International Journal of Parallel Programming, Vol. 33, No. 4, August 2005 (© 2005)
DOI: 10.1007/s10766-004-1187-0

An Experimental Evaluation of the HP V-Class and SGI Origin 2000 Multiprocessors using Microbenchmarks and Scientific Applications*

Ravi Iyer,[1] Jack Perdue,[2] Lawrence Rauchwerger,[2,4] Nancy M. Amato,[2] and Laxmi Bhuyan[3]

Received April 13, 2004; accepted October 29, 2004

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 multiprocessors. In this study, we present a detailed comparison of two architectures, the HP V-Class and the SGI Origin 2000. Our goal is to compare and contrast design techniques used in these multiprocessors. We

[1]Intel Corporation. E-mail: ravishankar.iyer@intel.com
[2]Parasol Laboratory, Department of Computer Science, Texas A&M University College Station, TX 77843-3112, USA. E-mail: {jkp2866, amato, rwerger}@cs.tamu.edu
[3]Department of Computer Science and Engineering, University of California Riverside, Riverside, CA 92521, USA. E-mail: bhuyan@cs.ucr.edu
[4]To whom correspondence should be addressed.
*A preliminary version of this paper appeared in the 13th ACM International Conference on Supercomputing (ICS'99).[13] This work was done while Iyer and Bhuyan were at Texas A&M. It was supported in part by a Hewlett-Packard Equipment Grant. Amato and Rauchwerger supported in part by NSF Grants ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACR-0081510, ACR-0113971, CCR-0113974, EIA-9810937, EIA-0079874, by the DOE ASCI ASAP program, and by the Texas Higher Education Coordinating Board grant ATP-000512-0261-2001. Perdue supported in part by a Dept. of Education Graduate Fellowship (GAANN).

present the impact of processor design, cache/memory hierarchies and coherence protocol optimizations on the memory system performance of these multiprocessors. We also study the effect of parallelism overheads such as process creation and synchronization on the user-level performance of these multiprocessors. Our experimental methodology uses microbenchmarks as well as scientific applications to characterize the user-level performance. Our microbenchmark results show the impact of Ll/L2 cache size and TLB size on uniprocessor load/store latencies, the effect of coherence protocol design/optimizations and data sharing patterns on multiprocessor memory access latencies and finally the overhead of parallelism. Our application-based evaluation shows the impact of problem size, dominant sharing patterns and number of processors used on speedup and raw execution time. Finally, we use hardware counter measurements to study the correlation of system-level performance metrics and the application's execution time performance.

KEY WORDS: Parallel architectures; performance analysis; shared memory.

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. Shared memory systems such as the SGI Origin 2000,(18) the HP V-Class(10) and the Sequent NUMA-Q,(20) multiprocessors are excellent examples of this trend.

As processor technology continues to rapidly advance, the comparatively slow improvement in memory speed is the main performance bottleneck in current and future shared memory multiprocessors. As a result, the use of larger caches and deeper cache hierarchies is becoming increasingly common. However, the impact of cache hierarchy on the cache/memory system performance is largely dependent on the application access pattern. The use of theoretical models such as the BSP(28) and LogP(5) is getting more and more difficult without significant extensions and extensive hardware performance measurements. Past performance measurement and analysis research(1,2,11,12,16,17,22,23,25) either used a microbenchmark-based or an application-based evaluation methodology. Furthermore, these studies focus on understanding the performance of a single multiprocessor. In this paper, our aim is to provide an in-depth understanding of the memory system performance of two multiprocessors, the HP V-Class and the SGI Origin 2000, using both evaluation methodologies. As a result, we compare and contrast design techniques used in these systems and also correlate observations between microbenchmark studies and application-based evaluation.

Our microbenchmark study illustrates 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. 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. We compare the effect of two layers of caching on the SGI Origin 2000 to a single level of cache on the HP V-Class. We show how a small translation lookaside buffer (TLB) on the SGI Origin 2000 can affect data access latency for a given configuration of page size. When comparing the two systems, we find that the uniprocessor load and store latencies measured on the HP V-Class tend to be 40–45% lower than those on the SGI Origin 2000. Using several multiprocessor benchmarks, we study the impact of sharing degree on the access latency of dominant sharing patterns. We observe that an increase in the sharing degree has relatively no impact on the HP V-Class latency, whereas it significantly increases the average latency experienced on the SGI Origin 2000. We also study the parallelism overhead using microbenchmarks that measure the latency for process creation and synchronization. We show that the amount of overhead differs significantly in the two systems and is heavily dependent on the mechanisms provided in the operating system.

Unlike microbenchmarks(1,2) that are targeted at evaluating the performance of particular access patterns, the use of several scientific applications(16,17) helps us understand how these individual factors affect overall user-level performance. In our application-based evaluation of the HP V-Class and the SGI Origin 2000, we use five representative scientific applications. We vary the degree of parallelism and the problem size to analyze application speedup and system execution time. We observe that the SGI Origin 2000 experiences lower execution time than the HP V-Class. However, we also observe that the HP V-Class generally exhibits better speedups. We correlate these observations using the known dominant sharing patterns of the applications to the performance of the respective microbenchmarks. Using these correlations, 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. The two different methodologies used in this paper provide an understanding regarding the performance of the two systems.

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 and parallelism overhead. Section 5 uses scientific applications for a user-level performance comparison of these memory systems and presents some insights to their performance using CPU hardware counts of instructions, cache misses and TLB misses. Finally, Section 6 summarizes the paper and presents a direction for future work.

2. SYSTEM OVERVIEW

In this paper, we study the performance of two commercial shared memory multiprocessors, the HP V-2200(10) running HIP-UX 11.0 and the SGI Origin 2000(18) running IRIX 6.5. While these systems scale up to or beyond 512 processors, we chose a 16-processor configuration due to its availability.(24,27)

2.1. System Architectures

A block diagram of the 16-processor HP server architecture is shown in Fig. 1. The symmetric multiprocessor design utilizes Exemplar Processor Agent Controllers (EPAC), each with its own Exemplar PCI Interface Controllers (EPIC - not shown) controlling an independent PCI bus and eight Exemplar Memory Access Controllers (EMAC). These controllers are connected centrally by a HP Hyperplane crossbar comprised of four Exemplar Routing Attachment Controllers (ERACs). The 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 MB of instruction cache and 2 MB of data cache. Each cache is direct mapped and has dual ports. 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 Fig. 1.

A block diagram of a 16-processor SGI multiprocessor is shown in Fig. 2. The SGI is a cache-coherent non-uniform memory access

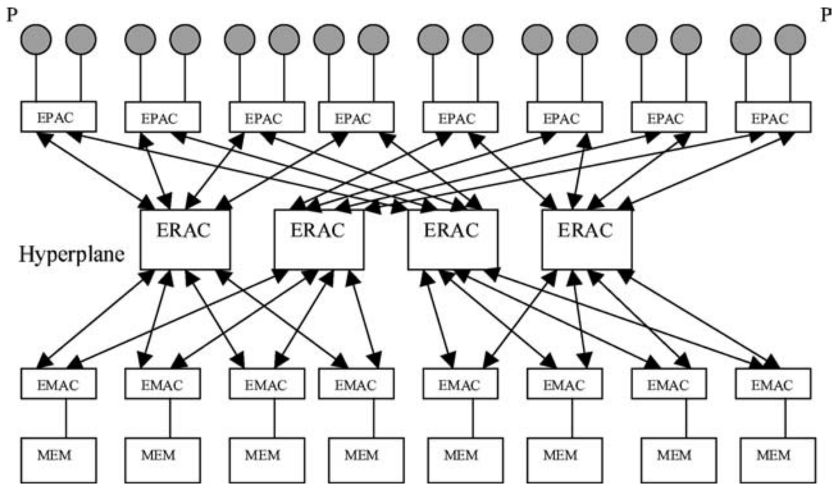


Fig. 1. 16-processor HP V-Class system.

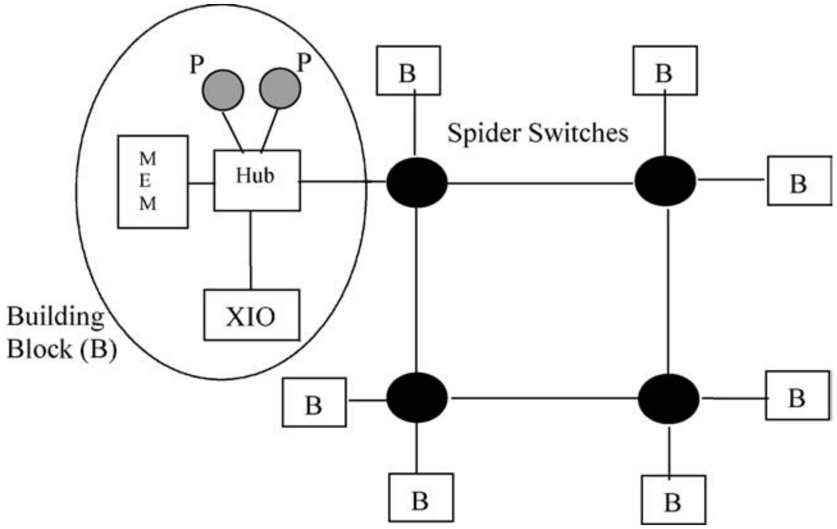


Fig. 2. 16-processor SGI Origin 2000 system.

(CC-NUMA) multiprocessor based on dual-processor basic building blocks. Each block contains two 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 250 MHz with 32 KB of L1 cache and 4 MB of unified L2 cache. The L1/L2 cache line sizes are 32 and 128 bytes, respectively. Both caches are 2-way set associative using an LRU replacement policy. 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 6×6 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.

2.2. Experimental Methodology

Several differences can be immediately observed when studying the 16-processor HP and the SGI architectures (Table I). A fundamental architectural difference is memory organization: the HP 16-processor node is a UMA multiprocessor with all the memories equidistant from the processors, while the 16-processor SGI exhibits a non-uniform memory access (NUMA) memory organization with a local memory and several remote memories. In this work, we limit ourselves to the performance character-

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

	HP V-Class	SGI Origin
Processor		
Speed	200 MHz	250 MHz
ILP Behavior	4-way	4-way
Cache Size	2 MB	32 KB/4 MB
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*2 entries
Coherence	MES-like	MESI-like
(optimization)	(migratory)	(speculative)
Scalability		
Node	16-way	2-way
Topology	crossbar	hypercube

ization of the cache/memory system and the parallelism overhead using microbenchmarks and scientific applications.

Our microbenchmark suite can be classified into three main categories and various sub-categories as listed below. 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 those of Abandah and Davidson.(2)

- *Uniprocessor Data Access*: We use load and store traversals through an array of length N. Furthermore, we also impose dependencies on the consecutive accesses to study the effectiveness of latency hiding through multiple outstanding cache transactions.
- *Multiprocessor Data Access*: We use microbenchmarks designed for producer–consumer data sharing patterns such as read-after-write and write-after-read to study the impact of sharing degree (number of consumers) on the coherence overhead of invalidation for producers and reading dirty data for consumers.
- *Parallelism Overhead*: We use microbenchmarks designed to study the overhead of process creation and synchronization using system calls provided in the operating system.

Our application-based evaluation is based on experimental runs of five scientific applications on the two multiprocessors. These applications were developed at Texas A&M University by the Parasol Laboratory.(24) The applications are Gaussian Elimination (GE), transitive closure of a matrix (TC), matrix multiplication (MM), Fast Fourier Transform (FFT) and Floyd Warshall's all pair shortest-path algorithm (FWA). We analyze results in terms of speedup and execution time by varying the problem size and the number of processors. In addition, we performed a detailed case study of two of the applications (FFT and FWA). We instrumented them to collect performance counter measurements such as cycles per instruction (CPI), cache performance and TLB statistics, Based on these hardware counter measurements and the microbenchmark data, we attempt to explain the different speedup characteristics of these applications.

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 Ref. 1 to compare the performance of the HP and the SGI multiprocessors.

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

Fig. 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];
```

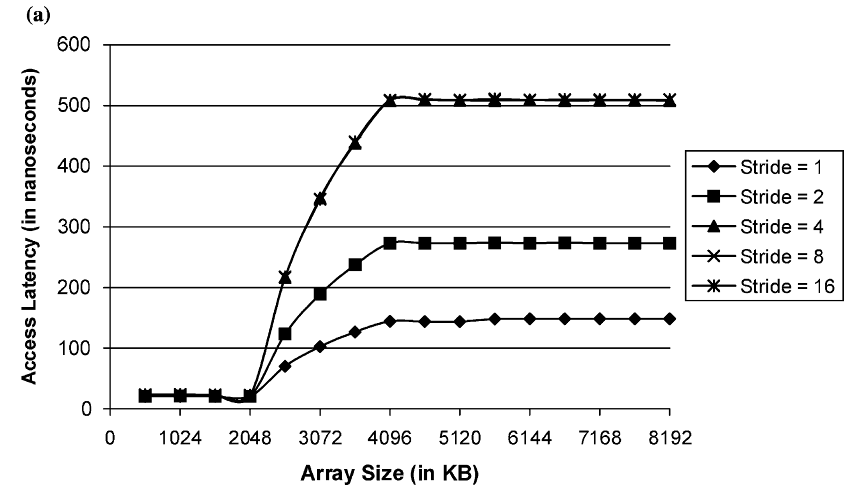
Fig. 4. The store-use benchmark.

3.1. Description of the Microbenchmarks

We characterize the latency of uniprocessor loads and stores using the Load-Use (Fig. 3) and Store-Use (Fig. 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[29] and the PA-8200,[9] 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. In order to study the impact of multiple outstanding misses, we modified the Load-Use and Store-Use benchmark. These modified benchmarks, Load-All and Store-All, are shown in Figs. 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.

3.2. Single Outstanding Load/Store Performance

Figure 5 presents the data gathered on the HP (Fig. 5(a)) and the SGI (Fig. 5(b)) using the Load-Use benchmark. On the HP multiprocessor, we find that the length of the stride does not impact the latency when the



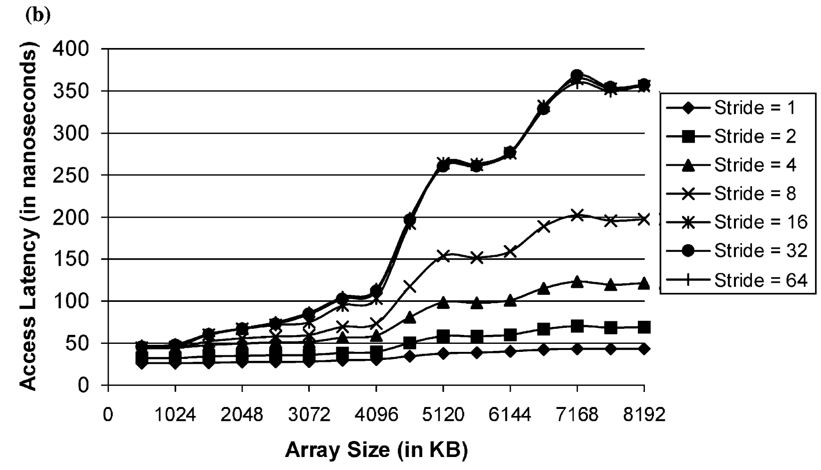
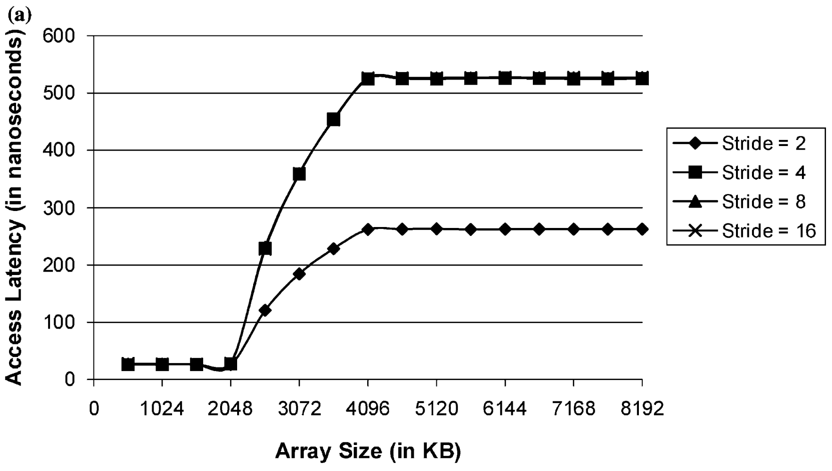


Fig. 5. Uniprocessor performance: single outstanding loads (note scale). Latencies on the HP for strides = 4/8/16 are roughly equivalent (a) HP V-Class (b) SGI Origin 2000.

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 three processor cycles). The average latency was found to be approximately 20 ns. 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, we find that length of the stride has little impact on the access latency when the data size is below 1 MB. The minor increase in access latency is attributed to the L1 cache of 32 KB 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 4 MB. However we find that the access latency does not remain constant between 1 and 4 MB. This is attributed to the fact that the R10000 TLB can serve up to 64 data page translations at an instance of time. With a 16 KB page size, the size of the contiguous memory (our array) addressable by the TLB is 1 MB. Thus the access latency remains constant until 1 MB 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 to 5 MB, while it remains relatively constant between 5 and 6 MB. In order to investigate this behavior, we used the hardware counters(14,15) on the system. We found that the number of additional L2 misses were roughly 17000 when the array size increased from 4 to 5 MB and roughly 7500 for an increase from 5 to 6 MB. This causes the unexpected behavior in the graphs obtained for the SGI and depends on several factors including associativity, organization, replacement policy and mapping conflicts between code and 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 8 MB is roughly 500 ns for the HP whereas the SGI has an average latency of 350 ns. 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 ($\geqslant$ 32 KB), supported by the IRIX operating system.



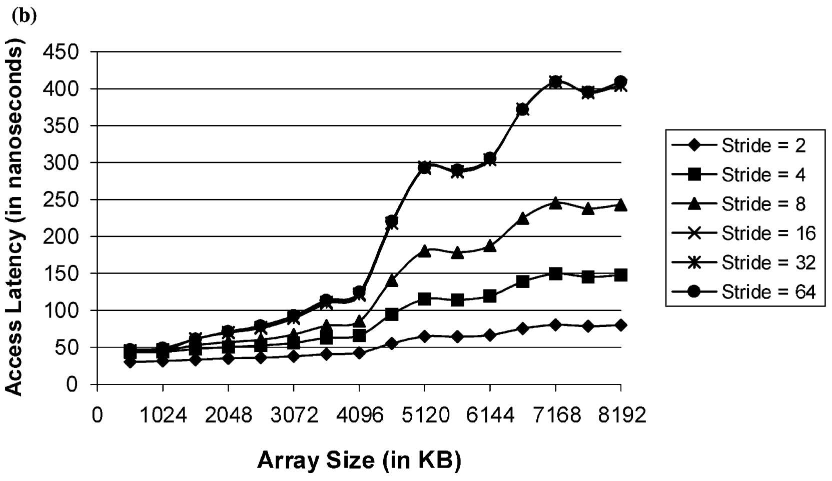


Fig. 6. Uniprocessor performance: single outstanding stores (note scale). Latencies on the HP for strides = 4/8/16 are roughly equivalent. (a) HP V-Class (b) SGI Origin 2000.

Figure 6 presents the data gathered on the HP (Fig. 6(a)) and the SGI (Fig. 6(b)) using the Store-Use microbenchmark. Note that results with a stride of one 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.

3.3. 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 by using the Load-All and Store-All benchmarks shown in Figs. 7 and 8, respectively.

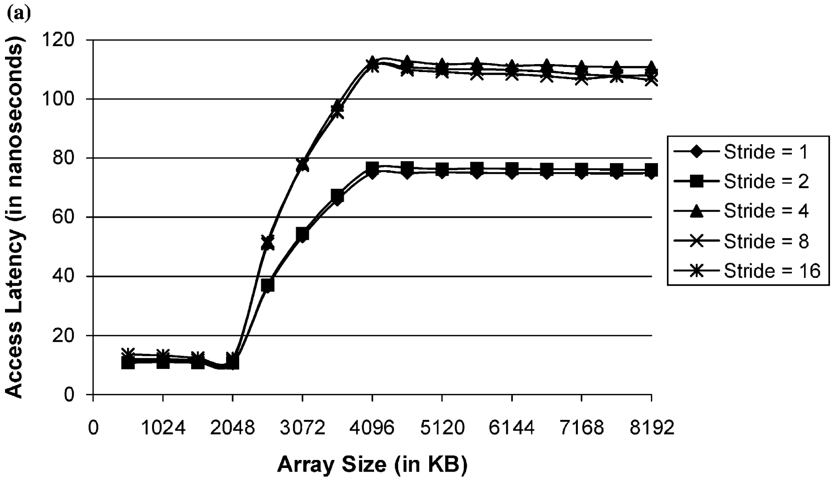
The results obtained from our Load-All and Store-All experiments are shown in Figs. 9 and 10, respectively. The access latency follows the same behavioral trend as seen earlier for Load-Use (Fig. 5) and Store-Use (Fig. 6). However, the measured average access latency is lower than that for the single outstanding load/store execution since much of the overall cache/memory access latency 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 two for both the HP and the SGI This ratio depicts the amount of communication overlap experienced during the execution, with a higher ratio resulting in lower execution time and lower 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 HP and the SGI, respectively. The low communication overlap obtained for the SGI may be attributed to the small page size, causing several TLB misses to form a part of the average cache miss latency.

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

Fig. 7. The load-all benchmark.

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

Fig. 8. The store-all benchmark.



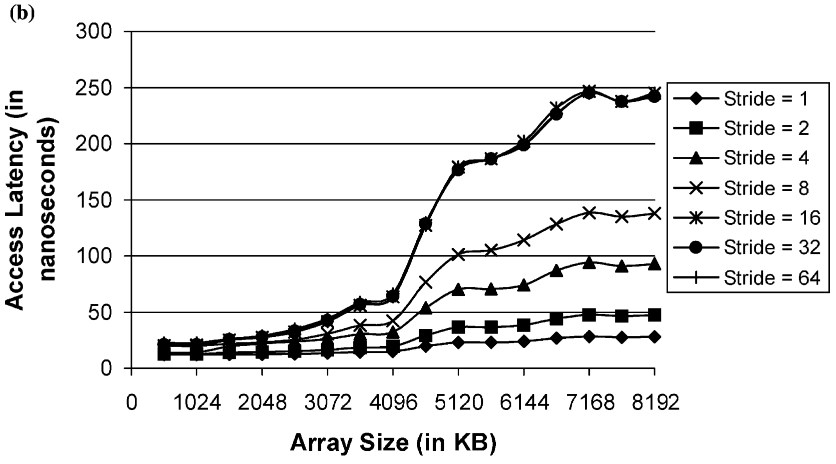
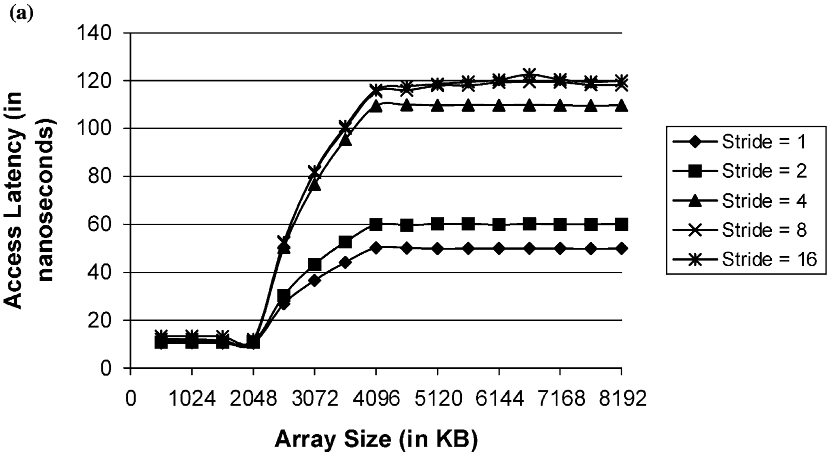


Fig. 9. Uniprocessor performance: multiple outstanding stores (note scale). Latencies on the HP for strides = 4/8/16 are roughly equivalent as are those for strides 1 and 2. (a) HP V-Class (b) SGI Origin 2000.

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, memory system performance in a multiprocessor system also depends highly on the hardware techniques used to maintain cache coherence. In Section 4.1, we present an overview of

(a)



(b)

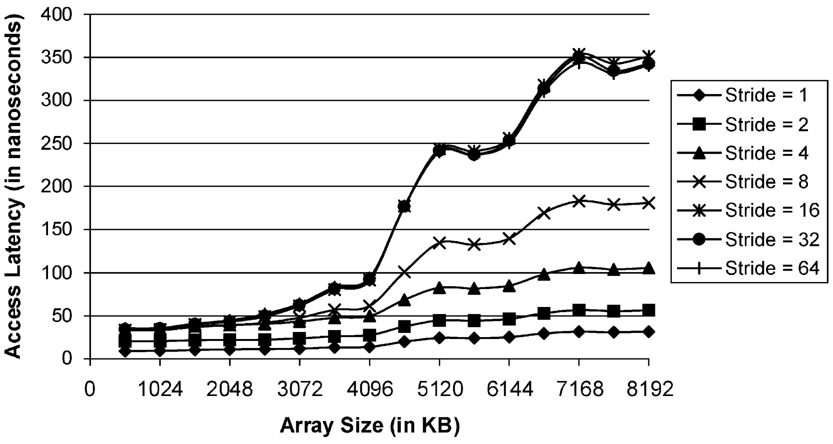


Fig. 10. Uniprocessor performance: multiple outstanding stores (note scale). Latencies on the HP for strides 8 and 16 are roughly equivalent. (a) HP V-class (b) SGI Origin 2000.

coherence protocols used in the HP and the SGI. Subsequently we present a detailed description of the microbenchmarks and finally the performance analysis based on each of the microbenchmark experiments.

4.1. Overview of Coherence Protocols

Both the HP(26) and the SGI(18) use a variation of the invalidate-based full-map directory protocol(4) and the well-known MESI cache protocol.(19) 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 HP 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 SGI protocol,(18) 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 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 HP protocol, in both cases, the owner retains the block in shared state in the cache.

4.2. Description of the Microbenchmarks

For our multiprocessor memory system evaluation, we use the commonly known sharing patterns, read-after-write (RAW) and write-after-read (WAR) to quantify multiprocessor performance because they are indicative of coherence protocol performance. Typically, a WAR access (store to a block present in several caches) takes longer to complete than a store to an unread block since the former requires invalidations. Similarly, a RAW access may take 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

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

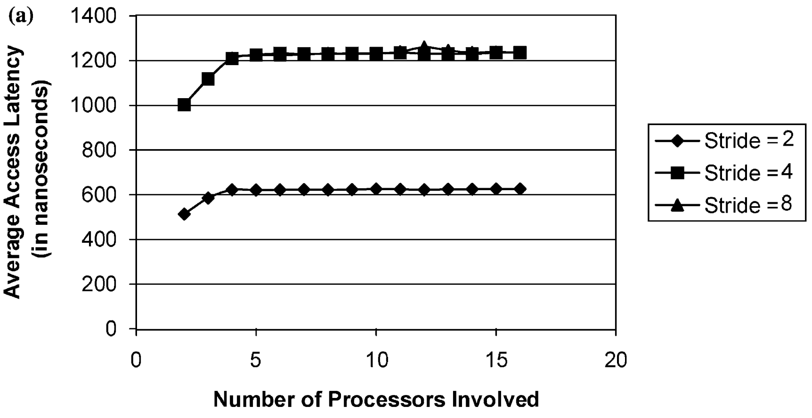
Fig. 11. The producer-consumer (PC) benchmark.

Fig. 11 to study RAW and WAR latencies. 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. Finally, we also analyze the overhead of parallelism, namely process creation and synchronization, in Section 4.5.

Results shown in Figs. 12–15 are obtained for a 1 MB array traversed using strides of different lengths. A 1 MB 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 for RAW/WAR sharing to identify the effect of different degrees of sharing. The results are analyzed in the following sections.

4.3. Read-After-Write Performance

We start with the results obtained for the HP. 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 ns for a stride length of four 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 ns for reading the block from the producer's cache. Comparing this value with the max load access latency (from Fig. 5) of 500 ns, we find that the overhead of transferring data from the producer to the consumer is twice the latency of reading the block from



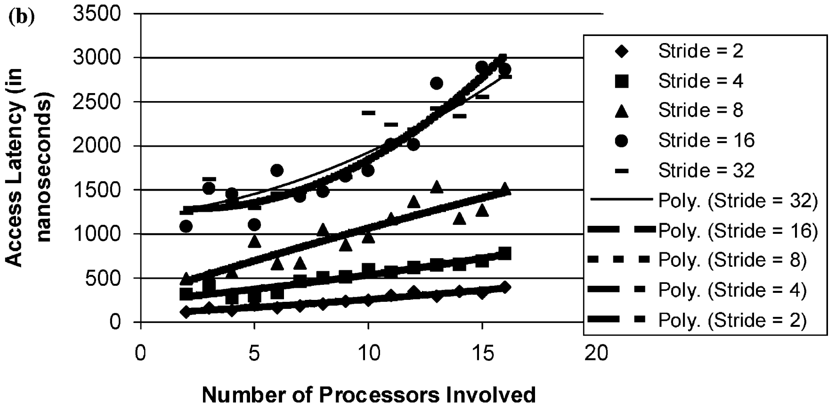
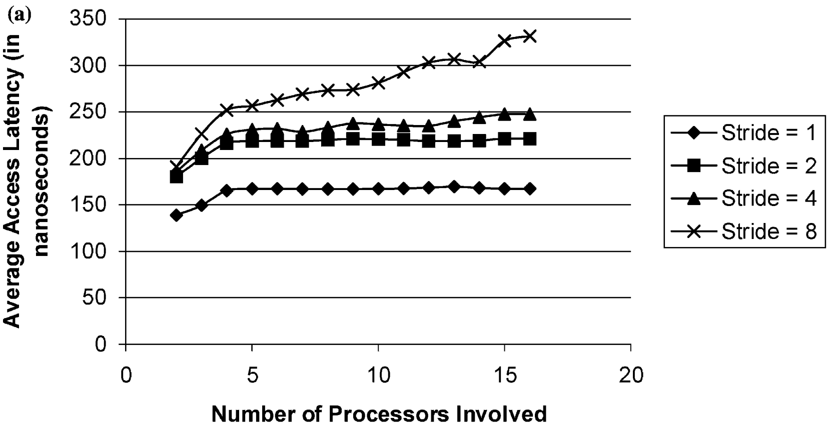


Fig. 12. Multiprocessor performance: single outstanding read-after-write (note scale). Latencies on the HP for strides 4 and 8 are roughly equivalent. (a) HP V-class (b) SGI Origin 2000.

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 HP 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.



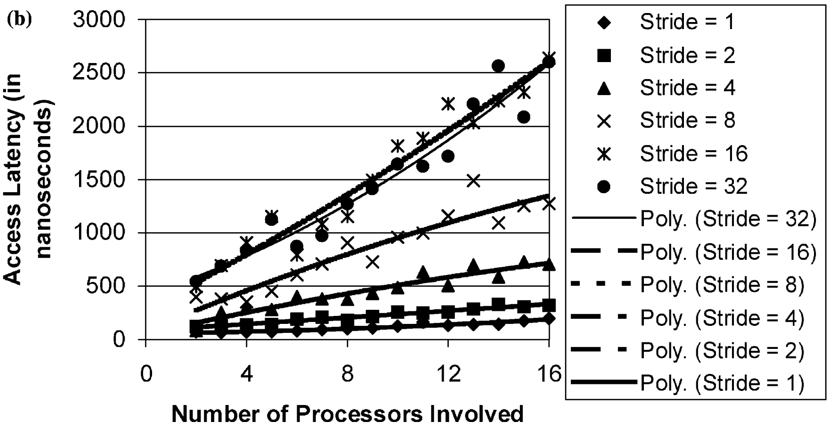
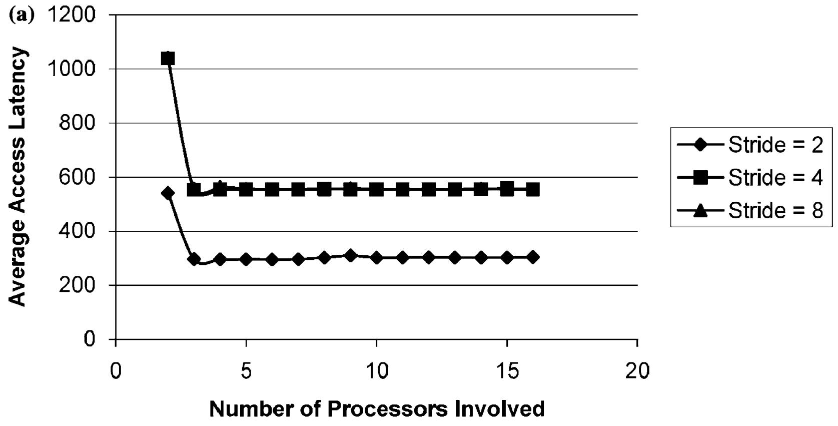


Fig. 13. Multiprocessor performance: multiple outstanding read-after-write (note scale). (a) HP V-class (b) SGI Origin 2000.

The SGI RAW results are shown in Figs. 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 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, as the number of processors involved in the experiment increases, the RAW 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 an increased number of consumers.

4.4. Write-After-Read Performance

In this section, we start by analyzing the results for the HP. From Figs. 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. 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



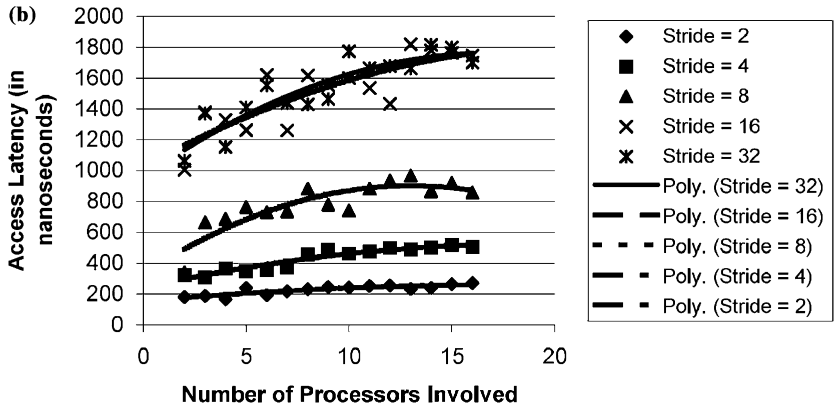
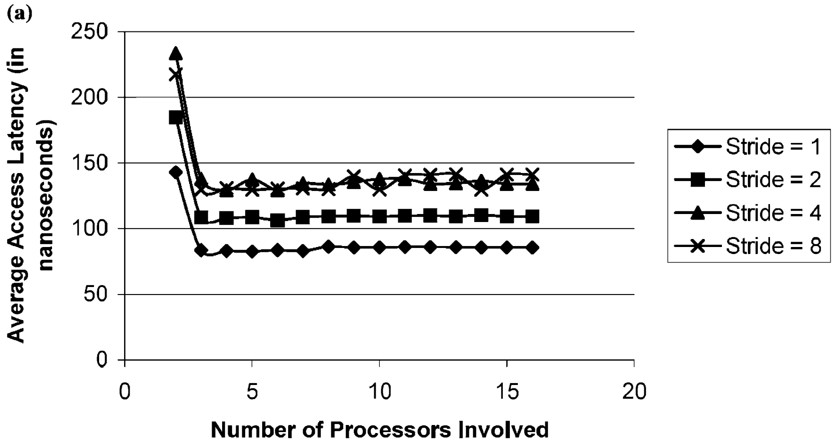


Fig. 14. Multiprocessor performance: single outstanding write-after-read (note scale). Latencies on the HP for strides 4 and 8 are roughly equivalent. (a) HP V-class (b) SGI Origin 2000.



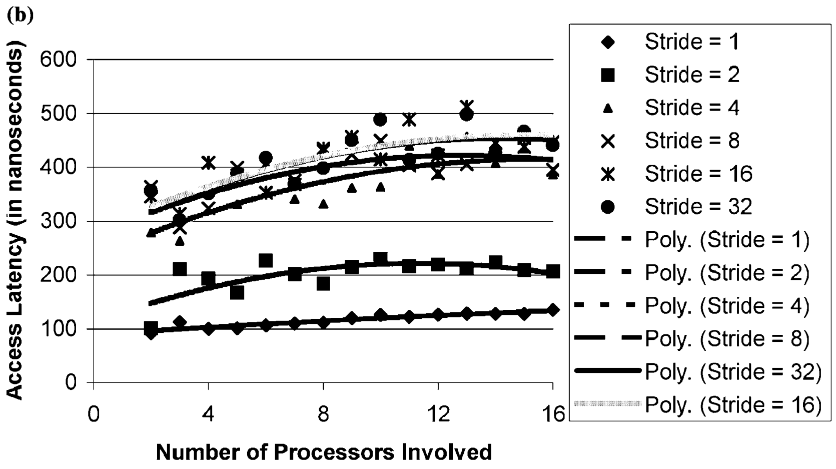


Fig. 15. Multiprocessor performance: multiple outstanding write-after-read (note scale). (a) HP V-class (b) SGI Origin 2000.

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) and 15(b) show the results of the WAR experiments on the SGI. For the SGI, we note that the overhead of invalidation increases as the number of consumers involved in the computation increases. Unlike the HP, there are no peculiarities in the average access latency for any single case.

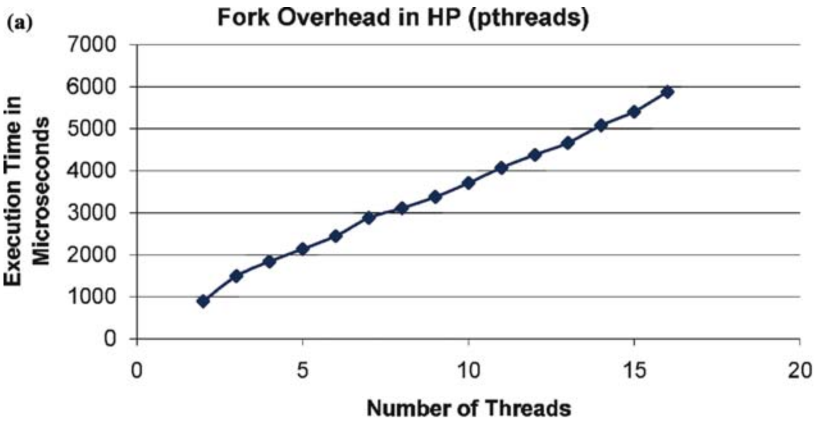
Finally, a comparison across multiprocessors shows that if the stride is closer to 1, then the latencies are more similar. For example, when the stride is equal to 2, the access latency for the HP is approximately 300 ns when the number of consumers is more than 1. Similarly the average access latency for the SGI ranges between 200 and 250 ns. When the stride increases higher than 8, however, the SGI 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.

4.5. The Overhead of Parallelism

The two major types of overhead immediately noticeable in a parallel program are process creation and synchronization. The overhead due to process creation can be substantial and affect application performance considerably. In order to minimize the impact of process creation on application performance, most recent multiprocessors, including the HP and the SGI, provide mechanisms to keep all created processes alive to be re-used through the execution of the application. For example, the *m_fork* routine on the SGI incurs the process creation overhead only the first time it is called. After each phase of execution, the process is put to sleep and thus subsequent *m_fork* calls re-use the sleeping threads. However, note that processes can be freed at any stage using the *m_kill_procs* call on the SGI routine for applications that require this feature.

In this section, we analyze measured results to quantify the overhead of process creation and barrier synchronization. The obtained results are shown in Figs. 16 and 17, respectively. Our intent is to study the impact of scalability on parallelism overhead and not to present concrete figures since these depend heavily on the mechanism employed on the multiprocessor. For example, we employ the *pthreads* package on the HP and use a user-level barrier synchronization mechanism that is based on the use of an array data structure. On the other hand, applications ported to the SGI employ the *m_sync* system call for batter synchronization.

From the figures, we find that both forking processes, as well as synchronization, are two magnitudes more expensive on the SGI than on the HP. However, note that the Origin 2000 suffers significantly when the number of processors employed increases beyond 4 (beyond a single router). We can expect a similar phenomenon when employing more than 16 processors (a NUMA configuration) on the HP. We also note that on both systems, the overhead of parallelism depends linearly on the number of threads/processes employed.



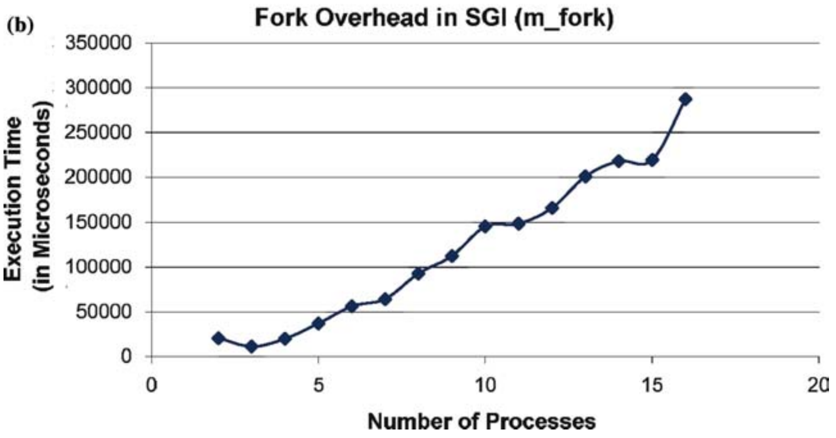


Fig. 16. Multiprocessor fork performance (note scale). (a) HP V-class (b) SGI Origin 2000.

5. USER-LEVEL PERFORMANCE

In the previous sections, we used microbenchmarks to study the impact of uniprocessor and multiprocessor access patterns. In this section, we use five scientific applications to study their user level performance. As mentioned in section 2.2, the chosen applications are Guassian Elimination (GE), matrix multiplication (MM), transitive closure of a matrix (TC), Fast Fourier Transform (FFT), and Floyd-Warshall's all pair shortest-path algorithm (FWA). These applications were developed in house at Texas A&M.(24)

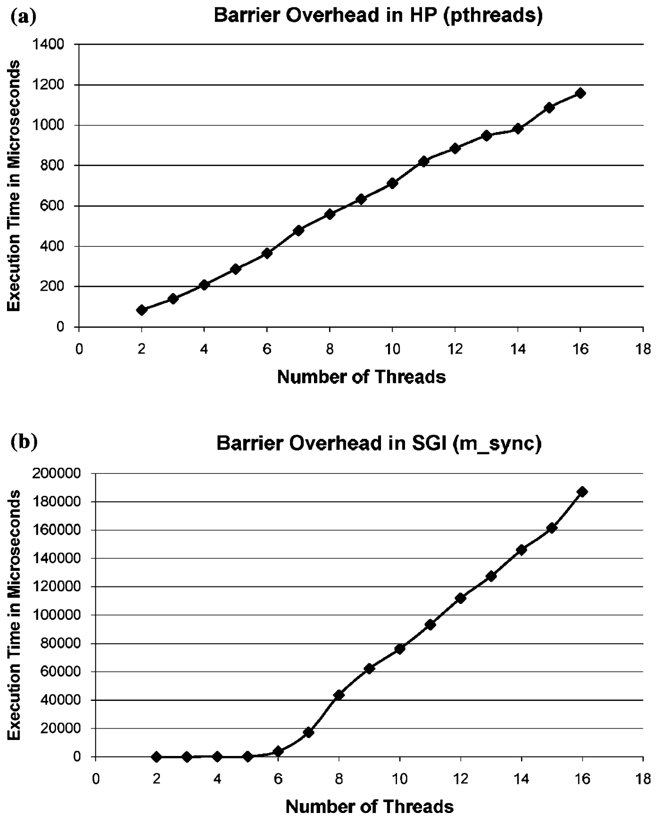
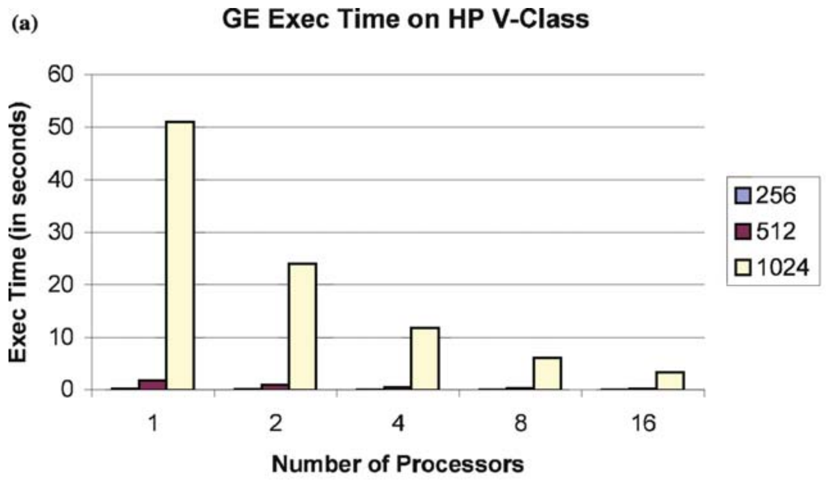


Fig. 17. Multiprocessor barrier performance (note scale). (a) HP V-class (b) SGI Origin 2000.

5.1. Overview of the Applications and Results

The GE application implements Gaussian Elimination with partial pivoting on a system of linear equations stored in a two-dimensional shared array. The computation on the rows is distributed across the processors in a round-robin fashion in order to maintain a good load balance as elimination proceeds. The access pattern of this application is single-write multiple-reader with an algorithm that is triangular in nature. The MM application performs the multiplication of two shared matrices. The result matrix is also shared. The result matrix is distributed into blocks depending on the number of processors available. The access pattern is mainly read-shared. The TC application uses iterations of MM to compute the transitive closure of a matrix. The current limitation of this

(a)



(b)

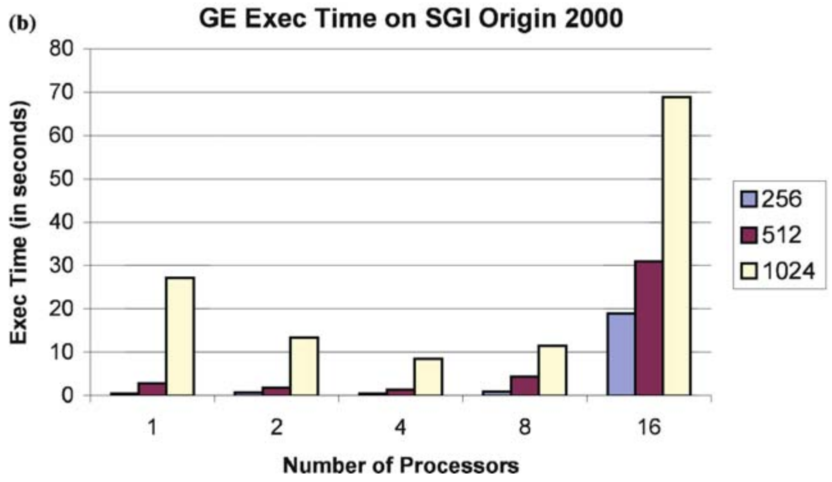


Fig. 18. GE Execution time results. (a) HP V-class (b) SGI Origin 2000.

algorithm is that it requires a square number of processors for parallel execution. The FFT application implements the Cooley–Tukey 1D FFT algorithm based on the several ($\log N$) stages of butterfly computation. The FWA application implements an all-pairs shortest path algorithm to compute the shortest path between two nodes in a graph.

The speedup and execution time results measured by running these applications under different data sizes and applications sizes are shown

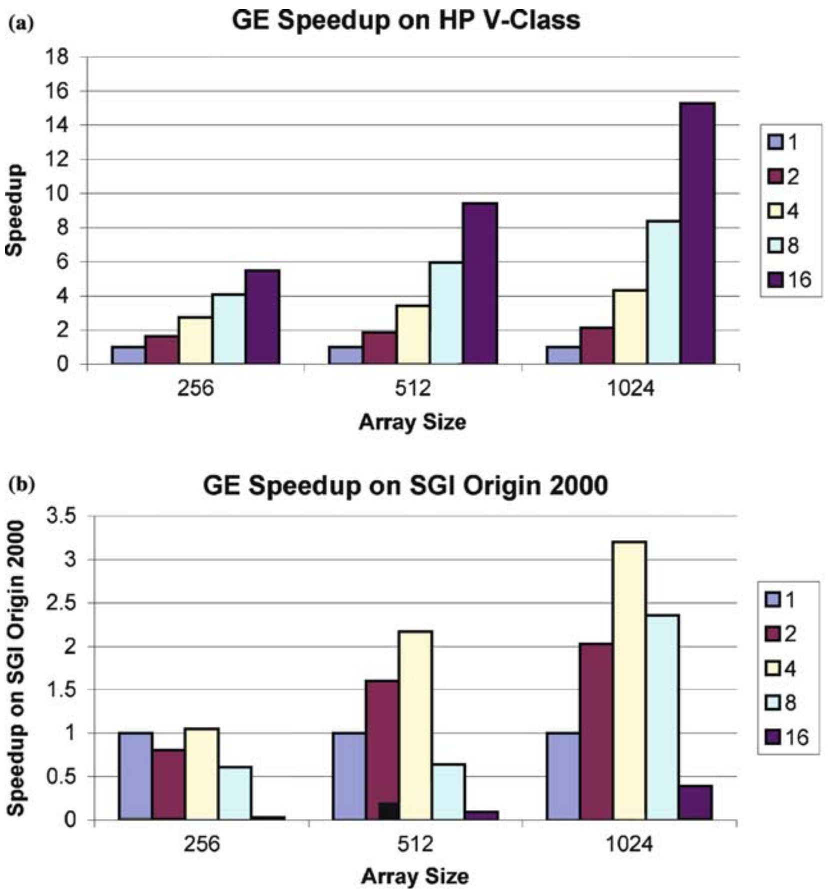
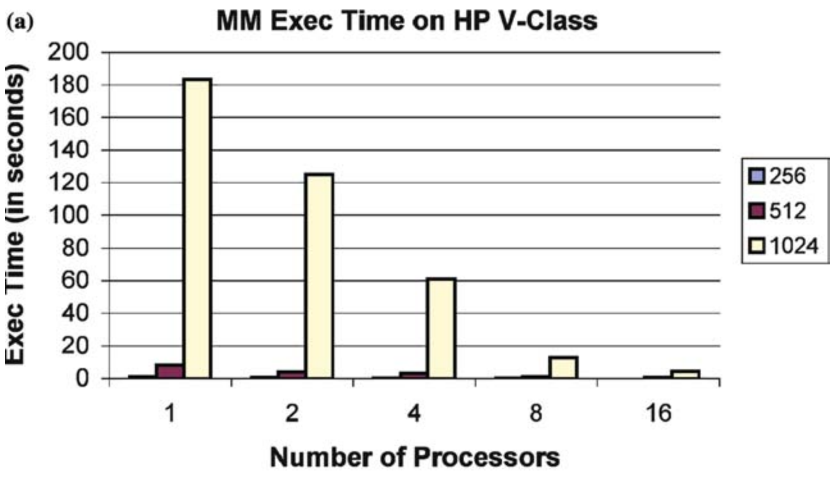


Fig. 19. GE Speedup results. (a) HP V-class (b) SGI Origin 2000.

in Figs. 18–25, 29 and 30. In general, we notice that as more processors are employed, the HP provides better speedup ratios than the SGI. The speedup is better on the HP because the communication to computation ratio on the SGI is much lower than on the HP. However, in most cases, the uniprocessor execution time on the SGI is much lower than the HP. This effect can be attributed to larger caches and more spatial locality within a larger L2 cache line on the SGI as compared to the HP. Below, we present a detailed analysis of performance using two of the five scientific applications, FFT and FWA.



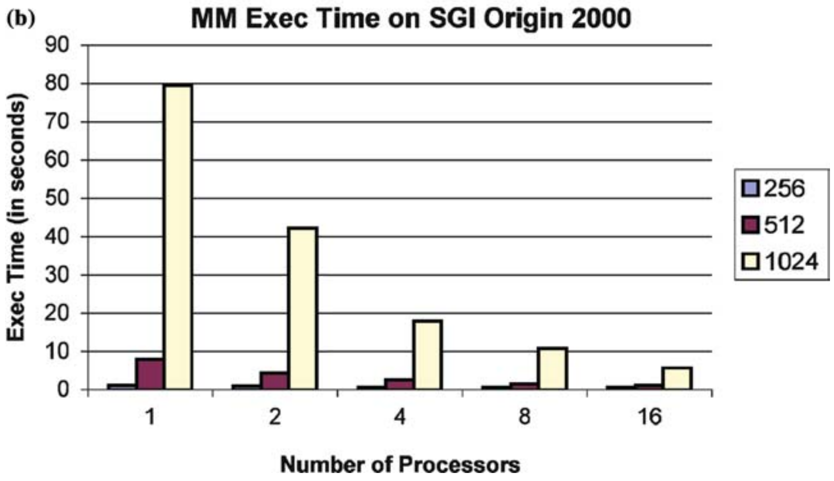
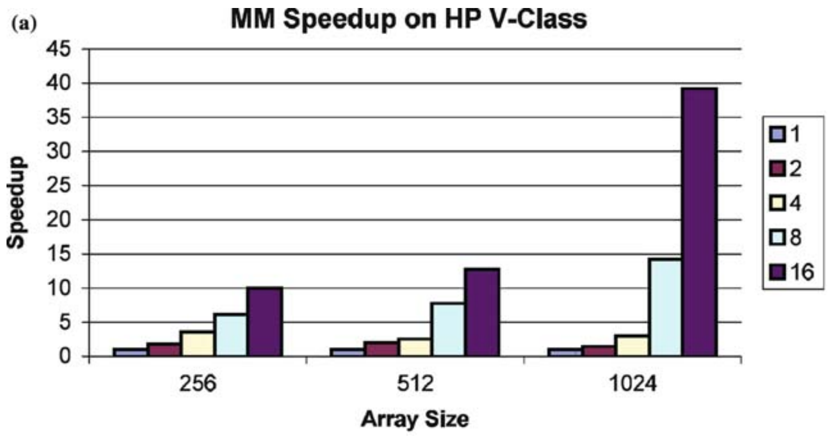


Fig. 20. MM Execution time results. (a) HP V-class (b) SGI Origin 2000.

5.2. Case Study of the FFT Application

The FFT application implements the Cooley–Tukey 1D FFT algorithm based on the several (log$N$) 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



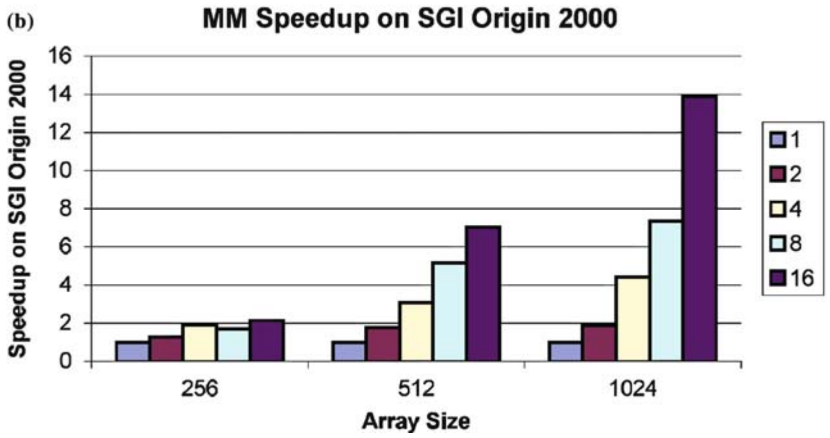


Fig. 21. MM Speedup results. (a) HP V-class (b) SGI Origin 2000.

exchange data in pairs. In each stage, one processor reads the data computed in the previous stage by a different processor.

Figures 24 and 25 present the execution time and observed speedup, respectively, of the FFT application under varied input data sizes and varied number of processors on the HP and the SGI. The application size is varied from 512 K points to 8 M points, with 32 bytes of data maintained per point. Hence, the total memory requirements varied from 16 MB (512 K $*$ 32 bytes) to 256 MB (8 M $*$ 32 bytes). The number of processors is varied from 1 to 16. The y-axis shows the obtained speedup and execution time. The results on the HP show good speedup improvements

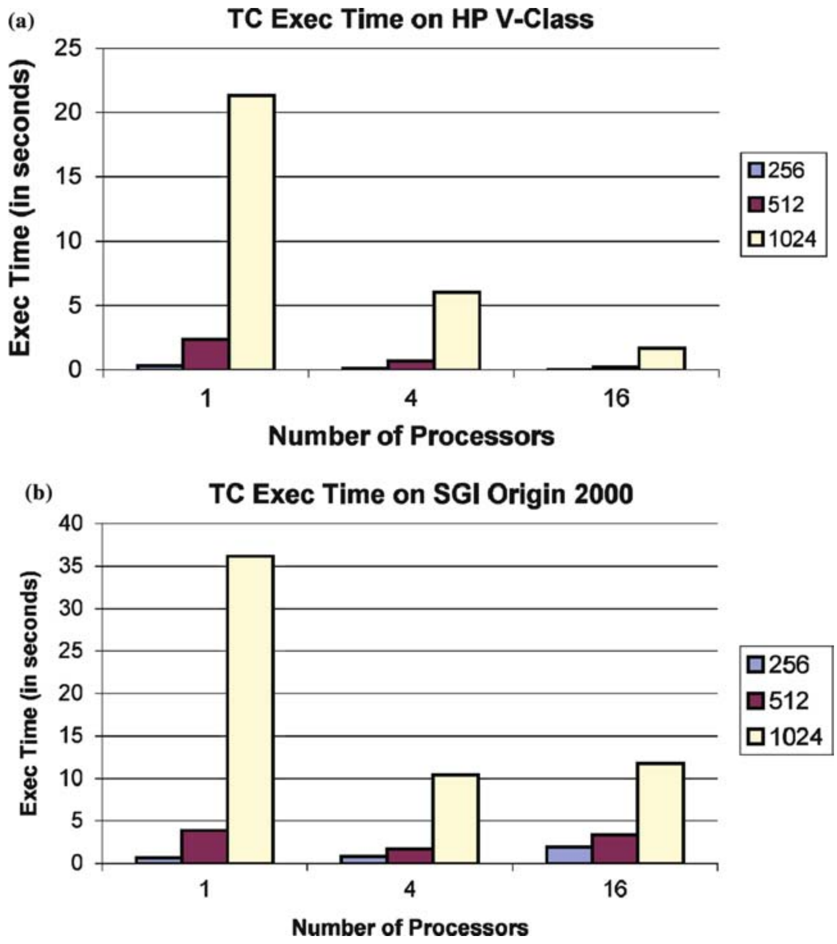
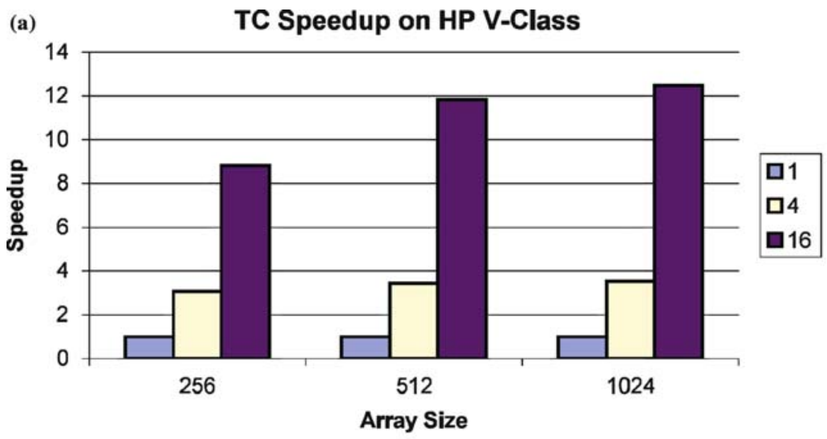


Fig. 22. TC Execution time results. (a) HP V-class (b) SGI Origin 2000.

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 is relatively low. We observe that the HP speedup obtained does not depend on the size of data input. However, when we measured the speedup obtained from the SGI, 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 MB. This suggests that communication to computation ratio

(a)



(b)

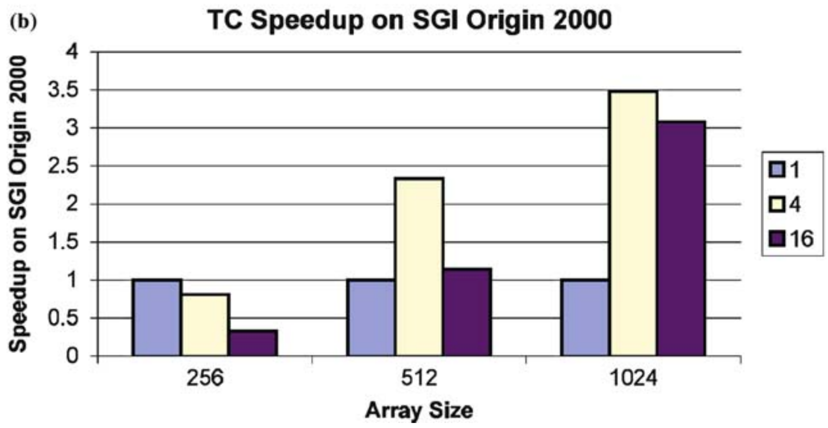
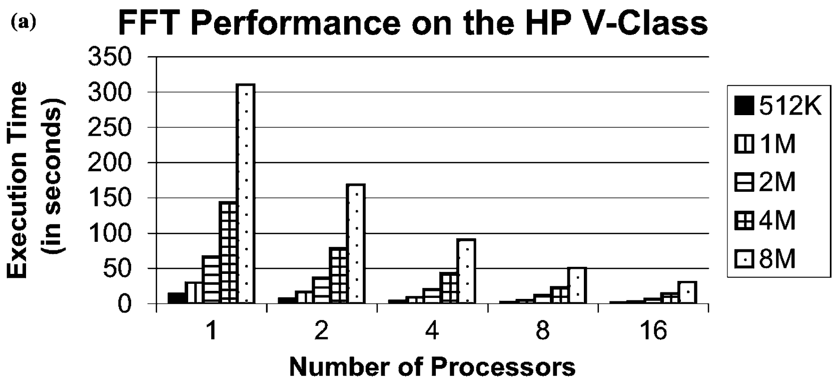


Fig. 23. TC Speedup results. (a) HP V-class (b) SGI Origin 2000.

is much higher than that for the HP. Finally, for 512 K and 1 M 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 Fig. 24. While the SGI had poor speedup ratios, it should be noted that the raw execution time is lower than on the HP. The execution time for 2 M 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. As we have seen in Section 3, the SGI load/store latencies were much

(a)


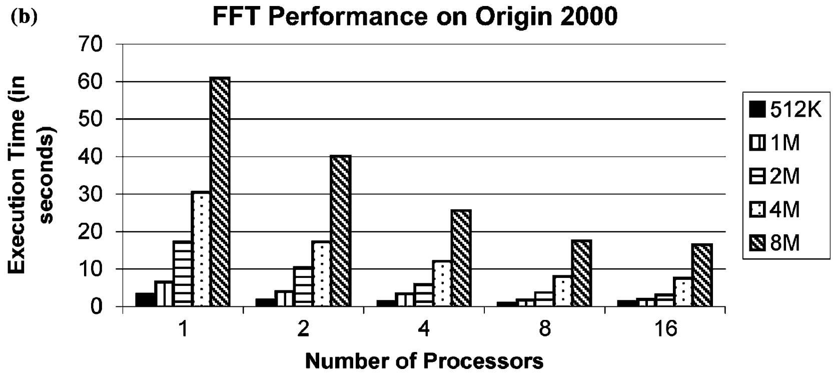
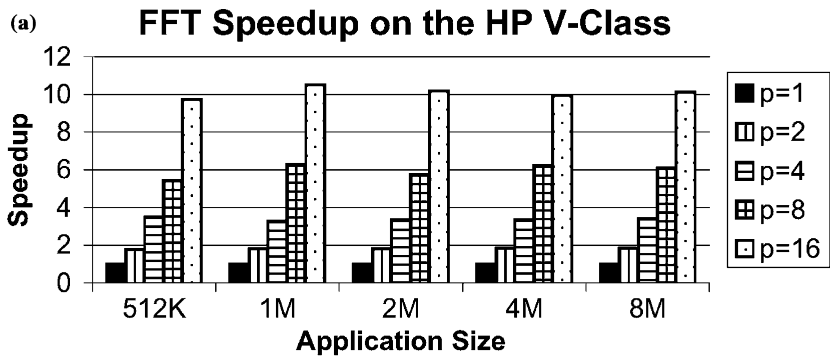
(b)


Fig. 24. FFT Execution time results. (a) HP V-class (b) SGI Origin 2000.

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 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 rarely be used during FFT execution.

In order to get a better understanding of what is going on, we used hardware counters on the respective systems to collect information about events that might be contributing to run times. On the HP, we used HP's CXperf,(6) which allows us to collect the hardware counters in a single shot. On the SGI, we used SGI's *perfex*(14) command to capture the application-wide counts for each of the events available. In order to get the most accurate counts on the SGI, we ran three iterations for each of the 16 pairs of counters(15) available instead of multiplexing all counters on a

(a)


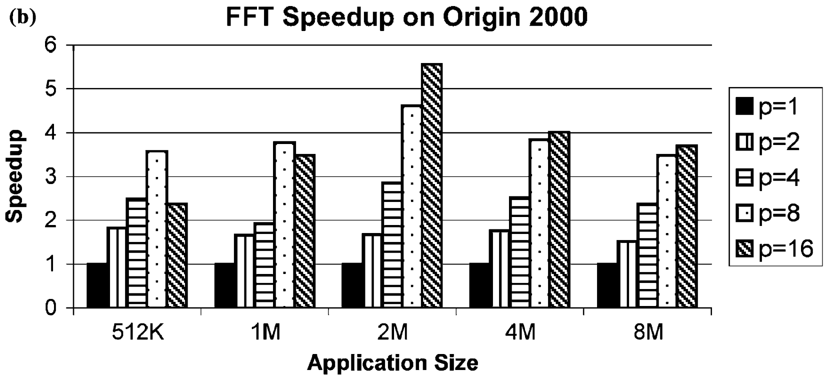
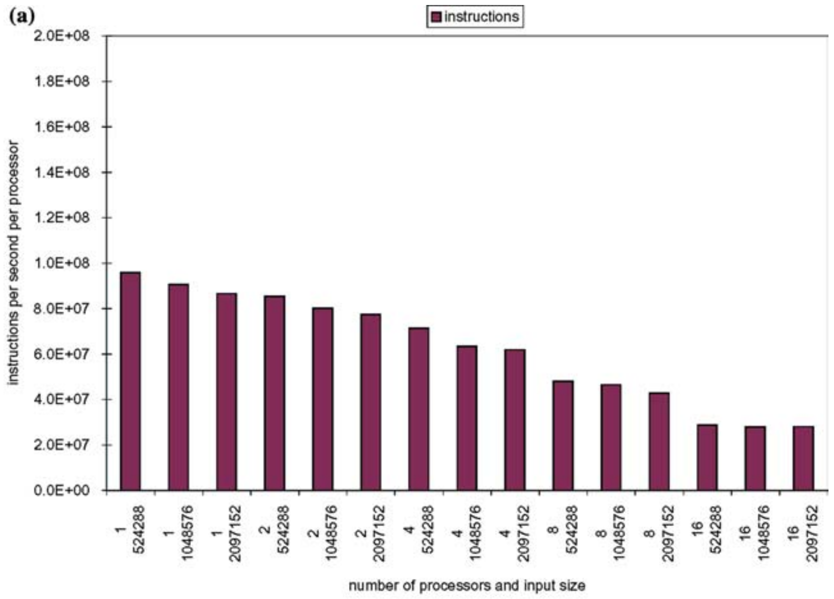
(b)


Fig. 25. FFT Speedup results. (a) HP V-class (b) SGI Origin 2000.

single run. Larger data sizes were excluded in our counter data due to the computational cost of this approach. Note that there is no one-to-one correspondence between counters on both architectures (e.g. the HP only has a single level of cache), but the results are revealing nonetheless.

The first note of interest is the instruction counts (average per processor per second) in Fig. 26. Given the above rates of instruction processing, it is clear why the HP took significantly longer than the SGI despite only a 20% difference in clock speed. Although it isn't apparent from this graph, the HP actually executed more than twice as many instructions than the SGI. Comparing *pthreads* and *m_fork* based implementations on the SGI indicates no substantial difference in instruction counts. In this case, the higher instruction count on the HP is indicative of the intrusiveness of HP's hardware counters—they require object code modification. For the applications studied, the intrumented object code is more than triple the size of the non-instrumented code on the HP.

(a)



(b)

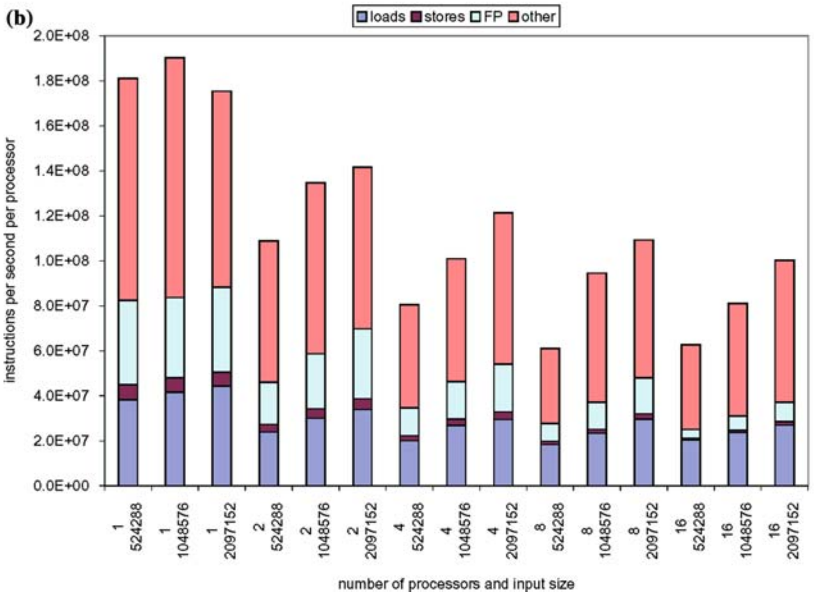


Fig. 26. FFT-Instructions retired graduated per second per processor. (a) HP V-class (b) SGI Origin 2000.

For the SGI graph, since there are counters for loads and stores issued, we've broken down the graph into loads, stores, floating point and other. It is apparent that loads are a significant percentage of the instruction count. On both systems, the cost of interprocessor communication is apparent—as more CPUs are utilized, the effectiveness of each CPU is diminished. Also note that due to this communication, both systems fall well short of executing one instruction per cycle. On the HP, at least one out of every two cycles is wasted. We will see in Section 5.3 that this isn't so with FWA where there is less interprocessor communication.

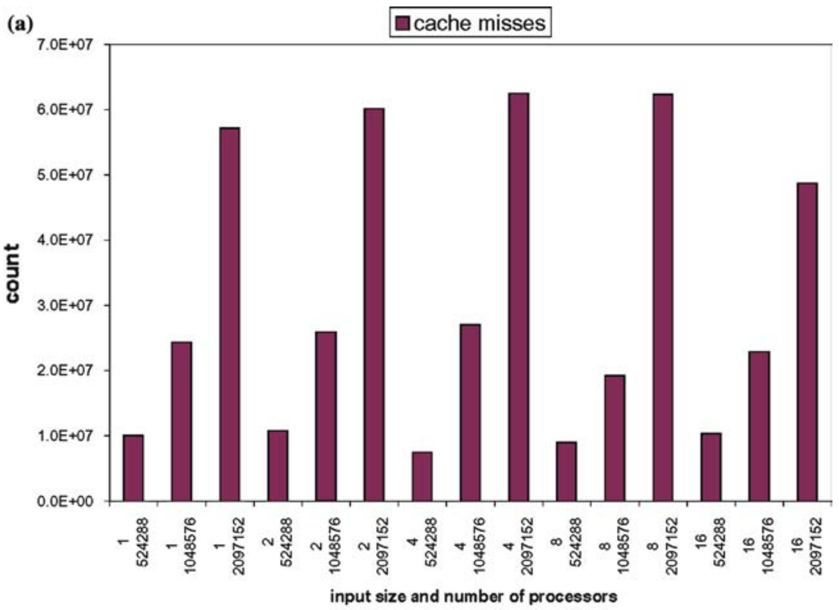
The interaction of the memory hierarchy while maintaining cache coherency is a useful indicator of performance since it gives an idea of how often the CPUs were waiting idle for data. Figure 27 shows the cache misses experienced on the HP and SGI. For the SGI, we've combined the data for L1 and L2 misses since that data was possible, so the total cache misses represented there are a bit misleading since sometimes an L1 miss will also trigger an L2 miss (i.e., there is some duplication represented in the graph). Also recall from Section 3.2 that we found a cache miss takes about 500 ns on the HP while an L2 miss on the SGI on takes about 350 ns to service.

For FFT, the system-wide TLB misses shown in Fig. 28 seem to be consistent among the experiments. Although the HP suffers fewer misses for the smallest data set, as the data sets grow, the HP and SGI TLB misses seem to equalize (despite the differing instruction counts and run times). More observations on the TLB behavior of the two systems will be made in Section 5.3 on FWA.

Although there are a considerable number of other counters available on the MIPS R10000, we haven't included them here due to a lack of complementary counters on the PA-8200. Given their usefulness(30) and potential ability to help predict performance,(3) this is unfortunate. Fortunately, HP's move towards Intels IA-64 adn AMD's opteron should help alleviate this shortcoming in the future.

5.3. Case Study of the FWA Application

The FWA implements an all-pairs shortest path algorithm to compute the shortest path between all pairs of vertices in 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. In our case, each vertex had a direct path (weighted edge) to approximately 70% of the other vertices. Within each iteration, every processor reads a single row of an adjacency matrix and updates a distance matrix to keep track of the distance



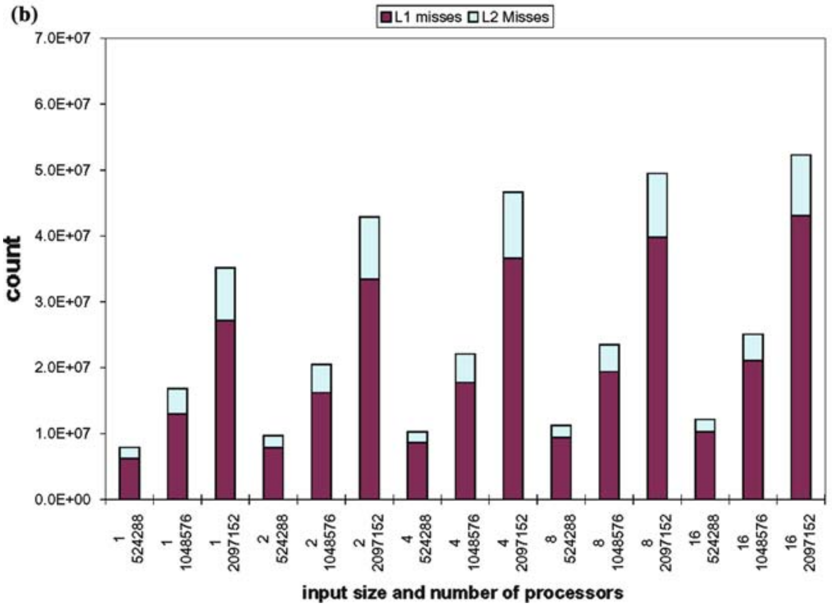
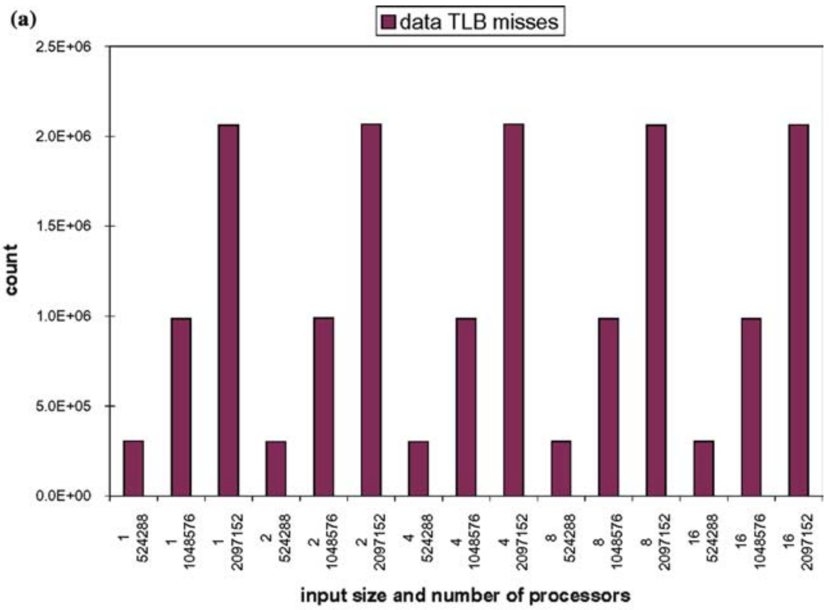


Fig. 27. FFT-Data cache misses. (a) HP V-class (b) SGI Origin 2000.



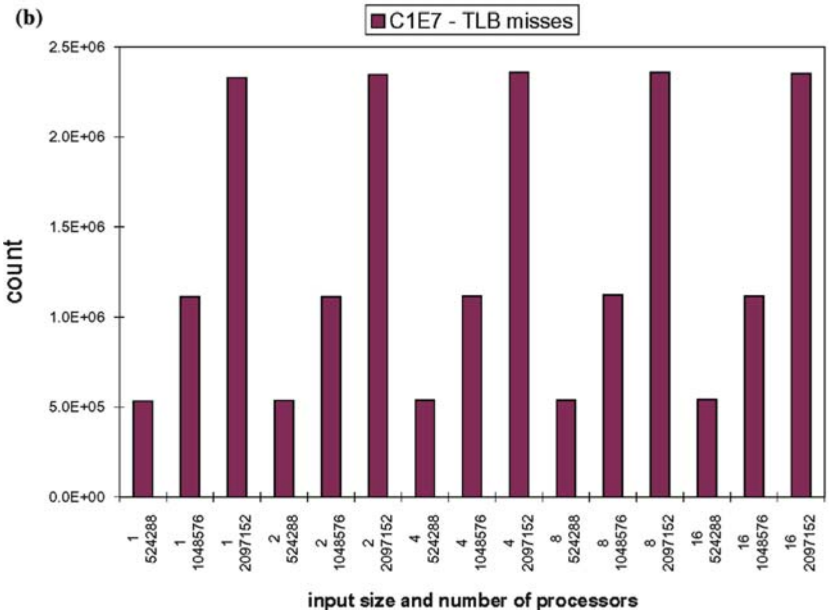


Fig. 28. FFT—TLB misses. (a) HP V-class (b) SGI Origin 2000.

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.

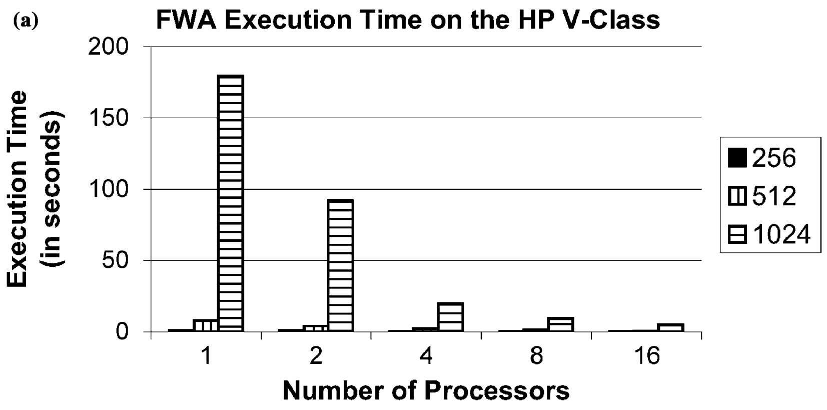
Figures 29 and 30 present the execution time and observed speedup, respectively, of the FWA application for various input data sizes and using a varied number of processors on the HP and the SGI, respectively. The size of the application is varied on the x-axis from 256 vertices to 1024 vertices. The storage requirements for the FWA data structures are 768 KB, 3 MB, and 12 MB for the predecessor (4 byte *ints*) and distance (8 byte *doubles*) matrices (size $N \times N$) for N equal to 256, 512, and 1024, respectively. The number of processors (shown in the legend) are varied from 1 to 16. The y-axis shows the execution time and the obtained speedup, respectively.

Like the FFT application results, the uniprocessor execution time is much higher on the HP than on the SGI Origin. However, as more processors are used, the HP execution time improves superlinearly beyond the SGI execution time. For example, with 1024 vertices and 16 processors, the execution time of the HP is approximately 13% lower than the SGI Origin execution time. On the SGI, 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 SGI tends to be much higher than the HP. This reduces the effect of all uniprocessor access pattern benefits on the SGI. From this result, we may infer that the HP offers better performance than the Origin 2000 for applications with high read sharing degrees.

From Fig. 30(a), we find that the speedup gained on the HP multiprocessor ranges from roughly nine (for 256 vertices) to approximately 35 (for 1024 vertices) using 16 processors. One of the main causes for the superlinear speedup on the HP is the larger data size. When the data fits into the cache (768 KB for example), the uniprocessor performance is good due to a higher cache hit ratio. On the other hand, with a data size of 12 MB, the working set overcomes the cache size, causing cache and TLB 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 are lower than on the HP, similar to the FFT execution. For a 256-vertex graph, four processors seem to be the optimal number, with performance deteriorating beyond this threshold. On the other hand, with 1024 vertices, the speedup increases at a steady pace.

As Fig. 31 verifies, there are significant cache effects in play here—the HP incurs fewer cache misses than the SGI, despite the smaller

(a)



(b)

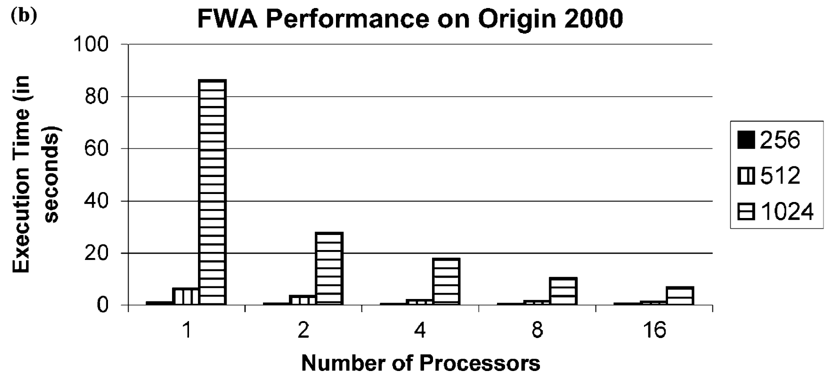
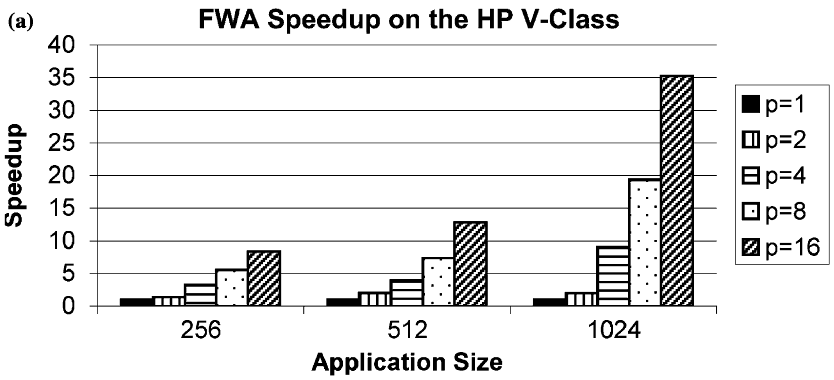


Fig. 29. FWA Execution time results. (a) HP V-class (b) SGI Origin 2000.

cache and smaller cache line (the larger numbers at the low-end will be partly explained by the TLB discussion below). In this particular case, the single-level cache *appears* to be doing better than a smaller first level cache coupled with a larger second level cache. Also note that while L1 misses on the SGI increase for FFT as we spread the problem across more processors, FWA misses L1 cache a consistent number of times regardless of how many processors are utilized. In the case of 1024 vertices on four processors for the HP, realize that although each processor will process 2 MB of the distance matrix, it may not necessarily make updates to the 1 MB predecessor matrix which help explains how most of the problem stays in the 2 MB cache.

Figure 32 shows the instructions performed per second by each processor in the two systems. Here, unlike FFT, we see signs of instruction-



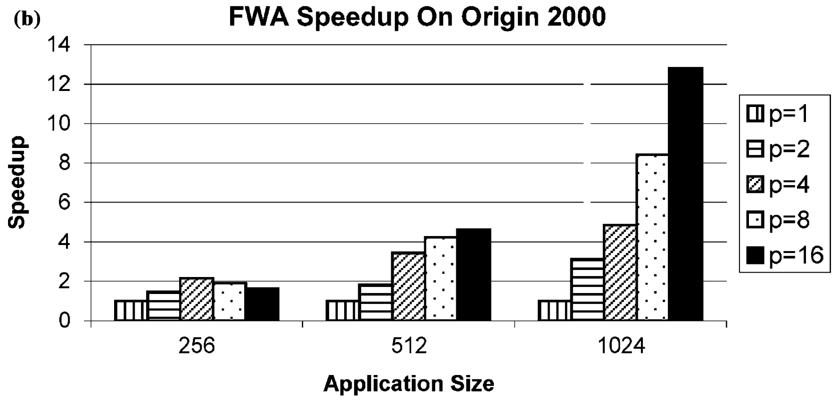


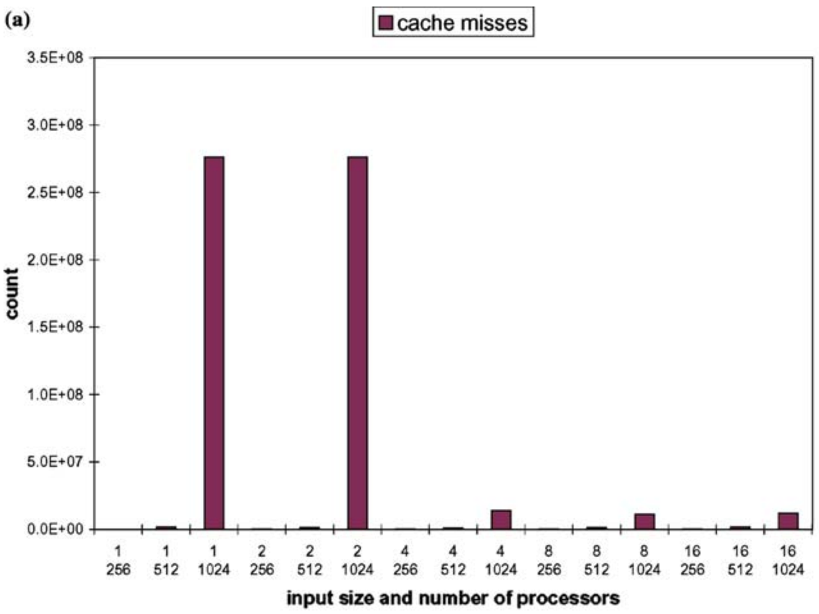
Fig. 30. FWA speedup results. (a) HP V-class (b) SGI Origin 2000.

level parallelism since in some cases the instruction rate exceeds the clock frequency. Although not shown here, it is interesting to note that unlike FFT, the instruction count between the two systems is very similar. In fact, on the high-end the HP accomplishes the same amount of work as the SGI by processing significantly fewer instructions which explains why the HP has better performance on the larger data sets.

Finally, we look at TLB misses on the two architectures in Fig. 33. Here, as opposed to the FFT, there are some some significant activities indicated as the problem is spread out among more processors. These are also reflected in the previous cache miss graphs.

Under HP-UX 11.0, physical to virtual memory page mappings can be made using differing page sizes—i.e., "By default, the system heuristically determines whether Large Pages are suitable for a given memory object and sets the page size hint accordingly." "So by default, most

(a)



(b)

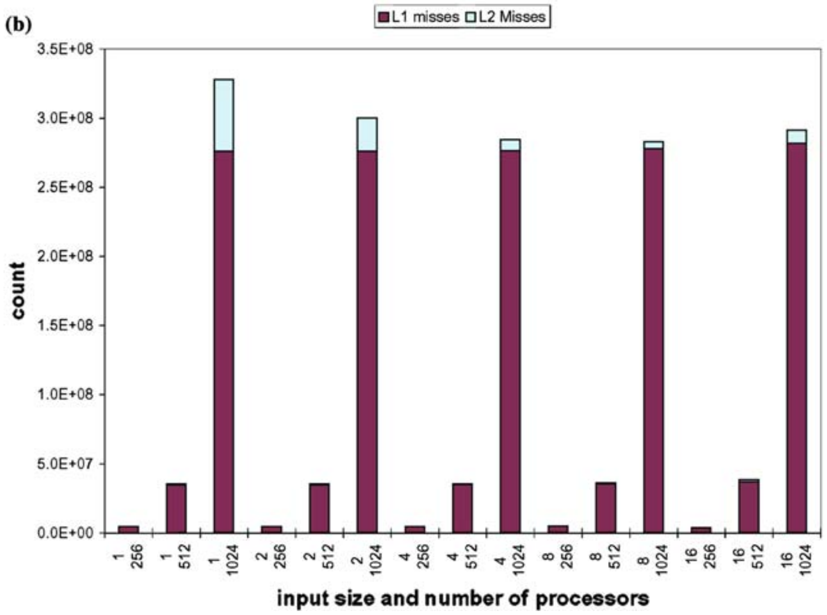
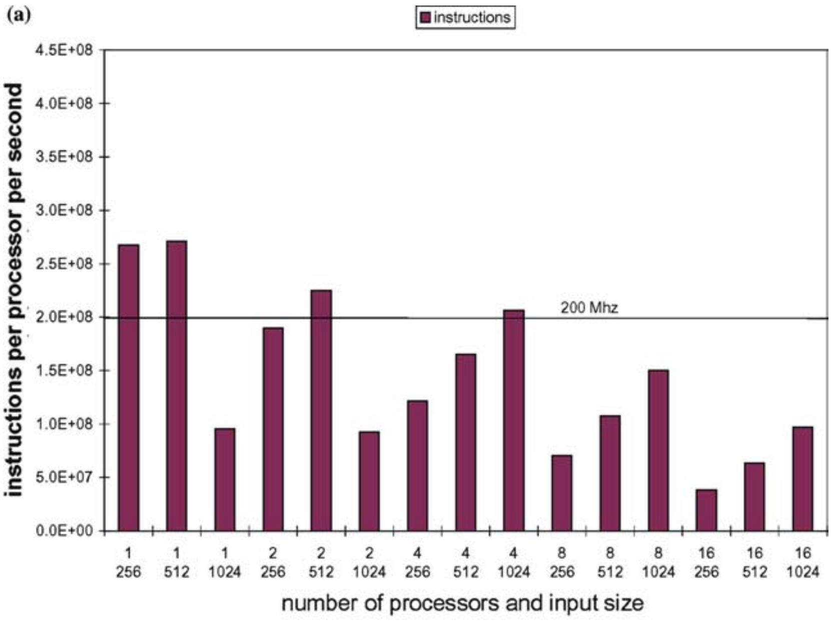


Fig. 31. FWA—Data cache misses. (a) HP V-class (b) SGI Origin 2000.

(a)



(b)

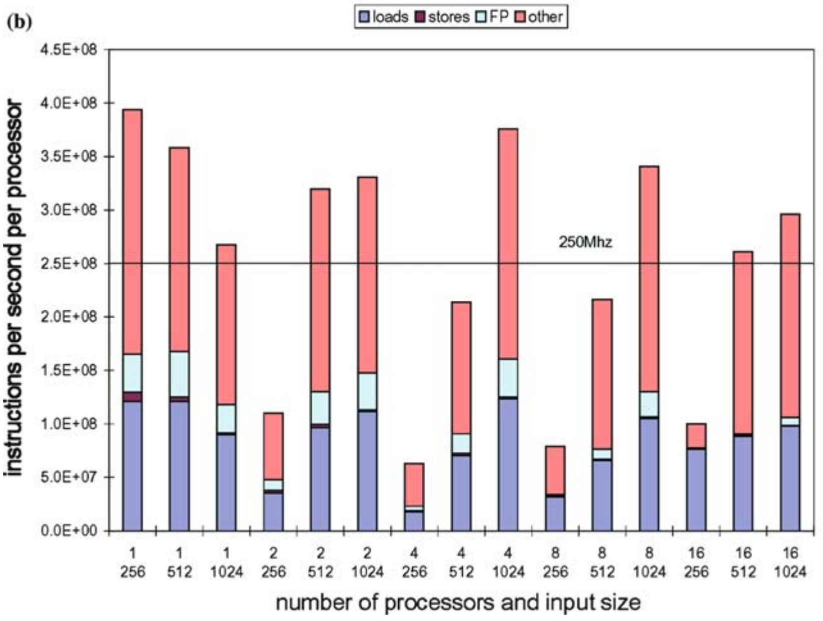
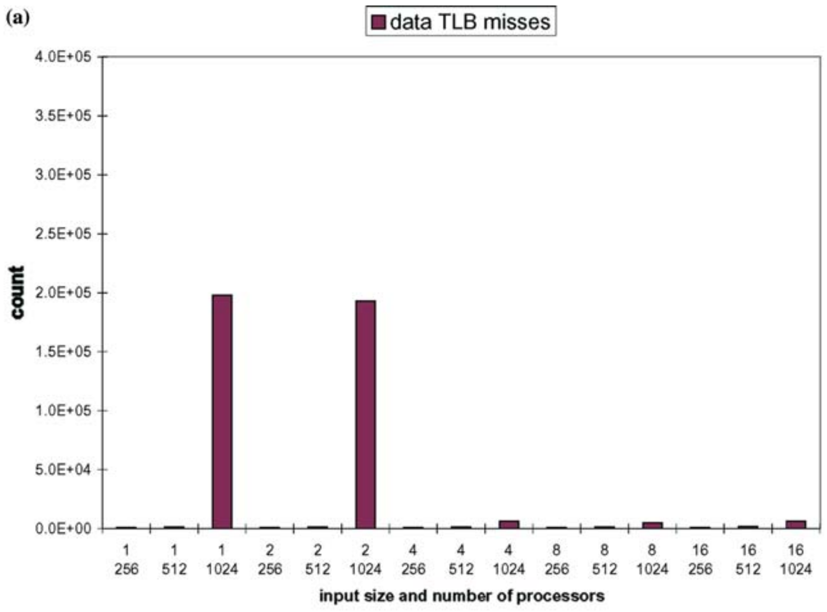


Fig. 32. FWA—Instructions retired/graduated per second per processor. (a) HP V-class (b) SGI Origin 2000.



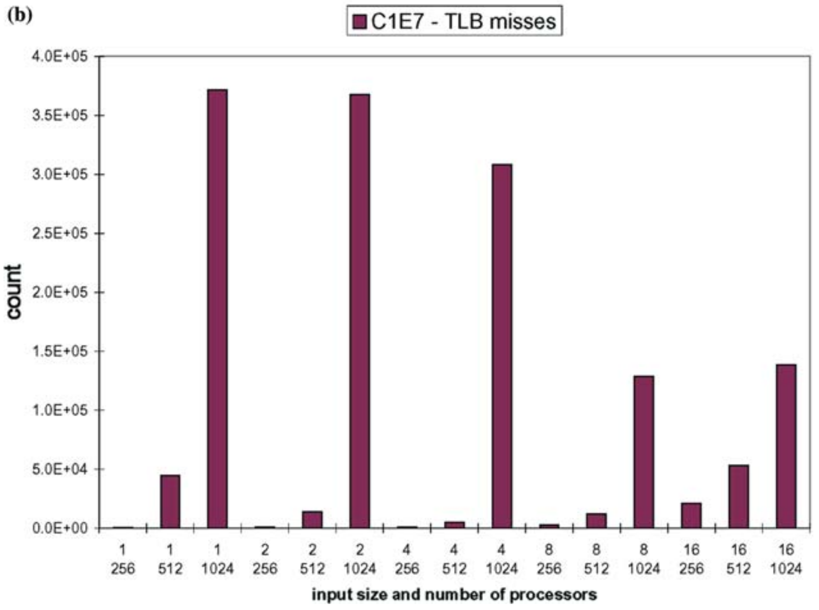


Fig. 33. FWA—TLB misses. (a) HP V-class (b) SGI Origin 2000.

memory objects will use 4 K pages as in previous releases of HP-UX, but the system may flag an object for 16 K pages if conditions warrant."(21) Using the HP *chatr*(7) command on the executable indicates that no page size was forced upon the program and so the OS was left to decide what size pages to use. Further investigation using HP-UX's *pstat_getprocvm*(8) function revealed that regardless of the data size, FWA allocate two pages of 4 KB and three pages of size 16 KB. Depending upon the size of the data set, it allocated an additional 13, 49 and 193 size 64 KB pages for the matrices for N equal to 256, 512 and 1024, respectively.

So, for 1024 vertices on a single processor, we've exceeded the 112 entries available in the TLB (actually 120 but shared between instruction and data) and will have to spend time on virtual to physical address translations. Also, recall the cache miss graph which complements Fig. 33.

The SGI also offers variable page sizes, but lacking programmer/user participation will use the default page size (in this case, 16 KB on Texas A&M University's SGI Origin 2000) determined by the system operator. Since the TLB only has 64 entries (as opposed to the HP's 120), then the total amount of memory that is accessible without incurring a TLB miss is 1 MB.

These differences in approaches to the default handling of page sizes makes for vastly different performance when doing virtual to physical address translations on the two systems, as Fig. 33 indicates.

6. 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 size 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 25–50% lower than the HP V-Class. However, when multiple requests are active in the system, the HP V-Class's performance surpasses the SGI Origin performance with average latencies about 55–60% lower than the SGI Origin 2000. We studied the impact of coherence protocol optimizations on access latencies of dominant multiprocessor access patterns like RAW and WAR. As a result, we learned that the HP V-Class's latencies remain relatively constant as more processors become active, whereas the SGI Origin 2000's latencies grow in a somewhat linear fashion.

Five different scientific applications 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. Although the execution times of the SGI Origin 2000 were superior in most cases, we did find that the HP V-Class obtains better speedups than the SGI Origin 2000. The SGI Origin's poor speedups can be attributed to better optimized uniprocessor execution. A detailed study of two of the five applications, each with a different dominant multiprocessor sharing pattern, was also presented to investigate several of these observations and analyze system performance.

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*, **9**(2)(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 (1998).
3. N. M. Amato, J. Perdue, A. Pietracaprina, G. Pucci, and M. Mathis, Predicting Performance on SMPs. A Case Study: The SGI Power Challenge, *International Parallel and Distributed Processing Symposium* (IPDPS), Cancun, Mexico, (2000).
4. L. M. Censier and P. Feautrier, A New Solution to Coherence Problems in Multicache Systems, *IEEE Transactions on Computers*, C-**27**(12):1112–1118(1978).
5. D. Culler, R. Karp, D. Patterson, A. Sahay, and E. Santos et al, Log P: a Practical Model for Computation, *Communications of the ACM*, **39**(11):78–85, (1996).
6. CXperf User's Guide. Hewlett-Packard Corp. http://docs.hp.com/hpux/onlinedocs/B6323-96001/B6323-96001.html
7. HP-UX man page for *chatr*. HP.
8. HP-UX man page for *pstat_getprocvm* (). HP.
9. HP RISC Precision Architecture 2.0 (PA-RISC 2.0) Document, Hewlett-Packard Corporation, http://wwwhp.com/ahp/framed/technology/micropro/
10. HP 9000 V-Class Server Architecture Document, 2nd Edition, Hewlett-Packard Corporation, http://docs.hp.com:80/hpux/systems/#vclass.
11. 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).
12. R. Iyer, G. Janakiraman, R. Kumar, and L. Bhuyan, A Trace-Driven Analysis of Sharing Behavior in TPC-C, *2nd workshop on Computer Architecture Evaluation using Commercial Workloads* (1999).
13. R. Iyer, N. M. Amato, L. Rauchwerger, and L. Bhuyan, Comparing the Memory System Performance of the HP V-Class and SGI Origin 2000 Multiprocessors using Microbenchmarks and Scientific Applications, *Proceedings of the 13th ACM International Conference on Supercomputing (ICS' 99)*, pp. 339–347 (June, 1999).
14. IRIX man page for *perfex*. SGI.
15. IRIX man page for *r10k_counters*. SGI.

16. D. Jiang and J.P. Singh, Scaling Application Performance on a Cache-coherent Multiprocessors, *Proceedings of the 26th International Symposium on Computer Architecture (ISCA)*, Atlanta, (May, 1999).
17. D. Jiang, and J. P. Singh, A Scaling Study of the SGI Origin2000: A Hardware Cachecoherent Multiprocessor, *9th SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, (1999).
18. J. Laudon and D. Lenoski, The SGI Origin: A ccNUMA Highly Scalable Server, *Proceedings of the 24th International Symposium on Computer Architecture*, pp. 241–251, (May, 1996).
19. D. Lenoski et al., The Stanford DASH Multiprocessor, *IEEE Computer*, **25**(3): 63–79 (1992).
20. T. Lovett and R. Clapp, STiNG: A CC-NUMA Computer System for the Commercial Marketplace, *23rd Annual International Symposium on Computer Architecture*, pp. 308–317 (1996).
21. C. Mather, K. Peterson, B. Raghunath, J. Reddy, I. Subramanian, B. Taylor, and E. Wong (all from Hewlett Packard), Performance Optimized Page Sizing in HP-UX 11.0, IWorks 1998 Presentation.
22. J. D. McCalpin, Memory Bandwidth and Machine Balance in Current High Performance Computers, *IEEE Technical Committee on Computer Architecture newsletter* (1995).
23. L. McVoy and C. Staelin, lmbench: Portable Tools for Performance Analysis, *Proceedings of USENIX*, San Diego (1996).
24. Parasol - HP V-Class Multiprocessor, Parasol Lab, Department of Computer Science, Texas A&M University, http://www.cs.tamu.edu/research/parasol.
25. 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 (1993).
26. K. Shaw and G. Astfalk, Four State Cache Coherence Protocol in the Convex Exemplar System, http://www.hp.com/wsg/tech/technical.html.
27. Titan - SGI Origin 2000, Supercomputing Center, Texas A&M University, http://anakin.tamu.edu/titan/
28. L. Valiant, A Bridging Model for Parallel Computation, *Communications of the ACM*, **33**(8):103–111(1990).
29. K. Yeager, The MIPS R10000 Superscalar Microprocessor, IEEE Micro, **16**(2):28–40 (1996).
30. M. Zagha, B. Larson, S. Turner, and M. Itzkowits, Performance Analysis Using MIPS R10000 Performance Counters, *Proceedings of Supercomputing'96*, November 17–22, 1996 Pittsburgh, PA, ACM Press and IEEE Computer Society Press, 1996.