in *Proceedings of IEEE International Conference on Computer Design (ICCD'97)*

A Comparative Evaluation of Hierarchical Network Architecture of the HP-Convex Exemplar *

Robert Castañeda
Division of Computer Science
The University of Texas at San Antonio
San Antonio, Texas 78249

Xiaodong Zhang
Department of Computer Science
College of William and Mary
Williamsburg, Virginia 23187

James M. Hoover Jr.
Tech NOW Inc.
San Antonio, Texas 78228

Abstract

The Convex Exemplar (SPP1000 and SPP2000 series) is a new commercial distributed shared-memory architecture. Using a set of system kernels and two application programs, we examine performance effects on network latency, hot spot contention, cache coherence and overall scaling capability, which result both from the choice of the network structure as well as from its CC-NUMA memory system feature. Since the KSR-1 was also targeted at scalable cache coherent shared-memory systems by using hierarchical interconnection networks, we compared the architecture and performance results with the KSR-1. Our experiments indicate that the memory access latency of the Exemplar is comparatively low due to its fast processor node and the unique network/system structure. In addition, the Coherent Torodial Interconnect (CTI) rings are efficient in handling cache coherence activities on the Exemplar. However, the Exemplar synchronization primitives need further exploit its hierarchical architecture in a high

*This work is supported in part by the National Science Foundation under research grants CCR-9102854 and CCR-9400719, by the U.S. Air Force Office of Scientific Research under grant AFOSR-95-1-0215, and by a Fellowship from the Southwestern Bell Foundation. Part of the experiments were conducted on the KSR-1 machines at Cornell University and at the University of Washington, and on the Exemplar at HP-Convex Technology Center in Richardson, Texas.

contention environment. As we expect with extremely fast processors in a hierarchical memory structure, the high-speed cluster-based Exemplar system is more sensitive to data locality.

1 Introduction

Large scale distributed shared-memory architectures provide a shared address space supported by physically distributed memory. The strength of such systems comes from combining the scalability of network-based architectures with the convenience of the shared-memory programming model. Computing performance on such systems is affected by two important architecture and system design factors: the interconnection network and the memory system structure. The choice of interconnection networks to link processor nodes to cache/memory modules can make non-uniform memory access (NUMA) times vary drastically, depending upon the particular access patterns involved. With respect to the kinds of memory organizations utilized, cache coherence shared-memory systems can be classified into CC-NUMA [1], [6], [11] and COMA [3], [5].

The KSR-1 has been available since late 1992 for general purpose parallel computing. It is the only commercial COMA machine. Although the KSR machine has been discontinued, its COMA system and hierarchical ring network are still interesting. Now, there is a commercial CC-NUMA machine: the CONVEX Exemplar (SPP1000 and SPP2000 series). In [9] and [10], execution performance results of scientific application programs on a single and two hypernode Exemplar (8-16 processors) are reported. Our performance results on multiple hypernodes of this machine and the architectural implications should provide additional useful references for developers in both architecture and application fields. Since both the KSR-1 and the Exemplar architecture are targeted at scalable cache coherent shared-memory systems by using hierarchical interconnection networks, an empirical comparison between the two specific system implementations and a study of their merits/limits of each machine are important for large-scale distributed shared-memory architecture design. Using a set of system kernels and two application programs, we examine performance differences and effects on network latency, hot spot contention, cache coherence and overall scaling capability, which result both from different choices of network structures as well as from the CC-NUMA/COMA feature of the two machines. Although the ring to processor speed ratio of the KSR-1 is much higher than the ratio of the Exemplar, the memory access latency of the Exemplar is significantly lower than the KSR-1 due to its fast processor module and the unique network/system structure. In addition, the CTI rings are efficient in handling cache coherence activities on the Exemplar. However, the KSR-1 synchronization primitives exploits the ring architecture well and performs more efficiently in a high contention environment than the Exemplar's. Our studies also indicate that the high-speed cluster-based Exemplar system is more sensitive to data locality. This is expected when one considers the extremely fast processors operating in a hierarchical memory environment.

The organization of the paper is as follows. The next section first overviews the Exemplar architecture, and compares the network/system with the KSR-1. It also qualitatively gives general performance expectations of the two machines. In Section 3, we discuss the experimental methodology for the evaluation. We also present the system kernels and the two application programs we used for measurements. Section 4 provides detailed architecture implications based on different types of experiments. Finally, we summarize the paper in section 5.

2 The Exemplar architecture

We first overview the system structure of the Exemplar, and then give comparisons with the KSR-1 in this section. The SPP2000 model is the newest model with faster processor nodes and larger hypernode. But only one hyernode of 16 processors is available for our experiments at the HP-Convex Technology Center. Since the performance of multiple hypernodes is important for us evaluate the impact of multi-level network choices and the scalability of the system, we choose the SPP1000 with 4 hypernodes of 32 processors for performance evaluation and comparisons. The processor nodes of the SPP1000 is HP PA7100 (100 MHz). In the SPP1000 series, there are other two variations: SPP1200 using HP PA7200 processor nodes (120 MHz), and SPP1600 using HP PA7200 processor nodes and 1 MByte instruction cache and 1 MByte data cache. Both SPP1000 and SPP1200 have 256 KByte instruction cache and 256 KByte data cache. The structure of the SPP2000 model is similar to the SPP1000 series, a CC-NUMA system. However, each hypernode has 16 processor nodes of HP PA8000 (200 MHz). A hypernode in SPP1000 series has 8 processor nodes.

2.1 CC-NUMA on crossbar multiple rings

The CONVEX Exemplar [1] is a CC-NUMA hierarchical architecture which is formed in three levels: functional units, clusters and the global shared-memory system.

Functional Unit. Each functional unit of the SPP1000 (see Figure 1) is comprised of two HP PA-RISC 7100 microprocessors for its computing power. Each processor uses a 6 stage pipeline for efficient execution of instructions. The processor speed is 100 MHz and can execute one integer instruction and one floating point operation in the same clock cycle. Besides internal instruction and data caches, each processor has an external data and instruction cache of 1 MByte in size. Communication between a processor and its own instruction/data caches takes approximately 2 clock cycles under no pipelining but could be as low as 1 clock cycle with pipelining. Processor cache lines are 32 bytes in size and a write-back policy is used to insure cache coherency. A functional unit can accommodate up to 512 MBytes of memory where a physical page is 4 KBytes in size. The memory can be further divided up to 3 different segments: *hypernode local memory* that is local to its hypernode, *subcomplex*[1] *global memory* that can be accessible by all other CPUs in a given subcomplex, and *interconnect cache* that is used to store copies of all data fetched from other hypernodes. Each functional unit also contains a CCMC (Convex Coherent Memory Controller) that provides an interface between the functional unit's memory and all other processors. It also takes care of cache coherency, as well as interfacing to the CTI for inter-hypernode communication. A CPU agent is used for processors to communicate with other processors as well as address mapping. The Convex Toroidal Interconnect (CTI), an SCI interface (based on the IEEE Standard

[1] A subcomplex is a virtual machine consisting of a specified number of processors on specified hypernodes

Cache Coherence Interface [4]), is used for inter-hypernode communications. The remaining sections of the paper will use the terms SCI and CTI interchangeably.

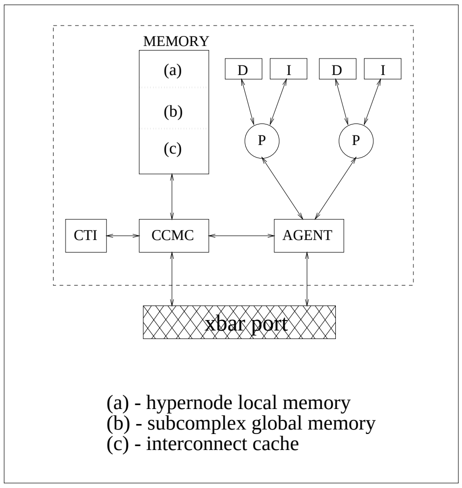


Figure 1: A functional unit on the Convex Exemplar SPP1000.

Hypernode. A hypernode is made up of four functional units grouped together. Functional units communicate with each other via a 5 port non-blocking crossbar (the fifth port is used to interface with I/O devices). The bandwidth of the crossbar is 1.25 GBytes per second. Memory access for a processor (even a processor within the same functional unit) must communicate through the crossbar. Access to any of the functional units from any processor takes the same time. Figure 2 shows a single hypernode with four functional units that can communicate with each other through a crossbar.

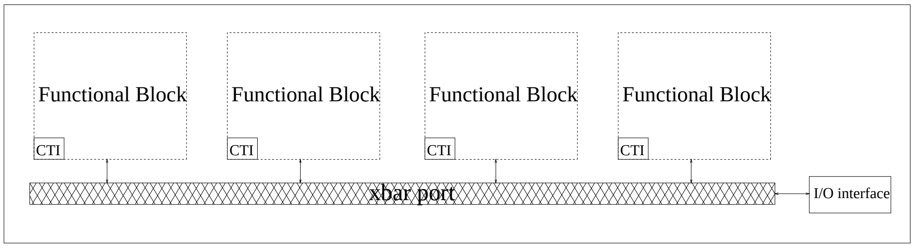


Figure 2: A single hypernode on the Convex Exemplar SPP1000.

The Global Shared Memory System. Functional units communicate across hypernodes via four CTI rings. A full configuration will enable 16 hypernodes to communicate through the four CTI rings (see Figure 3). Memory access through global memory is achieved by going through the local crossbar of the functional unit and then routed through the CTI interface of the functional unit that is associated with the memory location at the remote functional unit via the CTI ring. The request is then satisfied by returning along this same path back to the requesting processor and copied into the interconnect cache of the functional unit of which it resides. Packets of size 16

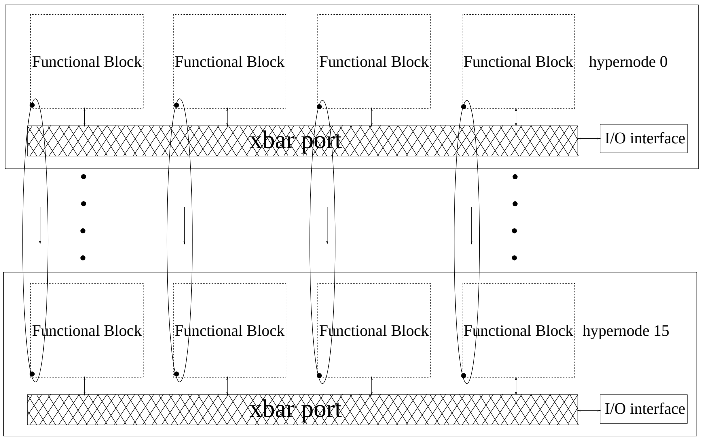


Figure 3: A system overview of the Convex Exemplar SPP1000.

bytes are used for a request, whereas 80 bytes are used for responses (of which 64 bytes are actual data) for communicating across these rings in a unidirectional path. The ring bandwidth is 660 MBytes per second per ring.

2.2 Discussions of major differences between the Exemplar and the KSR-1

Here we qualitatively and quantitatively discuss some important architecture and system implementation differences between the KSR-1 and the Exemplar.

2.2.1 A comparison of network latencies

Memory access latency is an important measure for a NUMA shared-memory system, which consists of network latency and memory latency. The network latency is a timing delay of a message in a network between its source and its destination. The memory latency is a timing delay for accessing a memory module. Here we use a simple model to compare the network latencies between the two machines. The major interconnection network of both machines is a ring. A unique feature of a memory access is that every remote request has to go around the entire ring before it returns to the requester. The other network factor that affects the performance is network bandwidth. In the KSR-1, the ring bandwidth is 1 GBytes/sec, about 50% higher than the individual ring bandwidth of the Exemplar (660 Mbytes/sec). Although the Exemplar has four rings, only one is used to satisfy a separate memory request. However, the number of nodes on a KSR-1 ring is 32 while the maximum number of nodes on an Exemplar ring is 16. A memory access in the Exemplar may go in two directions. First, fast crossbar memory accesses along the horizontal direction are connected inside each hypernode. Our experiments presented in Section 4 indicates that this access is a small constant delay. Second, an inter-hypernode memory access is along the vertical direction through a ring. If the source and the destination memory address are not on the same ring, the access is carried out in two steps: the request first goes through the crossbar to the Functional Unit, in the same hypernode, which is associated with the global ring interconnect to the remote destination

Functional Unit, and then the request goes to the destination processor through the ring. Let L_{xbar} be the inner-hypernode network latency, N_{nodes} be the number of nodes in a ring, and let L_{ksr_ring} be the ring latency from one slot to another in the KSR-1. The network latency of a single KSR-1 ring may be expressed as:

$$L_{ksr} = N_{nodes} \times L_{ksr\_ring}, \tag{2.1}$$

and the maximum network latency for a remote access in the Exemplar may be expressed as:

$$L_{exemplar} = N_{nodes} \times 1.5 \times L_{ksr\_ring} + L_{xbar}, \tag{2.2}$$

where $1.5 \times L_{ksr\_ring}$ is the latency between two slots on the Exemplar. Considering L_{xbar} in (2.2) is trivial comparing with the first term, and substituting $N_{nodes} = 32$ in (2.1), $N_{nodes} = 16$ in (2.2), we get the latency ratio of 1.33 between the KSR-1 and the Exemplar. In other words, the network latency of a single KSR-1 ring is 1.33 times higher than the network latency of the Exemplar. The latency of the KSR-1 with multiple rings is 3.8 times higher than a single ring system [5]. Thus, the latency of the KSR-1 with multiple rings is more than 5 times higher than the network latency of the Exemplar. In addition, the processor node of the Exemplar is 5 times faster than the processor of KSR-1, which makes a lower memory access time. Thus, the memory access latency of the Exemplar is significantly lower than the one of the KSR-1. Our experiments presented in Section 4 further confirm our analysis here. (For example, the experiments show that the memory access latency of the Exemplar is about 2 times lower than the latency of a single ring KSR-1, and over 6 times lower than the latency of a double ring KSR-1.)

2.2.2 Different cache coherence Implementations

The KSR-1 maintains consistency of the data in each cache using a write-invalidate cache coherence strategy. Whenever a data item is requested by a processor for an update, every other copy of that data item, in every other cache where that subpage is located, is marked "invalid" so that the proper value of the data item can be maintained. The system maintains a list of each subpage in the local cache, along with an indication of the "state" of that subpage. The maintenance cost of cache coherence bears no additional cost in the KSR-1, over and above the latency of the ring rotation itself. In one ring, the cache invalidation cost is independent of the number of processors. The invalidation cost increases only when the number of rings increases.

In contrast, the Convex Exemplar must deal with cache coherency issues on a system with two different memory sizes. The 32 byte data size of the HP PA-RISC chip and the 64 bytes transferred when working across hypernodes. Convex has tackled this issue by using a set of tag bits that determine the status of a block of memory, the size of this block of memory matches the granularity of hypernode interaction (i.e. 64 bytes). These tag bits are inaccessible by the programmer and only modified by the cache-coherency hardware to maintain the status of memory throughout the machine. The tag bit memory is not only inaccessible by the programmer, it is not counted when referring to system memory.

Intra-hypernode coherency is handled by the first level of hierarchy of cache-coherency and is the simpler of the two levels with a set of tag bits specifying the status of memory within the hypernode.

Inter-hypernode coherency is much more difficult and has additional functionality implemented at this level. Much of the capabilities of this hierarchy are acquired using Scalable Coherent Interface (SCI) protocols as well as additional Exemplar specific tag bits. To support this coherency level the tag bits contain pointers that allow a linked list to be formed when multiple hypernodes require access to data that resides on another hypernode. A combination of pointers, Exemplar tag bits, and SCI state bits provide the hardware with the knowledge of the cache states and what actions need to be taken as data is accessed throughout the system.

2.2.3 Other implementation differences

The KSR-1 processor is a 64-bit custom superscalar RISC processor of 20 MHz, which is 5 times slower than the Exemplar's processor. Besides the differences of processor speed, the network latency, and cache coherence management, there are some others:

- *Different memory sizes:* While the KSR-1 has 32 MBytes of memory per processor, the Exemplar has a 512 MBytes of memory per 2 processors. The crossbar in each hypernode provides fast access for each of the 8 processors to the entire addressing region of the 2 GByte memory.

- *Different cache line sizes:* The cache line size of the KSR-1 is 128 bytes (a subpage) which is 4 times larger than the cache size of the Exemplar (32 bytes). Larger cache line has prefetching advantages as well as false-sharing disadvantage. It also means that although the KSR-1 has higher memory access latency, the communication bandwidth in cycles per bytes is significantly larger on the KSR-1.

- *Different views of clusters:* A hypernode in the Exemplar is viewed as a Symmetric Multiprocessor (SMP) with fast memory access. During execution, it is possible for coherence misses to be satisfied in the cache of another processor on the same hypernode, which makes the latencies of the misses much smaller than the latencies involved in crossing hypernodes. On the KSR-1, each 32-processor ring is viewed as a cluster which is 4 times larger than a hypernode in the Exemplar. In addition, the ratio between intra-cluster and inter-cluster latencies is higher in the Exemplar than the one of the KSR-1. Thus, the Exemplar is expected to have a higher NUMA effect in terms of locality management.

Comparative performance expectations: A single hypernode of the Exemplar offers superior performance due to the fast processor and low latency of the crossbar network. In comparison, using a small number of processors the KSR-1 will interact all 32 nodes in a ring with much higher latency. Although locality management is a major issue for an efficient parallel program on both machines, the Exemplar is more sensitive to it. Parallel task allocation and data distribution on the Exemplar should be accomplished very carefully. If a parallel computation does not make best use of intra-hypernode cycles, its execution on the Exemplar will not be very efficient. The cache coherence structure of the KSR-1 implies that the latency of one cache invalidation in the ring is equivalent to the latency to invalidate the entire ring, except the owner processor. Thus, even though the KSR-1 has no directory hierarchy, it does have higher cache invalidation latency than the Exemplar. This is particularly true in practice, because the number of caches to be invalidated is often much smaller than the number of processors involved in the actual computation of a program.

3 Evaluation methodology

Two types of programs were used in this empirical study. The first type consists of hot spot testing kernels and spin-lock synchronization routines. The purpose of running this group of programs on both machines is to compare and test the effects of network contention which will provide better insight into the understanding of the systems. The second type consists of two scientific application programs from the SPLASH-2 benchmark suite. A characterization study of SPLASH-2 programs was performed through execution-driven simulation, using the Tango-Lite reference generator to drive a multiprocessor cache and memory system simulator [12]. Patterns of communications and network contention are two highly architecture dependent program characteristics which could be examined precisely by program execution on a particular architecture. By tracing program executions on the Exemplar, we present overall scaling capability of the architecture. We also compared the system and application performance with the performance of the KSR-1 reported in [13], [14] and [15]. The testing programs on the Exemplar were run within the *CXpa* (CONVEX Performance Analyzer) [2] environment for collecting the execution timing results. The CXpa is a software performance monitor which has the capability to time loops and count cache misses. Although the profiler intruded slightly into the code and could possibly affect timings, we found this intrusion to be minimal.

3.1 Evaluation of network contention

Hot spot contention on a large distributed shared-memory architecture occurs when a large number of processors try to access a globally shared variable across the network. Although hot spot behavior may be program dependent and unavoidable, the resulting traffic and network contention is highly architecture dependent. For example, Pfister and Norton [7] generalize the problem as a type of network traffic non-uniformity. Their analytical models and simulation results for blocking Multistage Interconnect Network (MIN) based architectures show that the hot spot effects may severely degrade all network traffic, not just the traffic to the shared variables. The effect is defined as *tree saturation*, where traffic to the hot memories backs up at the switch and interferes with other traffic, including that to non-hot memories. Furthermore the modeling and experimental work in [15] indicates that blocking MINs are much more hot spot sensitive than slotted rings although the blocking MINs may provide a fast connection for memory access. Therefore, hot spot tests are important to show how tolerable a network structure to non-uniform network traffic patterns. Comparative network reactions to hot spot contention may also be used as a factor of architecture design.

In our experiments, the hot spots were generated in two different ways:

1. *Via read and write references:* a set of processors were used to make a target cache/memory module hot by reading and writing the same location in that cache/memory module.

2. *Via block transfer:* a set of processors were used to make a target cache/memory hot by using the block transfer operation to copy data from that cache/memory module to their local cache/memory modules. We formed the data block size as 128 bytes in the Exemplar.

In the CC-NUMA system Exemplar, the hot spot is allocated in a fixed physical memory location. The hot spot is accessed through remote accesses by other processors in the system. The following measurements will present the network/system effects to hot spots on the Exemplar: 1) latencies of normal memory accesses without a hot spot, 2) latency increment of accessing the hot spot from cool processors (processors not involved in generating the hot spot), and 3) latency increment of accessing cool memories from cool processors in the presence of a hot spot.

3.2 Cache coherence invalidation measurement

We ran experiments on the machine to measure its cache invalidation costs. We also compared the cost with the one on the KSR-1 to study the cache coherence implementation advantages and limits.

In the experiment on the Exemplar, we assign one processor to write to an array of 500,000 elements. This makes the processor have the most recent copy of the array at this time and have a copy of the array in its local data cache. All other processors then read the array to obtain a copy of it in their local data cache. Then the original processor writes to each element of the array once to cause invalidations to be done on all other copies of the element. This experiment will show us the invalidation effects from the crossbar and from the CTI rings. For characterizing the crossbar effect we use the experiment on 1 to 8 processors in the the same hypernode. For the CTI ring effects, we run the tests by scaling the processors from 16 (2 hypernodes) to 32 (4 hypernodes).

3.3 Spin-lock synchronization performance

Spin-locks are important system primitives for parallel application programs. Again, the performance of the locks are architecture dependent, which can be used to study the shared-memory systems and to identify optimal primitives for applications. Three types of synchronization locks were used in our experiments: simple spin-locks, the ticket lock and the distributed lock. The basic structure of a simple spin-lock consists of a globally shared lock allocated on a single processor to which all access is serialized. When a processor gets the lock, it immediately sets the lock to busy and starts the critical section operations. The lock is released by the processor immediately after the critical section work is finished. The rest of the processors have to busy-wait for the lock using remote memory accesses through an interconnection network. Variations of the simple spin-lock may be implemented by applying time delays between attempts to access the lock variable for each processor. The delay could be a constant (static delay) or could increase exponentially with every failed attempt to acquire the lock (exponential delay).

The ticket lock algorithm derives dynamic delay information based on the number of processors contending for a given lock in a different way. In this algorithm, two lock variables are used; the first is a "ticket" counter, which is automatically incremented by a processor to obtain its "ticket"; the second is the "current" counter, which is the number of the tickets that have access to the critical section. After obtaining a "ticket" for itself, each processor then proceeds to spin on the "current" lock until its turn arrives. Upon exiting the critical section, the processor will increment the current lock to allow the next ticket holder into the critical section. The most attractive feature is that the delays can be computed based on the difference between the held "ticket" and the "current" lock value, since this difference reflects the number of processors that will enter the critical section first. In other words, the delay function may be better predicted for reducing network traffic than with fixed functions applied to the simple spin-lock.

A distributed algorithm decentralizes locks throughout the whole system. The basic structure of the algorithm is an array, where each element in the array resides on a different node in a distributed shared-memory multiprocessor. As processors arrive at the lock, they atomically increment a counter to obtain the index of the array element upon which to spin. Each processor spins locally until its predecessor releases the lock and transfers control to it. The implementations and performance of these three types of spin-locks have been discussed in [13].

In our experiments, each processor requests lock acquisitions 1000 times in a loop. The critical sections for all these experiments were set to a constant. In addition, all experiments on both machines were run under benchmark mode. Therefore, the systems are solely used for these measurements. The experiments obviously generate a high contention environment on both networks, which may not be realistic for practical synchronization algorithm testing. However, the purpose of our work here is not to evaluate the performance of these algorithms but to observe different execution behavior of spin-locks between the two machines, and to give their architectural implications. A high contention environment could help us to understand and distinguish the network/system merits and limits.

3.4 The two benchmark programs

Barnes: a physics simulation program. Using the Barnes-Hut hierarchical N-body method, this application program simulates the interaction of a system of bodies, such as particles, in three dimensions over a number of time-steps. The main data structure of the program is a hierarchical representation of space by the Barnes-Hut tree. Two arrays are used to implement the changes of the tree in each time-step: an array of bodies that are leaves of the tree, and an array of internal cells in the tree. There are several computational phases within a time-step, each of which is executed in parallel. Some computation phases require interprocessor communication and synchronization, such as the tree-building and center-of-mass computation phases. The granularity of computation of the program is large.

Cholesky decomposition. This program factors an $n \times n$, symmetric, positive-definite matrix A into the product of a lower triangular matrix L and its transpose, L^T. It uses a column-oriented, supernodal factorization technique. Dynamic scheduling is involved to obtain supernodes. The program is locality-sensitive according the execution evaluation on DASH [8] and KSR-1 [16]. The ratio of the synchronization time to the total execution time is also much higher than that of the Barnes program.

4 Experimental results and architectural implications

4.1 Hot spot results

The hot spot experiments on the Exemplar were first conducted on one hypernode (8 processors) and then extended over to 4 hypernodes (32 processors). Thirty processor's were used to generate the hot spot. The program was run within the *CXpa* (Convex Performance Analyzer) environment in order to obtain the timings. Tables 1 and 2 report the hot spot results on a single hypernode and 4 hypernodes respectively. In Table 1, rows two and four (referencing from the cool memory) show only a small degradation in their referencing, respectively. This is important because this referencing is occurring away from the hot spot at a different memory location on the hypernode. This shows that the hot spot has a small effect on the crossbar for all other referencing that may be done by another processor. In a similar fashion, referencing from the hot memory (rows three and five) also showed only a slight performance degradation in timing. The hot spot has only a small effect on both types of references, implying that memory referencing on one hypernode during the presence of a hot spot is minor. Table 2 shows the experiments extended to a 4 hypernode system to show the inter-communication over the CTI rings in the presence of the hot spot. As can be seen in rows two and four, only a small difference in timing delays are incurred by the hot spot. Rows three and four (referencing from the hot memory) shows similar results. These statistics show that the processor speed is fast enough to handle the many requests located at the hot memory. In consequence, the referencing from the hot memory is again only slightly affected. Another comparison between the experiments on the two machines indicate that the memory access latencies increased significantly as the size of the accessed data increased on the KSR-1 [15], while the latencies showed little increase on the Exemplar. This shows the effectiveness of the high network latency and the fast processor speed of the Exemplar. The experiments also indicate that memory access latency of regular or irregular patterns is as low as in the range of 1-2 μs inside a hypernode, and around 5 μs among the hypernodes. The timings shown for the Exemplar are approximately twice the amount for what the published Convex reference timings should be. This is because of the two memory accesses that are occurring due to the array elements being a block structure as mentioned earlier.

Although there was very little change in performance between the references (hot memory versus cool memory) in the Exemplar, this is in contrast to the fixed-type (in CC-NUMA fashion) hot spot experiments on the KSR-1 reported in [15]. In [15] there is a significant difference in referencing from the cool memory versus referencing from the hot memory. On the KSR-1 referencing from the cool cache showed almost no difference in performance but was significantly affected when referencing from hot cache (delayed by about a factor of 9). The cause of the degradation is actually at the hot processor location itself. When a processor is receiving many requests, a queue will made of these

	Read (1 Blk)	Read (2 Blks)	Read (3 Blks)
Normal Reference	1.19	1.20	1.22
Hot Spot (Var) From Cool Memory	1.56	1.60	1.62
From Hot Memory	1.51	1.58	1.56
Hot Spot (Block) From Cool Memory	1.62	1.62	1.63
From Hot Memory	1.65	1.63	1.65

Table 1: Timing measurements (μs) during the presence of a hot spot on the Convex Exemplar SPP1000 for one hypernode.

	Read (1 Blk)	Read (2 Blks)	Read (3 Blks)
Normal Reference	4.56	4.58	4.59
Hot Spot (Var) From Cool Memory	4.94	5.03	5.07
From Hot Memory	5.09	5.20	5.23
Hot Spot (Block) From Cool Memory	5.16	5.20	5.25
From Hot Memory	5.19	5.20	5.24

Table 2: Timing measurements (μs) during the presence of a hot spot on the Convex Exemplar SPP1000 for multiple hypernodes.

requests. Eventually it will be able to process them. Since the processor speed is relatively low, this will obviously slow down the time for the cool processors that are trying to reference the cool variables that are located on the hot cache. For similar memory access, the high speed processor of the Exemplar shows its advantage by quickly processing the multiple requests.

4.2 Cache Coherence management and cost

We measured the cache invalidation latency as the number of processors increases on the Exemplar. Our experiments show that the invalidation cost is proportional to the number of processors involved. For example, the latency per processor in each hypernode is about 0.21 μs, and the latency per processor using the CTI rings among the hypernodes is about 0.39 μs. The slope of the CTI ring invalidation cost curve is only moderately increased beyond 8 processors which implies that the CTI rings would scale well when larger numbers of processors are involved. In practice, the number of cache invalidations in application programs is limited. In contrast, the invalidation latency cost on a single ring of the KSR-1 is a constant of 2-3 μs. If the invalidation is conducted across the two rings, the latency is about 10 times higher due to the high latency remote ring access. Thus, the cache coherence schemes in the Exemplar are expected to perform much more efficiently than the one in the KSR-1.

4.3 Spin-lock synchronization results

Figure 4 shows the performance of spin-locks on the Exemplar. The worst performance was given by the ticket lock algorithm. The fact that a waiting processor would locally spin on a local cache copy of the "current" lock caused cache invalidation when the active processor exited from the critical section. On the Exemplar, the "current" lock always resided within a memory location of one of the memory banks and this location was always updated by the exiting processor. Since the Exemplar is a CC-NUMA machine the waiting processors were able to locally spin on a copy of this lock while waiting to enter the critical section. The COMA system KSR-1 had the "current" lock migrating among the processors as they updated its value, thus, resulting in greater network traffic. As with the KSR-1, a degradation of performance on the Exemplar can be attributed to the arranging of processors within a FIFO service [13]. This could lead to a successor to the exiting processor being physically located on a different hypernode causing extra latency to be encountered by using the CTI. Further, unnatural ordering across multiple hypernodes will lead to a much higher latency for overall performance.

The distributed algorithm also performed poorly on the Exemplar for the same reasons that were given for the performance of the ticket lock algorithm. However, the distributed algorithm (which also guarantees FIFO service) still out-performed the ticket lock algorithm. This is because there are fewer invalidations that occur due to the fact that only the successor will be informed of its turn to enter the critical section. Other processors will remain undisturbed while they locally spin waiting for their turn. The delay caused by the FIFO service that occurred in the ticket lock algorithm also applies to the performance of the distributed algorithm.

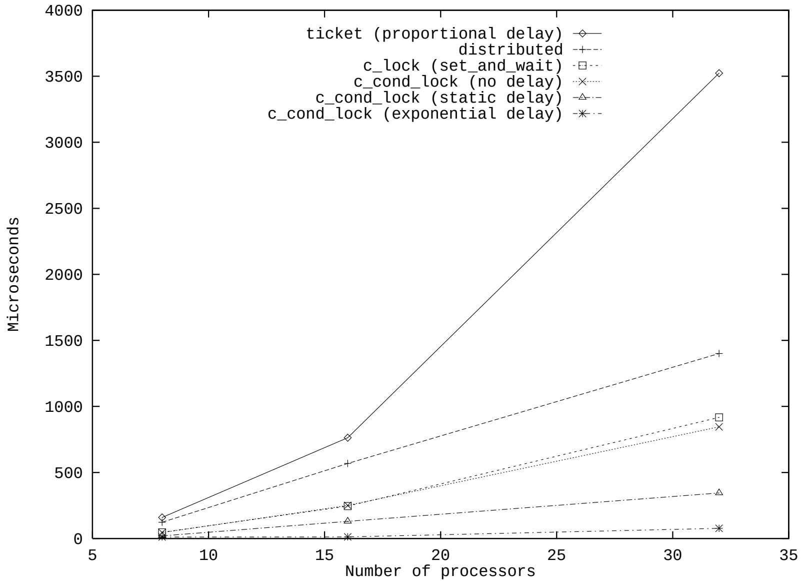


Figure 4: Performance of spin-locks on the Exemplar.

The *c_lock* (set-and-wait), *c_cond_lock* (static delay), and *c_cond_lock* (no delay) performed similar to each other for this experiment. All had high latency owing to the continually atomic request for the lock. The performance of the *c_cond_lock* (exponential delay) was much improved over the former three. In fact it proved to be the most efficient of all algorithms. This was due to the algorithm adapting itself to when it should re-attempt exclusive access to the lock. This lowered the amount of contention that lead to the physical location of the lock. The fact that waiting processors were exponentially delaying while they locally spun may have allowed the exiting processor to re-acquire the lock over the other competing processors. This phenomena was credited with the performance of the synchronization experiments that were conducted on the KSR-1.

4.3.1 Comparisons with the primitives of the KSR-1

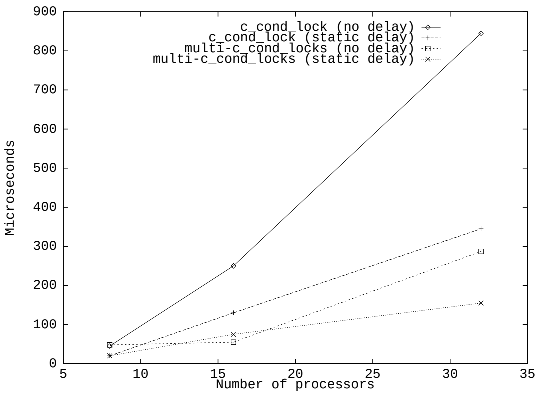
A close examination of the performance of the algorithms between Figure 4 and the the similar experimental results reported in [13] will reveal that the performances on the Exemplar exhibit similar latencies to the performances on the KSR-1 although the Exemplar has significantly lower network latency. The latencies of simple locks on the Exemplar are slightly lower, while the latencies were nearly same for the distributed lock and the ticket lock in comparison to the KSR-1. If looking at the results of the hot spot experiments for these machines, the natural implication would be that there should be much better performances on the Exemplar. The reasons lie in the hardware primitives that are being used for these algorithms. On the KSR-1 the hardware primitive *gspnwt* is being used to emulate the test-and-set operation. Request by processors are sent out onto the ring to atomically capture the subpage where the lock is currently residing. If a processor cannot atomically secure the subpage of this lock then the request is sent back to the processor indicating its failure. There is no queue of waiting processors formed for the subpage. Naturally the processor will re-attempt its request (depending upon how long the delay is) out onto the ring. Being that the ring balances out network transactions, the hot spot effect will be low, as was demonstrated in the hot spot experiments. This will cause the latency measurements to be low. When the fixed hot spot experiments were conducted, a waiting queue was formed for the requests at the memory location of the hot variable. This was the primary reason for the performance degradation when accessing from this hot cache location on the KSR-1.

On the Exemplar, a global computation is highly sensitive to the locality issue because a hypernode is fast and latency of inter-hypernode is relatively high. Simple spin-locks do not consider the hierarchical structure, which increases the access latency in the high contention case. Looking at the results of the hot spot experiments, one would believe that the latency measurements for these lock algorithms would be much lower than on the KSR-1. Although a spin-lock synchronization forms a hot spot, it is different from the hot spot experiments. The overhead for exclusive access to a lock by the hardware primitive is high when compared to the normal access of a particular data item. Continuous efforts are being made by the waiting processors to atomically have access to the lock. Thus, causing higher contention by the overhead of the primitive operation. When normal memory access is being conducted (as in the hot spot experiments) a queue of requests is created at the memory location and the speed of the processor enables it to handle the requests very fast. The processor on the Exemplar is 5 times faster than the processor of the KSR-1, thus enabling the request to be processed much faster. In summary, the hardware primitive on the KSR-1 better exploits the architecture for which it was designed. The Exemplar leaves some potential space for performance improvement of spin-lock algorithms by considering locality issues and exploiting its cluster/hierarchical structure.

On both machines, the ticket lock and distributed lock algorithms performed poorly in performance. This is because these algorithms could not take advantage of the ordering property of the ring networks.

4.3.2 A multi-lock implementation on the Exemplar

The simple locks implemented on the Exemplar require all involved processors to compete for the single lock in a fixed memory module in the system. Simultaneous accesses to the single lock from all the processors in different hypernodes could generate heavy network contention. Carefully looking into the lock algorithms, one would realize that many remote accesses are not necessary. For example, if we could arrange only one processor from each hypernode to compete for the lock at a time, the inter-hypernode traffic would be reduced significantly. The basic idea of the multi-lock implementation is to set a simple lock inside each hypernode, and to set the real lock among the hypernodes. The processor which gets the local lock can compete with other processors from other hypernodes to get the real lock. As soon as a processor gets the real lock, it releases the local lock. Another processor in the hypernode enters the local lock, and then joins the second level competition for the global lock. This implementation insures that the number of processors competing for the real lock using the global rings is no more than the number of hypernodes. Figure 5 compares the latency curves and the numbers of cache misses between the single/multiple lock implementations



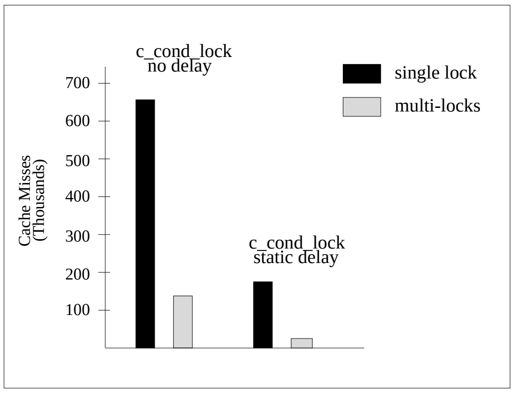


Figure 5: Comparisons of single/multiple lock implementations on the Exemplar: (left) spin-lock latencies, (right) the numbers of cache misses.

of a simple lock without delay, and a simple lock with a constant delay. The latencies and the number of cache misses are significantly reduced in the multiple-lock implementations by better exploiting the architectural locality.

A variation of the multi-lock implementation is to apply a distributed lock algorithm inside each hypernode, and a simple lock algorithm among the hypernodes. In each hypernode, the competing processes are formed into a queue based on the arrival order. Little performance gain would be expected for this implementation because a simple lock is extremely efficient in an 8 processor hypernode but a FIFO service would be enforced on each hypernode.

4.4 Execution performance of the application programs

The Barnes application and the Cholesky programs were run on the Exemplar by scaling the number of processors among the hypernodes as more processors are needed (i.e. 8 processors on one hypernode, 16 processors on two hypernodes, etc.). We selected 96,384 particles for Barnes: and system size 6,000 for Cholesky on the Exemplar. We used the *c_cond_lock* primitive to ensure mutual exclusion into critical sections throughout the programs.

As indicated in Figure 6, the performance of the Barnes program was superior on a single hypernode (almost 100% efficiency and close to the ideal linear speedup). The speedup of the Barnes program grew at a linear rate as the number of processors increased and we expect this speedup to continue well past 32 processors. The Cholesky program is locality sensitive with quite a few synchronization points. The performance was degraded as the the number of processors increased. We expect that the performance would be better when a larger size of the problem is used. This is because the communication to computation ratio of this program is large. Since the computing power of the KSR-1 machine is significantly lower than the Exemplar, using same size problem running on the both machines would not make fair comparisons. The execution results of the two programs of smaller size (16K particles for Barnes, and system size of 4K for Cholesky) on the KSR-1 reported in [8] and [16] show similar speedups on a single ring. However, the speedup on the KSR-1 sharply decreased when two rings were used. This is because the latency penalty on

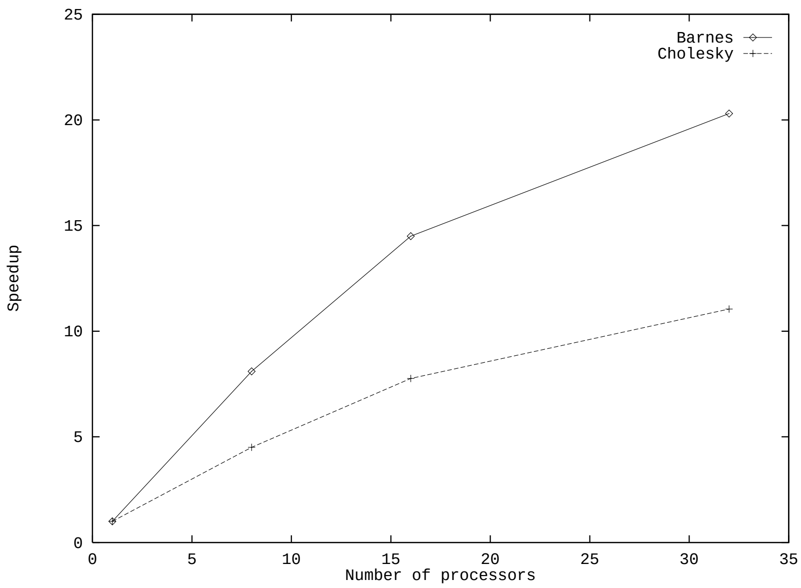


Figure 6: Speedup of the Barnes and Cholesky programs on the Exemplar.

the Exemplar is much smaller than that of the KSR-1.

5 Summary and Conclusions

Here we summarize the performance implications of the Convex Exemplar and some comparisons with the KSR-1.

- A tightly-coupled single hypernode Exemplar has superior performance owing to its fast processor and fast crossbar network. A single ring KSR-1 also performs well for application programs with large problem sizes and low communication to computation ratio, such as the Barnes application program.

- The processor speed of the Exemplar is 5 times faster than the one of the KSR-1, and its memory access latency is over 6 times lower than the latency of the KSR-1. This implies that the Exemplar is not only a faster system but also has more efficient communications. Although both architectures are, in general, not sensitive to hot spot contention due to a message control structure of ring networks, the Exemplar has two advantages based on our experiments. First, the remote memory access latency of the Exemplar is almost independent of the data size up to 384 bytes while the latency of the KSR-1 increases proportionally. Second, when the hot spot is fixed in a memory module, the memory access to the hot spot is significantly delayed in the KSR-1, but the latency only slightly increases in the Exemplar.

- Comparing a COMA to a CC-NUMA system, if the communication to computation ratio is large, the CC-NUMA system is expected to have more advantages [14]. The Barnes application program has low communication to computation ratio which favors the the COMA system. However, the performance was poor when two rings of the KSR-1 were used. We believe this has little to do with the COMA system, but is related to its higher latency of the interconnect. Since the speeds of the processors and interconnects for the two machines are significantly different, it would not be fair to compare the general merits/limits of the two memory systems based on this case study.

- The CTI rings of the Exemplar seem to work efficiently in handling cache coherence activities with respect to latency and scalability. Considering the processor to overall interconnect speed ratio, network latency, the memory system, and the cache coherence management, the Exemplar should scale well to its target number of processors, 128, for large and regular scientific applications.

- The hardware synchronization primitives of the KSR-1 exploit the ring architecture well and perform more efficiently in a high contention environment than the Exemplar.

- FIFO synchronization spin-locks, such as the ticket and distributed lock algorithms are not suitable for either architecture because they could not take advantage of the ordering property of the ring interconnect.

- The Exemplar is more sensitive to the data locality. This implies that a user needs to make a better effort to manage hypernode locality, the best use of the hypernode resources and reduction of the inter-hypernode communications in order to achieve high performance. The Exemplar CC-NUMA software environment provides options for a user to explicitly handle data locality for maximum performance improvement.

We are currently studying architectural requirements to effectively support commercial data transactions on distributed and SMP shared-memory systems.

Acknowledgement: We are grateful to Greg Astfalk of HP-CONVEX Technology Center for providing important suggestions and comments to improve the technical quality, clarity and readability of the paper.

References

[1] CONVEX Computer Corporation, *CONVEX Exemplar Architecture*, Second Edition, Document # 081-023430-001, November, 1994.

[2] CONVEX Computer Corporation, *CXpa Reference Manual*, Second Edition, Document # 710-004730-005, November, 1994.

[3] E. Hagersten, A. Landin and S. Haridi, "DDM — a cache-only memory architecture", *IEEE Computer*, September 1992, pp. 44-54.

[4] IEEE, *IEEE Standard for Scalable Coherent Interface*, IEEE Std 1596-1992, 1993.

[5] Kendall Square Research, *KSR-1 Technology Background*, 1992.

[6] D. Lenoski et al., "The DASH Prototype: logic overhead and performance", *IEEE Transactions of Parallel and Distributed Systems*, Vol. 4, No. 1, 1993, pp. 41-61.

[7] G. F. Pfister and V. A. Norton, " 'Hot spot' contention and combining in multistage interconnection networks", *IEEE Transactions on Computers*, Vol. c-34, No. 10, pp. 943-948, 1985.

[8] J. P. Singh, et. al., "An empirical comparison of the KSR and DASH multiprocessors", *Supercomputing 93*, IEEE Computer Society Press, November 1993, pp. 214-225.

[9] T. Sterling, et. al., "A performance evaluation of the Convex SPP1000 scalable shared memory parallel computer", *Proceedings of Supercomputing'95*, December, 1995.

[10] R. Thekkath, et. al., "An evaluation of a commercial CC-NUMA architecture — the CONVEX Exemplar SPP1200", *Proceedings of 11th International Parallel Processing Symposium*, IEEE Computer Society Press, April, 1997.

[11] Z. G. Vranesic, et. al. "Hector: a hierarchically structured shared-memory ' multiprocessor", *IEEE Computer*, Vol. 24, No. 1, 1991, pp. 72-79.

[12] S. C. Woo, et. al., "The SPLASH-2 programs: characterization and methodological considerations", *Proceedings of of the 22nd International Symposium of Computer Architecture*, June 1995, pp. 24-36.

[13] X. Zhang, R. Castañeda, and W. E. Chan, "Spin-lock synchronization on the Butterfly and KSR-1" *IEEE Parallel and Distributed Technology*, Vol 2, Spring Issue, 1994, pp. 51-63.

[14] X. Zhang and Y. Yan, "Comparative modeling and evaluation of CC-NUMA and COMA on hierarchical ring architectures", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 12. 1995, pp. 1316-1331.

[15] X. Zhang, Y. Yan and R. Castañeda, "Comparative performance evaluation of hot spot contention between MIN-based and ring-based shared-memory architectures", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 8, 1995, pp. 872-886.

[16] X. Zhang, Y. Yan, and K. He, "Latency metric: an experimental method for measuring and evaluating program and architecture scalability", *Journal of Parallel and Distributed Computing*, Vol. 22, No. 3, 1994, pp. 392-410.