-of-order superscalar processor that runs at a speed of 200 MHz with 2 Mbytes of instruction cache and 2 Mbytes of data cache. Each EMAC memory controller is connected to a piece of the memory. The memory node is interleaved using four banks. Cache lines are interleaved across the four banks within a memory node and across the eight memory units, resulting in an overall 32-way interleaving in the memory system. The 16-processor configuration follows the uniform memory access (UMA) principle as shown in Figure 1.

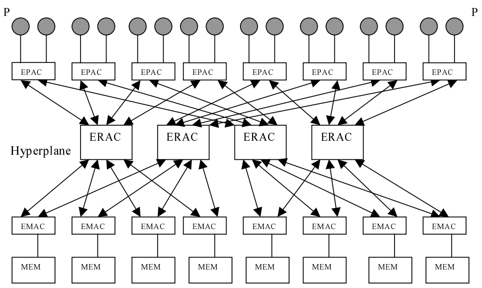


Figure 1: 16-processor HP V-Class system.

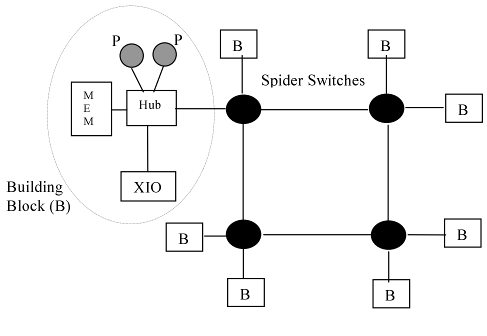


Figure 2: 16-processor SGI Origin 2000 system.

A block diagram of a 16-processor SGI Origin 2000 multiprocessor is shown in Figure 2. The SGI Origin 2000 is a cache-coherent non-uniform memory access (CC-NUMA) multiprocessor based on dual-processor basic building blocks. Each block contains 2 MIPS R10000 processors and 512 MB of memory contained in 1-8 memory banks (DIMMs). The R10000 is a 4-way out-of-order superscalar processor running at a speed of 250MHz with 32 Kbytes of L1 cache and 4MB of unified L2 cache. The L1/L2 cache line sizes are 32 bytes and 128 bytes respectively. Within each node, the processors are connected to a Hub chip, an ASIC that provides connections to the local memory, the I/O, and the hypercube interconnect. The hypercube is made up of 6x6 bi-directional SGI Spider crossbar switches connected using the Craylink interconnect. Each router connects two local blocks (4 processors) and two other routers in a 16-processor configuration. Each DIMM memory bank provides 4-way memory interleaving on a cache line basis, so a node board could support up to 32-way interleaving (however, full bandwidth can be obtained with only 4-way interleaving).

	HP V-Class	SGI Origin
Processor		
Speed	200 MHz	250 MHz
ILP Behavior	4-way	4-way
Cache Size	2MB	32KB/4MB
Cache Speed	3 cycles	3/10 cycles
Cache Line	32 bytes	32/128 bytes
Memory		
Organization	UMA	NUMA
Interleaving	32-way	4-way
Management	round-robin	round-robin
TLB Issues	120 entries	64 entries
Coherence (optimization)	MESI-like (migratory)	MESI-like (speculative)
Scalability		
Node	16-way	2-way
Topology	crossbar	hypercube

Table 1: High Level Comparison of 16 processor HP V-Class and SGI Origin 2000.

2.2 Coherence Protocols

Both the HP V-Class [11] and the SGI Origin 2000 [7] use a variation of the invalidate-based full-map directory protocol [3] and the well-known MESI cache protocol [8]. However, each multiprocessor has its own enhancement to the protocol. An overview of the protocols is as follows. The MESI protocol uses four states, *modified, exclusive, shared* and *invalid*, to identify the state of the line in the cache. The full-map directory protocol maintains a bit per processor per block to keep track of all the sharers of a block. The directory also maintains the state of each block, namely *private, shared* and *not_present*. In an invalidation based protocol, when ownership is requested for a block in *shared* state, invalidation signals are used to purge the existing copies in each of the sharer's caches.

In the V-Class protocol [2], the enhancement to the protocol is targeted towards migratory sharing patterns. When a processor (requester) issues a read to a block that is currently dirty in another processor's (owner's) cache, the block is invalidated from the owner's cache and supplied to the requester in exclusive state. Such a transfer does not update memory with the data. To convert from exclusive to shared state later, in the case of multiple consumers, the data is sent back to memory from the first consumer and then sent to subsequent consumers.

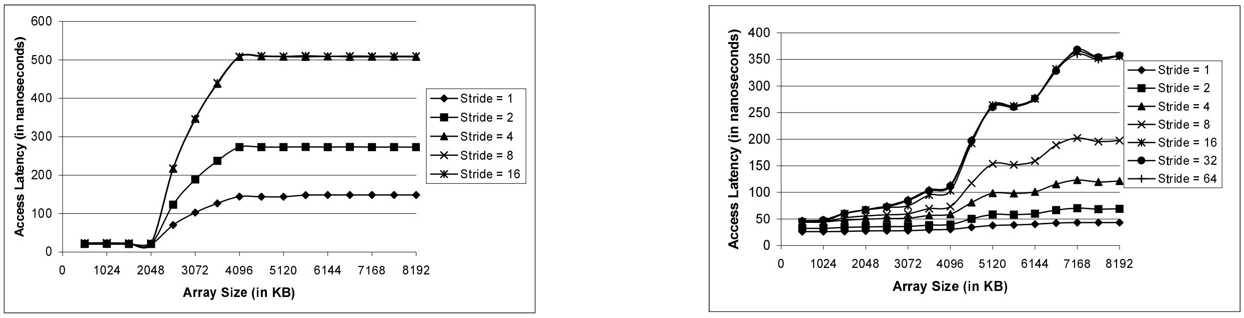
In the Origin protocol [7], the enhancement to the protocol is for memory requests that find data in the private state in the directory. The data either exists in an *exclusive*

```
Array : A[0:i:N] : initialized to i-stride
tmp = A[N-1];
for (; tmp > = 0;)
     tmp = A[tmp];
```

Figure 3: The Load-Use Benchmark

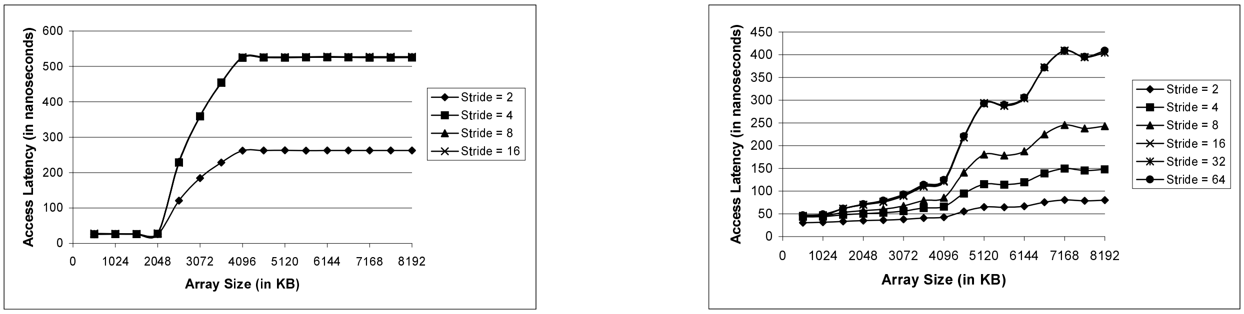
```
Array : A[0:i:N] : initialized to i-stride
A[N-2] = 0;
tmp = A[N-1];
for (; tmp > = 1;)
     A[tmp-1]=0;
     tmp = A[tmp];
```

Figure 4: The Store-Use Benchmark



(a) HP V-Class **(b) SGI Origin 2000**

Figure 5: Uniprocessor Performance: Single Outstanding Loads (change of scale).



(a) HP V-Class **(b) SGI Origin 2000**

Figure 6: Uniprocessor Performance: Single Outstanding Stores (change of scale).

state or *modified state* in the owner's cache. Since transferring data from the owner to the requester takes longer than transferring control information, speculative data is supplied from the memory node (by assuming that the block is in exclusive state) and an intervention request is sent to the owner. When the owner receives the intervention request, if the block is indeed in exclusive state, it sends only control information regarding this to the home node and the requester. However, if the owner holds the block in modified state, the owner sends dirty data to the requester as well as to the memory for updating the block. Unlike, the V-Class protocol, in both cases, the owner retains the block in shared state in the cache.

2.3 Experimental Methodology

Several differences can be immediately observed when studying the 16-processor HP V-Class and the Origin 2000 architectures (Table 1). A fundamental architectural difference is memory organization: the HP V-Class 16-processor node is a UMA (uniform memory access) multiprocessor with all the memories equidistant from the processors, while the 16-processor Origin 2000 exhibits a NUMA (non-uniform memory access) memory organization with a local memory and several remote memories. In this work, we limit ourselves to the performance characterization of the cache and memory system using microbenchmarks and scientific applications.

Using uni-processor benchmarks, we study the access latencies of simple loads and stores with varying degrees of spatial locality. We also characterize the impact of processor design such as the number of outstanding cache misses. Next, we use multiprocessor benchmarks to show the impact of the coherence protocols and their optimizations. Each microbenchmark was written in C, optimized by the compiler at level O2, and run over 100 times to determine the average latency per access. We verified many of our results using hardware counters on the machine and qualitative comparisons with existing results such as [2]. Finally, we use scientific applications to present an example of user-level performance such as speedup and execution time.

3 Uniprocessor Performance

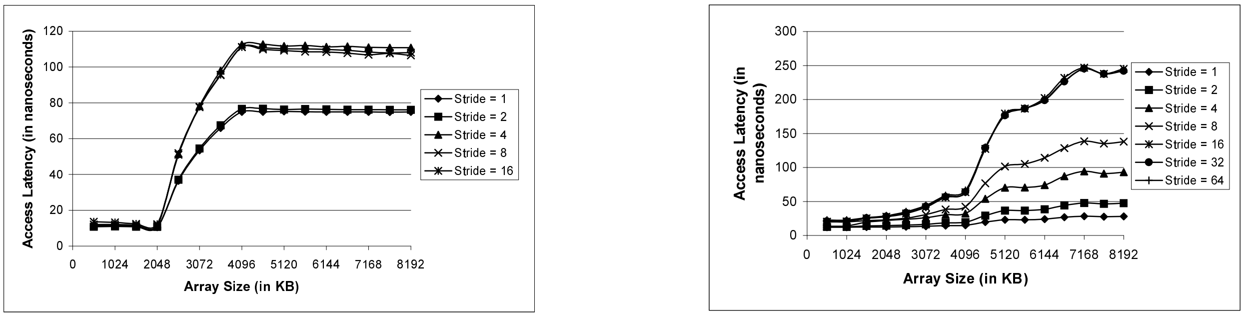
In this section, we begin our uniprocessor characterization using cache/memory access latency as our metric. We use several uni-processor load and store microbenchmarks based on [1] to compare the performance of the HP V-Class and the SGI Origin 2000 multiprocessors.

```
for (i = N-1; i > = 0; i-=stride)
      tmp = tmp + A[i];
```

Figure 7: The Load-All Benchmark

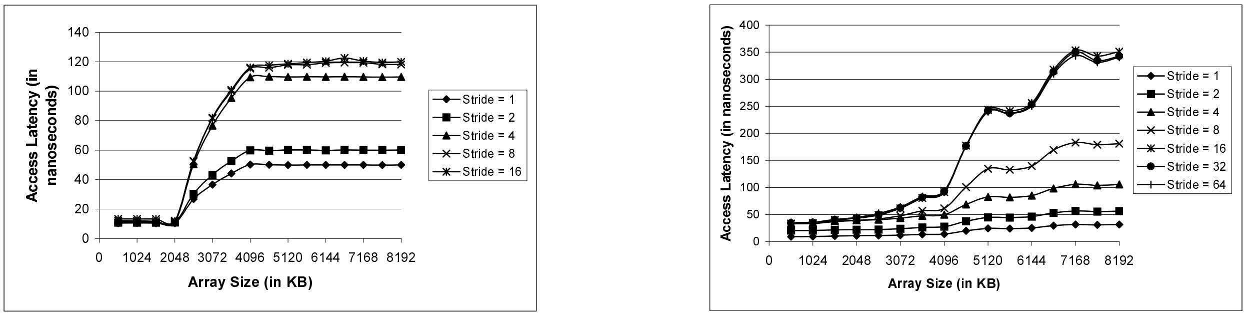
```
for (i = N-1; i > = 0; i-=stride)
      A[i] = 0;
```

Figure 8: The Store-All Benchmark



(a) HP V-Class **(b) SGI Origin 2000**

Figure 9: Uniprocessor Performance: Multiple Outstanding Loads (change of scale).



(a) HP V-Class **(b) SGI Origin 2000**

Figure 10: Uniprocessor Performance: Multiple Outstanding Stores (change of scale).

3.1 Single Outstanding Load/Store Performance

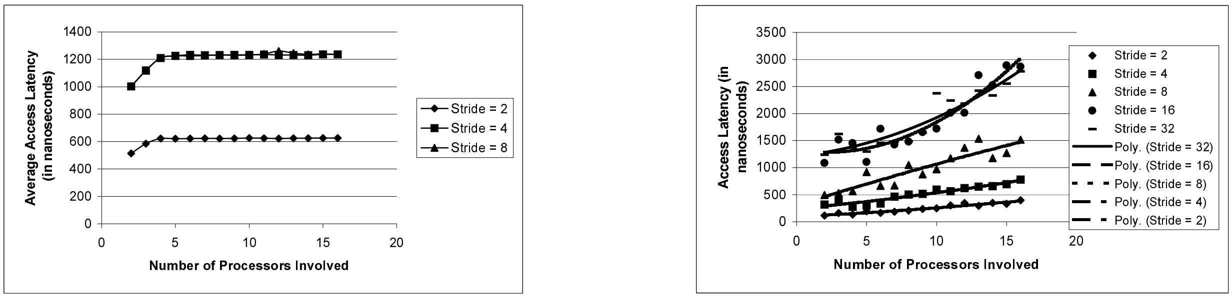
We characterize the latency of uniprocessor loads and stores using the Load-Use (Figure 3) and Store-Use (Figure 4) benchmarks adapted from Abandah and Davidson [1]. The benchmarks traverse an array of N 8-byte elements using different stride lengths specified in the number of elements. Current superscalar processors, including the R10000 [13] and the PA 8200 [4], allow multiple outstanding loads and stores to take advantage of latency hiding. To identify the performance of a single outstanding processor load or store, the microbenchmarks use the value read from the array to index the next location in the array. This ensures a dependency between consecutive loads/stores and thus both requests cannot be issued simultaneously.

Figure 5 presents the data gathered on the HP V-Class (Figure 5a) and the SGI Origin 2000 (Figure 5b) using the Load-Use benchmark. On the HP V-Class multiprocessor, we find that the length of the stride does not impact the latency when the data size is below 2 MB. This is directly attributable to the 2 MB cache size, so that all load accesses result in cache hits (approximately 3 processor cycles). The average latency was found to be approximately 20 nanoseconds. As the data size grows beyond the cache size, the chosen stride length has a growing impact on the latency. Consider a single cache line of 32 bytes. A stride of length 1 accesses four elements residing within the same cache line. Thus one outstanding cache miss brings four data elements (32 bytes) into the cache and serves three subsequent processor requests as cache hits. This results in the lowest average processor load latency. As the stride grows from 1 to 4, the number of requests served as cache hits decreases, thus causing the processor load latency to increase significantly. When the stride increases beyond 4, there is no degradation in access latency since all accesses are cache misses.

On the SGI Origin 2000, we find that length of the stride has little impact of the access latency when the data size is below 1MB. The minor increase in access latency is attributed to the L1 cache of 32 Kbytes with a line size of 32 bytes. When the stride grows, the number of misses served in the L1 cache (2-3 processor cycles) decreases and requests are satisfied in the L2 cache (within 8-10 processor cycles). This shows the impact of a multilevel cache hierarchy. Finally, note the L2 cache size for R10000 is 4MB. However we find that the access latency does not remain constant between 2MB and 4MB. This is attributed to the fact that as soon as more than 128 pages are accessed, we will require a TLB miss per page. With a 16K page size, the size of the contiguous memory (our array) addressable without TLB misses is 2 MB. Thus the access latency remains constant until 2MB and increases beyond this array size. As the stride length increases, the access latency increases and stabilizes at a stride length of 16, owing to the fact that the cache line size is 128 bytes. From the figure, we observe that the increase in access latency is not smooth when the array size increases beyond the cache size. For example, we find that the access latency increases at a rapid pace as the array size increases from 4 MB to 5 MB, while it remains relatively constant between 5 MB and 6 MB. In order to investigate

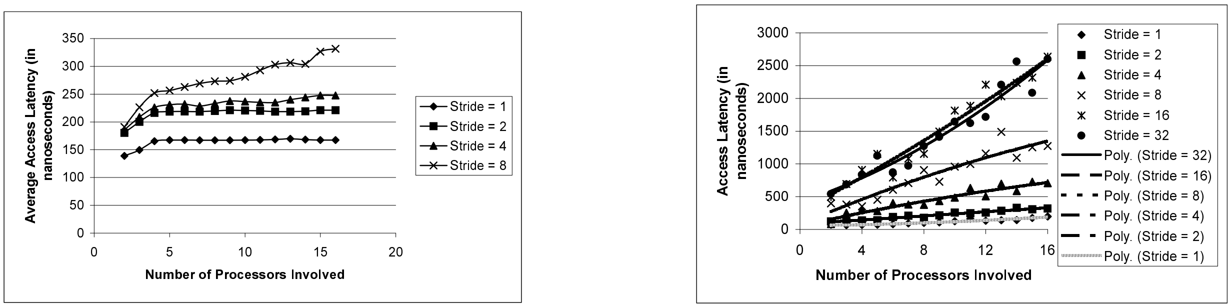
```
Iteratively perform
      One processor writes array A using
      either StoreUse (PC1) or StoreAll (PC2)
      Barrier Wait
      All other processors read array A using
      either LoadUse (PC1) or LoadAll (PC2)
      Barrier Wait
```

Figure 11: The Producer-Consumer (PC) Benchmark



(a) HP V-Class **(b) SGI Origin 2000**

Figure 12: Multiprocessor Performance: Single Outstanding Read-After-Write (change of scale)



(a) HP V-Class **(b) SGI Origin 2000**

Figure 13: Multiprocessor Performance: Multiple Outstanding Read-After-Write (change of scale)

this behavior, we used the hardware counters [14] on the system. We found that the number of additional L2 misses were roughly 17000 when the array size increased from 4 MB to 5 MB and roughly 7500 for an increase from 5 MB to 6 MB. This causes the unexpected behavior in the graphs obtained for the SGI Origin and depends heavily on several factors including associativity, organization, replacement policy and mapping conflicts between code/data blocks.

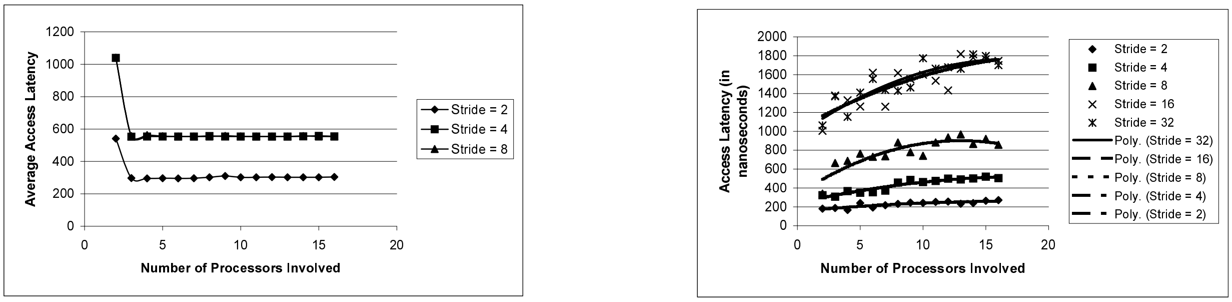
Finally, a comparison across multiprocessors leads to several observations. First, the latency of all accesses that result in a cache miss for an array size of 8MB is roughly 500 nanoseconds for the HP V-Class, whereas the SGI has an average latency of 350 nanoseconds. Second, when the strides are low, indicating high spatial locality, the larger L2 cache line size of the R10000 L2 cache helps by providing several cache hits. For linear array traversals, the performance of the R10000 processor is hindered by the number of TLB entries and not by the cache size for our experiments. This can be remedied by using a larger page size ($\geq 32KB$), supported by the IRIX operating system.

Figure 6 presents the data gathered on the HP V-Class (Figure 6(a)) and the Origin 2000 (Figure 6(b)) using the Store-Use microbenchmark. Note that results with a stride of unit length are not shown for the stores, because the Store-Use benchmark requires a minimum stride of 2. The analysis of the obtained results is identical to the load results presented above. A minor increase in access latency is observed as compared to the load performance because each store requires a following load (resulting in a cache hit always) to maintain the dependency for the single outstanding criterion.

3.2 Multiple Outstanding Load/Store Performance

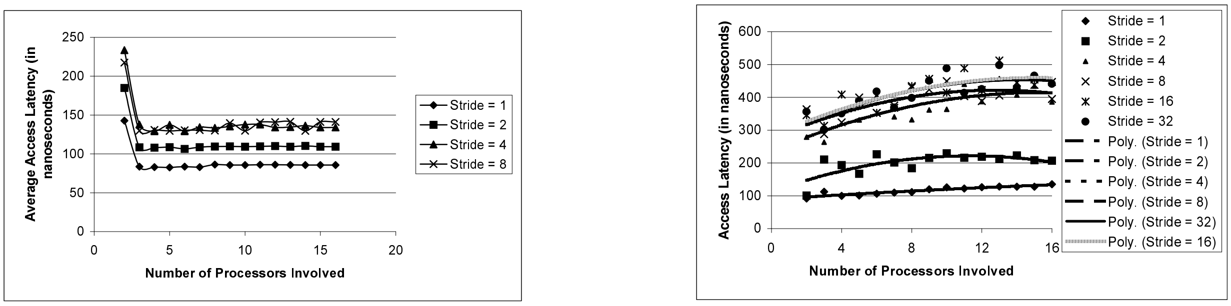
In this section, we identify the effects of multiple outstanding requests on the average access latency of processor loads and stores. The Load-All and Store-ALL benchmarks used for this experiment are in Figures 7 and 8 respectively. Unlike the Load-Use and Store-Use benchmarks, these benchmarks traverse an 8-byte element array of size N with no restriction on the ordering of the requests.

The results obtained from our Load-All and Store-All experiments are shown in Figure 9 and Figure 10 respectively. The access latency follows the same behavioral trend as seen earlier for Load-Use (Figure 5) and Store-Use (Figure 6). However, the measured average access latency is lower than that for the single outstanding load/store execution since much of the overall cache/memory access latency



(a) HP V-Class **(b) SGI Origin 2000**

Figure 14: Multiprocessor Performance: Single Outstanding Write-After-Read (change of scale).



(a) HP V-Class **(b) SGI Origin 2000**

Figure 15: Multiprocessor Performance: Multiple Outstanding Write-After-Read (change of scale).

gets overlapped. A comparison across these results will yield the amount of latency overlap that occurs. Consider the Load-Use and the Load-All results (a similar analysis of the store experiments can be derived). When the loads and stores result in cache hits (array size $<$ 2 MB), the ratio between the average access latency for single outstanding execution and that for the multiple outstanding execution is approximately 2 for both the HP V-Class and the SGI Origin 2000. This ratio depicts the amount of communication overlap experienced during the execution, with a higher ratio resulting in lower execution time and average access latency. For example, the measured ratio of 2 shows that the processors are capable of performing two loads resulting in cache hits simultaneously. On the other extreme, when all loads and stores result in cache misses, we find that the ratios become 4.5 and 1.4 for the V-Class and the Origin 2000 respectively. The low communication overlap obtained for the Origin 2000 may be attributed to the small page size, causing several TLB misses to form a part of the average cache miss latency.

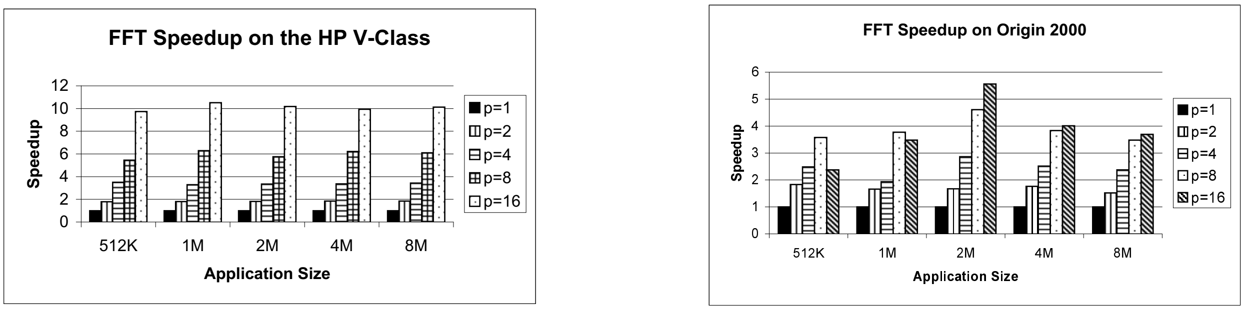
4 Multiprocessor Performance

In this section, our goal is to evaluate the multiprocessor performance of the two shared memory systems. Apart from the raw access latency analyzed in the previous sections, multiprocessor performance also depends highly on the hardware techniques used to maintain cache coherence. In this section, we use the commonly known sharing patterns, read-after-write (RAW) and write-after-read (WAR) to quantify multiprocessor performance since they depend highly on the coherence protocol employed. Typically, a WAR access (store to a block present in several caches) may take much longer to complete than a store to an unread block since the former requires invalidations. Similarly, a RAW access may take much longer than a load to a unmodified block since the former requires a data transfer from the owner's cache to the requester. We use the Producer-Consumer benchmark (PC) shown in Figure 11. In each iteration of this benchmark, the first phase involves the modification (store access) to a block while the second phase involves all other processors reading the modified block.

Results shown in Figures 12, 13, 14 and 15 are obtained for a 1MB array traversed using strides of different lengths. A 1MB array was chosen since it fits into the cache, keeping the latencies devoid of replacement effects. The first two figures are for RAW performance, while the latter two figures depict the results for WAR performance. The number of processors involved in the experiment was also varied to identify the effect of different degrees of sharing.

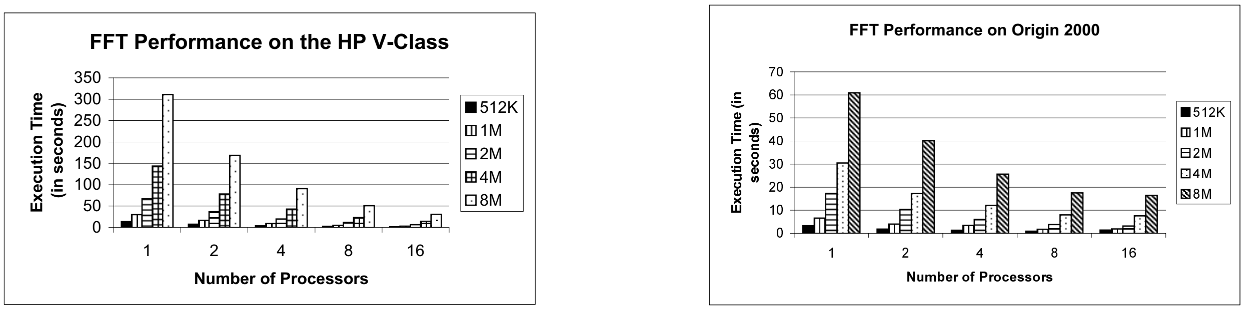
4.1 Read-After-Write Performance

We start with the results obtained for the HP V-Class. Figures 12(a) and 13(a) show the number of processors involved in the experiment on the x-axis and the average access latency in the y-axis. For example, when two processors are used in the experiment, we have one producer and one consumer, and the average read latency is approximately 1000 nanoseconds for a stride length of 4 elements. When we vary the stride length from 2 to 4, we find that the average access latency increases by a factor of two. When the stride is increased beyond 4, we see no impact on the average access latency since all loads result in cache misses. When there is only a single consumer, we find that the average load latency is roughly 1000 nanoseconds for reading the block from the producer's cache. Comparing this value with the max load access latency (from Figure 5) of 500 nanosec-



(a) HP V-Class **(b) SGI Origin 2000**

Figure 16: FFT Speedup Results



(a) HP V-Class **(b) SGI Origin 2000**

Figure 17: FFT Execution Time Results

onds, we find that the overhead of transferring data from the producer to the consumer is twice the latency of reading the block from the memory. As the number of consumers increases from 1 to 3, the latency increases significantly. Beyond three consumers, the latency remains constant. The reason for this behavior is the longer handshake required by the V-Class protocol to convert from exclusive to shared state. The first consumer obtains the block from the owner in exclusive state. When a subsequent processor requests this block, control information flows back to the memory from the first consumer and the memory sends the data to the second consumer to convert the block to shared state. This results in a longer latency for the second consumer.

The SGI Origin read-after-write (RAW) results are shown in Figures 12(b) and 13(b); note that a regression analysis has been used to fit a curve to these data points. Similar to the HP V-Class, the average access latency increases as the stride increases from 2 to 16, since 16*8 (128 bytes) is the size of the cache line. However, unlike the HP V-Class, as the number of processors involved in the experiment increases, the read-after-write latency increases. The queuing of memory requests or responses at the memory module as well as the network topology may be the reason for this increase in latency with increased number of consumers.

4.2 Write-After-Read Performance

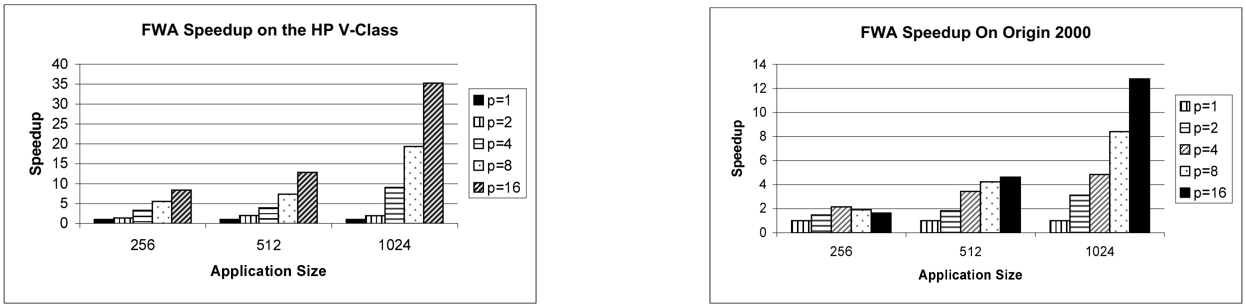
In this section, we start by analyzing the results for the HP V-Class. From Figures 14(a) and 15(a), we observe that the impact of stride length remains similar to the earlier results. We observe that when only one consumer participates in the experiment, the average latency is higher than that for multiple consumers. This is due to the migratory enhancement described in a previous subsection. When only one consumer participates in the experiment, it invalidates the owner's copy of the block and obtains an exclusive copy of the block. When the previous owner subsequently modifies the block, it has to regain ownership from this consumer. Since the consumer only holds exclusive access, the data transfer has to be done by reading memory after it invalidates the exclusive copy of the block. When more than one consumer is involved, the state of the block at the end of the consumer phase is *shared*. Since the state of the block is shared, only invalidations need to be sent to the consumers. With the non-blocking crossbar, the time taken to send invalidations seems to remain independent of the number of processors involved in the experiment.

Figures 14(b) & 15(b) show the results of the write-after-read experiments on the SGI Origin 2000. For the SGI Origin 2000, we note that the overhead of invalidation increases as the number of consumers involved in the computation increases. Unlike the HP V-Class, there are no peculiarities in the average access latency for any single case.

Finally, a comparison across multiprocessors shows that if the stride is closer to 1, then the latencies are more similar. For example, when the stride is equal to 2, the access latency for the HP V-Class is approximately 300 nanoseconds when the number of consumers is more than 1. Similarly the average access latency for the SGI Origin 2000 ranges between 200 and 250 nanoseconds. When the stride increases higher than 8, however, the SGI Origin 2000 pays a higher penalty for communication. This communication overhead is hidden by the 128-byte cache line fetch when the stride is low, and exposed when all accesses end up as cache misses.

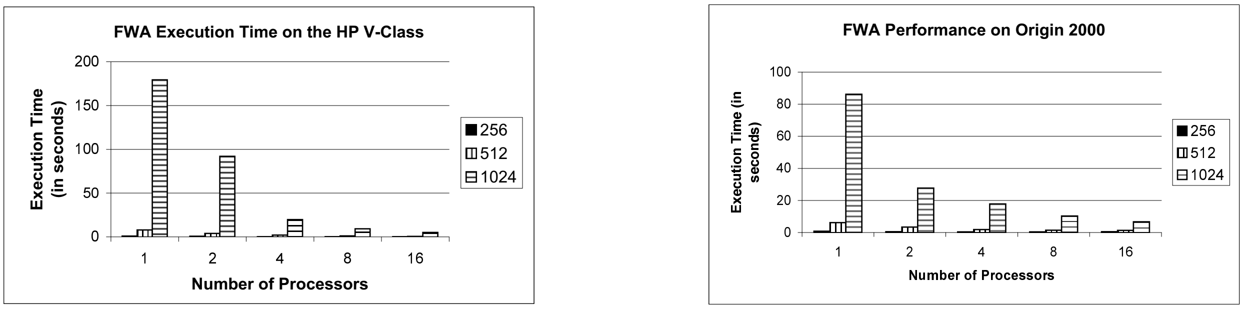
5 User-Level Performance

In the previous sections, we used microbenchmarks to study the impact of uniprocessor and multiprocessor access pat-



(a) HP V-Class **(b) SGI Origin 2000**

Figure 18: FWA Speedup Results



(a) HP V-Class **(b) SGI Origin 2000**

Figure 19: FWA Execution Time Results

terns. In this section, we run two chosen scientific applications to identify their user level performance.

FFT: The Fast Fourier Transform (FFT) application implements the Cooley-Tukey 1D FFT algorithm based on the several (logN) stages of butterfly computation. The input to the application is the number of points and the number of processors to be used for this computation. The execution flows as follows. The first $\log(N/P)$ stages are performed locally at each processor each with a data set of size N/P. The remaining $\log(P)$ stages require processors to exchange data in pairs. In each stage, one processor reads the data computed in the previous stage by a different processor.

FWA: The Floyd Warshall's algorithm implements an all-pairs shortest path algorithm to compute the shortest path between two nodes on a graph. The input to this algorithm is the number of vertices (or nodes) in the graph and the weights on the edges (specified in a input file "weight.mat"). The execution requires N (number of vertices) iterations. Within each iteration, every processor reads a single row of an adjacency matrix and updates a distance matrix to keep track of the distance between pairs of vertices on the graph. Thus in each iteration, the principle access pattern is a multiple processor RAW access pattern. The number of processors involved in the RAW access pattern heavily depends on the input graph.

Figures 16 and 17 present the observed speedup and execution time, respectively, of the FFT application under varied input data sizes and varied number of processors on the HP V-Class and the SGI Origin 2000. The application size is varied from 512K points to 8M points, with 32 bytes of data maintained per block. The overall data size of the application varies from 16 Mbytes to 256 Mbytes. The number of processors are varied from 1 to 16. The y-axis shows the obtained speedup and execution time.

The results on the HP V-Class show good speedup improvements from a 2-processor execution (a speedup of 2) to a 16-processor execution (a speedup of approximately 10). A value of 10 for speedup with 16 processors leads us to believe that the communication cost to computation cost ratio in the HP V-Class is relatively low. We observe that the V-Class speedup obtained does not depend on the size of data input. However, when we measured the speedup obtained from the SGI Origin, we found that the relative speedup over sequential execution is much lower. For example, the maximum speedup is roughly 5.5 with 16 processors for a data size of 64 Mbytes. This suggests that communication to computation ratio is much higher than that for the HP V-Class. Finally, for 512K and 1M points, the speedup decreases beyond 8 processors, implying that communication overhead negates the gain from parallel computation .

We next look at the execution time results obtained in Figure 17. While the Origin 2000 had poor speedup ratios, it should be noticed that the raw execution time is lower than on the HP V-Class. The execution time for 2M points ranges from 66 to 6 seconds for the HP multiprocessor, and from 17 to 3 seconds for the SGI multiprocessor. Much of this difference in execution time is due to optimized uniprocessor access patterns on the SGI Origin. As we have seen in Section 3, the SGI load/store latencies were much lower than the HP load/store latencies due to a larger cache size and line size. From a qualitative perspective, the migratory optimization to the HP V-Class coherence protocol affects the performance of this benchmark negatively since it involves mostly two-processor sharing. The SGI protocol optimization has no performance impact since speculative sharing can be rarely used during FFT execution.

Figures 18 and 19 present the observed speedup and execution time, respectively, of the FWA application under varied input data sizes and varied number of processors on the HP V-Class and the SGI Origin 2000. The size of the application is varied on the x-axis from 256 vertices to 1024 vertices. The FWA application stores two dimensional arrays, a distance matrix and an adjacency matrix, each element of which requires 8 bytes and 4 bytes of space respectively. Thus the overall data size of the application varies from 768 Kbytes to 12 Mbytes. The number of processors (shown in the legend) are varied from 1 to 16. The y-axis shows the obtained speedup and the execution time respectively.

From Figure 18(a), we find that the speedup gained on the HP V-Class multiprocessor ranges from roughly 9 (for 256 vertices) to approximately 35 (for 1024 vertices) using 16 processors. One of the main causes for the superlinear speedup with 16 processors is the data size. When the data fits into the cache (768 Kbyte for example), the uniprocessor performance is good due to a higher cache hit ratio. On the other hand, with a data size of 12 Mbytes, the working set overcomes the cache size, causing cache misses. In such cases, using multiple processors increases the overall cache space available and thus improves the access latency by over a magnitude. We also observe that speedups on the SGI Origin 2000 are lower than on the V-Class, similar to the FFT execution. For a 256-vertex graph, four processors seem to be the optimal number, with performance deterioriating beyond this threshold. On the other hand, with 1024 vertices, the speedup increases at a steady pace.

We next analyze the execution time for the FWA application. Like the FFT application results, the uniprocessor execution time is much higher on the HP V-Class than on the SGI Origin. However, as more processors are used, the HP V-Class execution time improves superlinearly beyond the SGI Origin execution time. For example, with 1024 vertices and 16 processors, the execution time of the HP V-Class is approximately 13% lower than the SGI Origin execution time. On the SGI Origin 2000, we had seen in Section 4 that the overhead of invalidation and multiple consumer RAW increased linearly, and that as more processors are used the RAW and WAR latency of the Origin tends to be much higher than the HP V-Class. This reduces the effect of all uniprocessor access pattern benefits on the SGI Origin 2000. From this result, we may infer that the HP V-Class offers better performance than the Origin 2000 for applications with high read sharing degrees.

6 Summary and Conclusions

In this paper, we employed microbenchmarks and scientific applications to study the impact of several architectural, technological and protocol choices used in the HP V-Class and SGI Origin 2000 multiprocessors.

In our suite of microbenchmarks, we used uniprocessor load/store benchmarks to identify the impact of cache size, cache line size, outstanding misses, TLB and page size. We found that in the presence of only single outstanding load/store requests, the SGI Origin 2000 experienced average latencies that were approximately 25% to 50% lower than the HP V-Class. However, when multiple requests are active in the system, the V-Class's performance surpasses the Origin performance with average latencies about 55% to 60% lower than the Origin 2000. We studied the impact of coherence protocol optimizations on access latencies of dominant multiprocessor access patterns like read-after-write (RAW) and write-after-read (WAR). As a result, we learned that the HP V-Class's latencies remain relatively constant as more processors become active, whereas the Origin 2000's latencies grow in a somewhat linear fashion.

Two scientific applications, with different dominant access patterns, were employed to study the user-level performance of these multiprocessors. Using applications, we took into account issues such as synchronization and process/thread creation that affect speedup and execution time considerably. We found that the HP V-Class obtains better speedups than the Origin 2000. However, we did find that the execution times of the Origin 2000 were superior in most of the cases. The poor speedups can be attributed to better optimized uniprocessor execution on the Origin 2000.

Future work in this area includes expanding our microbenchmark suite and the scientific application suite to cover a wider range of characteristics. Correlating the performance of applications to the latencies observed using the microbenchmarks is also currently underway. While we exercised only a single UMA node of the HP V-Class system in this paper, we intend to extend our study to a larger HP V-Class system which will introduce NUMA effects.

References

[1] G. Abandah and E. Davidson, "Characterizing Distributed Shared Memory Performance: Case Study of the Convex SPP1000," IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 2, Feb. 1998.

[2] G. Abandah and E. Davidson, "Effects of Architectural Trends and Technological Advances on the HP/Convex Exemplar's Memory and Communication Performance," 25th Annual International Symposium on Computer Architecture, pp. 318-329, June 1998.

[3] L. M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Transactions on Computers*, vol. C-27, no. 12, pp. 1112–1118, December 1978.

[4] HP RISC Precision Architecture 2.0 (PA-RISC 2.0) Document, Hewlett-Packard Corporation, http://www.hp.com/ahp/framed/technology/micropro/

[5] HP 9000 V-Class Server Architecture Document, 2nd Edition, Hewlett-Packard Corporation, http://docs.hp.com:80/hpux/systems/#vclass.

[6] C. Hristea and D. Lenoski, "Measuring Memory Hierarchy Performance of Cache-Coherent Multiprocessors Using Micro Benchmarks," Proceeding of Supercomputing: High Performance Networking and Computing, 1997.

[7] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," Proc. of the 24th International Symposium on Computer Architecture, pp. 241-251, May 1996.

[8] D. Lenoski et al., "The Stanford DASH Multiprocessor," IEEE Computer, 25(3), pages 63-79, March 1992.

[9] Parasol - HP V-Class Multiprocessor, PARASOL Lab, Department of Computer Science, Texas A&M University, http://www.cs.tamu.edu/research/parasol.

[10] R. Saavedra, R. Gaines and M. Carlton, "Characterizing the Performance Space of Shared Memory Computers Using Micro-Benchmarks," Technical Report # USC-CS-92-547, Department of Computer Science, University of Southern California, July 1993.

[11] K. Shaw and G. Astfalk, "Four State Cache Coherence Protocol in the Convex Exemplar System," http://www.hp.com/wsg/tech/technical.html.

[12] Titan - SGI Origin 2000, Supercomputing Center, Texas A&M University, http://anakin.tamu.edu/titan/

[13] K. Yeager. "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, 16(2):28-40, April 1996.

[14] M. Zagha, et al., "Performance Analysis Using MIPS R10000 Performance Counters," Proceedings of Supercomputing'96, Nov 1996.