ORNL/TM-12331

Engineering Physics and Mathematics Division

Mathematical Sciences Section

MULTI-RING PERFORMANCE OF THE KENDALL SQUARE MULTIPROCESSOR

Thomas H. Dunigan

Mathematical Sciences Section
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367
thd@ornl.gov

Date Published: October 1994

Research was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

Contents

MULTI-RING PERFORMANCE OF THE KENDALL SQUARE MULTIPROCESSOR

Thomas H. Dunigan

Abstract

Performance of the hierarchical shared-memory system of the Kendall Square Research multiprocessor is measured and characterized. The performance of prefetch is measured. Latency, bandwidth, and contention are analyzed on a 4-ring, 128 processor system. Scalability comparisons are made with other shared-memory and distributed-memory multiprocessors.

1. Introduction

In September of 1991, Kendall Square Research (KSR) installed their first multiprocessor (serial number one) at Oak Ridge National Laboratory (ORNL). The 32-processor KSR was procured as part of the High Performance Computation and Communication (HPCC) initiative. During the first year, the KSR system was evaluated and a number of applications (including Grand Challenge applications) were ported to the shared-memory KSR [7]. In September of 1992, a second ring of 32 processors was added, and we began further evaluations of the performance and scalability of the KSR multiprocessor.

It is generally believed that shared-memory multiprocessors are easier to program than distributed-memory multiprocessors, which usually employ a message-passing programming model. However, bus-based shared-memory multiprocessors usually include fewer than 30 processors, whereas distributed-memory multiprocessors often contain hundreds of processors. Thus it is with great interest that we study the KSR shared-memory multiprocessor, as it supports both shared-memory and scalability to hundreds of processors. This report summarizes our initial experiences with multi-ring KSR multiprocessors.

The distinguishing feature of the KSR multiprocessor is its shared-memory architecture. Each processor has 32 megabytes of memory. Up to 32 processors are connected to a slotted, pipelined ring, called an ACE:0. Larger configurations are formed by connecting ACE:0's to an interconnecting ring (ACE:1) with directory/routing modules (ARD's), providing up to 1,088 processors. The memories of all of the processors are part of a 40-bit virtual address space managed as a cache, where the ring is used to transport cache lines to satisfy "cache faults." Custom CMOS chips manage the cache, ring, and ring-to-ring routing. Section 2 and [18] provide more detail on the actual implementation.

The KSR shared-memory architecture is similar to the bus-based, uniform memory architecture (UMA) Sequent multiprocessors in that there is one cached address space. The KSR differs from the Sequent in that the Sequent does not have a notion of "local cache," and in that the KSR architecture is extensible beyond 30 processors. The BBN shared-memory multiprocessors, a nonuniform memory architecture (NUMA), share KSR's extensibility, but under the BBN's Uniform operating system there is no caching. Instead, a reference to a "remote" shared location will always be remote, and replication is under software control. KSR differs from the mesh-based, distributed shared-memory multiprocessors DASH [14] and PLUS [1] in that these multiprocessors do not provide strongly ordered read/write memory operations. DASH and PLUS must use explicit synchronization operations when a specific ordering is required in accessing a shared location. The KSR memory architecture is both *sequentially consistent* [12] and *strongly ordered* [4], so ordinary read/write memory operations can be used to implement synchronizations. The KSR's ring-based memory architecture is quite similar to MEMNET [2], except that a MEMNET processor has a local memory independent of the ring-based shared memory. Also, a shared memory location on MEMNET has a "home" location, a feature not required on the KSR. Delp [2] notes that the ring topology supports broadcast and provides an ordering of memory accesses so a coherency protocol is easy to implement. Both KSR and MEMNET pipeline the ring, so that more than one memory transaction may be on the ring at the same time. The Swedish DDM [10] [11] is a cache-only memory architecture (COMA) like the KSR but is based on a hierarchy of buses and directories.

Additional details of the implementation of the KSR shared-memory architecture are provided in Section 2. Section 3 describes various performance measurements of the memory hierarchy of multi-ring KSR systems. Section 4 discusses the scalability of algorithms and applications on multi-ring KSR systems. Section 5 summarizes scalability issues.

2. Implementation

The KSR ACE:0 consists of a 34-slot backplane populated with 32 processor boards, or *cells*. The remaining two slots are used for interconnects (ARD's) to the next level of the ring hierarchy (ACE:1). Each cell consists of 12 custom CMOS chips. Four Cell Interconnect Units (CIU) and four Cache Control Units (CCU) manage the shared memory. The remaining chips comprise the four functional units — the Cell Execution unit (CEU), the 30 Megabytes/second (MBs) external I/O unit (XIU), the integer unit (IPU), and the floating point unit (FPU). An instruction pair is executed on each cycle, with one member of the pair coming from either the CEU or XIU and the other member coming from either the FPU or IPU. Thus an address calculation, load/store, or branch can be executed concurrently with either an integer or floating point instruction.

Each cell runs at 20 MHz, and the floating point unit supports a pipelined adder and multiplier for a peak performance rate of 40 Megaflops per cell. Thus the KSR processor is very similar to other superscalar processors such as the Intel i860 and the IBM RS/6000 (see Appendix A). The floating point unit uses 64 64-bit registers, and the integer unit has 32 64-bit registers. The CEU uses an additional set of 32 40-bit address registers. Each cell holds a 256KB data cache and a 256KB instruction cache, and a 32 MB daughter board is attached to the back of each processor board. Release 1.0 of the OSF-based operating system consumes about 14 Megabytes on each cell. KSR calls the local memory on each processor *cache* and refers to the 256KB data cache as the *subcache*.

The memory of every cell is part of a single 40-bit virtual address space managed as a hierarchy of caches. If a processor requests a location that is not in the local data subcache then the data is fetched from the on-cell memory (cache). If the data is not in the on-cell memory, then the data is fetched from the memory of one of the other cells on the ring(s). In each case the processor stalls until the data arrives. The latencies and capacity of each level of the cache hierarchy are listed in Table 2.1 [18]. The hardware subcache is two-way set associative with random replacement and write-back and uses a 2KB unit of allocation and a 64-byte unit of transfer. The memory cache is 16-way set associative with a 16KB allocation unit and a 128-byte unit of transfer (subpage) from the ring. Various options are available for managing a "set-full" in the memory cache [18], and alternate strategies are still being evaluated.

Memory Latencies		
from:	cycles	capacity
hardware cache	2	256KB
local memory	18	32MB
ACE:0	126	1GB
ACE:1	600	34GB

Table 2.1: *Vendor-stated memory latencies and capacities.*

An ACE:0 consists of two subrings, each 128 bits wide and clocked at 40 MHz (twice the processor clock speed), implemented in a 34-slot midplane. Thus the data rate and bisection bandwidth of an ACE:0 is 1 GB/second. Ring requests are interleaved on the two subrings based on the context virtual address. The ring is managed as a circular pipeline with four stages. The 128-byte packet (plus header) occupies ten pipeline stages. The time for the leading edge of a packet to travel the ring is 3.4 μs [22]. Another 3.4 μs is consumed in launching the request, retrieving the data from the responder's cache, and delivering the data to the requesting processor. The expected latency then is about 6.8 μs. If the next generation processor ran twice as fast, the latency could be expected to drop to 4.7 μs. The interconnecting ring (ACE:1) in a multi-ring configuration is another 34-slot midplane operating at 1 GB/second, but the interconnection speed from an ACE:0 to an ACE:1 is only 100MB/second. (A system, however, can be figured with multiple ACE:1 connections (ARD's) from an ACE:0.)

KSR provides several mechanisms (prefetch, poststore, automatic prefetch) to avoid or reduce the latency of a cache fault. The programmer or compiler can use a non-blocking prefetch instruction (up to four may be in progress from each processor) to reduce the latency. It takes a processor 13 cycles to issue a prefetch, and another 23 cycles to service the reply. It takes the responding processor 23 cycles to issue the reply. These figures suggest that the maximum request rate of a processor is roughly 40MB/second, and the maximum service rate of a processor is roughly 100MB/second. (In the next section we will measure these data rates.) The poststore instruction broadcasts a subpage to all processors that have an invalid copy. The poststore can reduce coherency misses and the attendant latency. Coherency cache misses are further reduced by automatic prefetch or "read snooping." If a processor sees a memory reply on the ring for a cache line that is presently invalid in its own cache, it will update its cache with the new data. Multiple requests for the same subpage on another ring are collapsed into a single request by the ARD, thus reducing ring traffic and load on the processor that owns the subpage. Instructions to lock and unlock 128-byte subpages are provided for serializing updates to shared information.

3. Multi-ring Memory Performance

A number of low-level tests were developed to evaluate the performance and scalability of the KSR memory system and its locking and synchronizing primitives. Tests were developed to measure latency and bandwidth of single and multiple ring configurations. Contention-free tests were used to validate the vendor-stated memory performance numbers (Table 2.1). Several concurrent memory tests were used to identify bottlenecks in contending for a single shared-memory location and to measure the aggregate data rate and scalability of interconnected rings. The KSR's performance is compared with other multiprocessors whose configurations are summarized in Appendix A.

Latency and bandwidth

We measured the latency from the cache to the subcache at about 1 μs, in close agreement with the number of cycles stated by KSR in Table 2.1. (A cycle is 0.05 μs.) If the datum is not found in the local cache, the processor stalls and a request is issued on the local ring. We measured the latency between two processors on the same ring to be 6.9 μs, which gives a data rate of 18.6 MB/second with the 128-byte data packet. The rate is measured to the subcache.

If the data item is not on the local ring, the request packet is routed to an appropriate ACE:0 via the interconnecting ring (ACE:1). Thus a request packet that must travel to another ring, traverses three rings. The measured latency is 24.7 μs with a corresponding data rate of 5.2 MB/second.

If a KSR processor does not find data in its cache, then the resulting latency or access time will depend on whether the data is found on the local ring or a remote ring. If p processors are being used in the parallel application, and the needed data item is equally likely to reside on any of the $p-1$ other processors, then we can calculate the expected access time for a cache-miss on a multi-ring KSR (Figure 3.1). For a single ring ($p < 33$), the access time is just 6.9 μs. If $p > 32$, then the expected remote access time grows asymptotically toward 24.7 μs. (If another level of the KSR hierarchy were available (ACE:2), another two rings would be traversed, and we conjecture that the curve would ramp up again, asymptotically approaching 35 μs.) Although a function of the application, this increasing access time could cause the performance of an application to degrade as processors are added. Remote access times for other scalable shared-memory multiprocessors (DASH [14] and DDM [10]) also grow with the number of processors. Non-scalable shared memory multiprocessors (Cray Y-MP, Sequent, Encore) have flat remote access times.

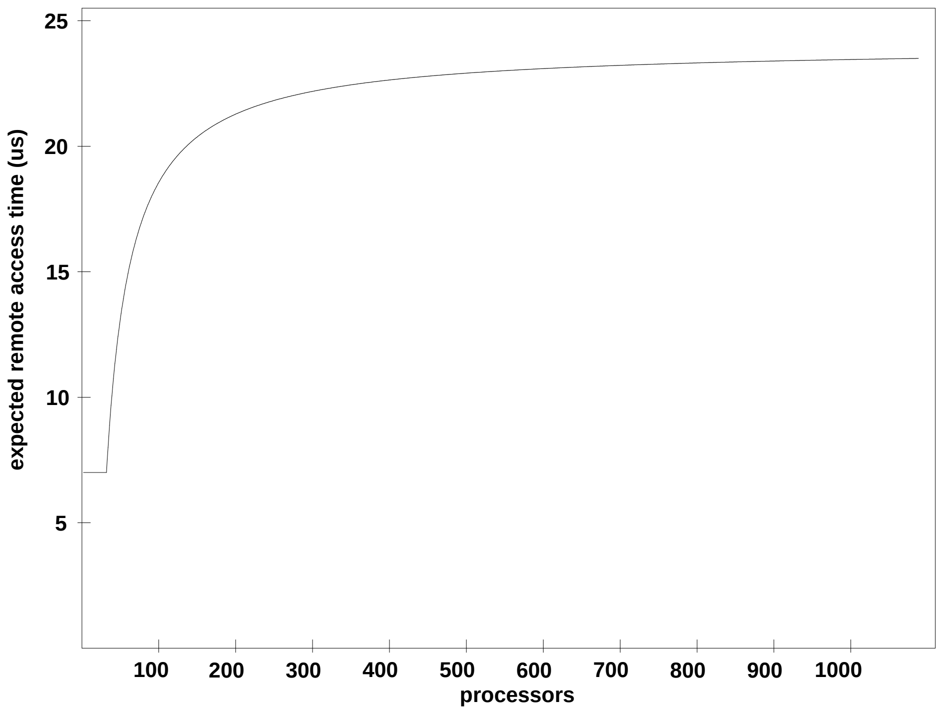

Figure 3.1: *Expected average access time for a cache miss.*

We developed some simple test codes that demonstrated that both prefetch and poststore can eliminate the latency in a cache fault. Our initial experiences with inserting prefetch and/or poststore in real applications showed little improvement in performance. Prefetch, however, can also be used to increase the effective bandwidth between two processors. Up to four prefetches per processor can be issued, providing a measured data rate of over 36 MB/second from one processor to another on the same ACE:0 [8]. If the two processors are on different ACE:0's, then the prefetch bandwidth drops to 19 MB/second.

Ring and processor contention

Several tests were constructed to measure the performance of the memory system under concurrent use, in an effort to determine if the processor or interconnect ring was the limiting factor in memory performance.

In [8] we measured the prefetch data rate between independent processor pairs running concurrently on a single ring ($P_1 \rightarrow P_2, P_3 \rightarrow P_4, \ldots$). We performed the same tests on multiple-ring configurations and found that the aggregate data rate still scales linearly with the number of processor pairs, with the data rate from one processor to another at 36 MB/second. For 16 pairs (one ACE:0), the aggregate rate is 573 MB/second. For 64 pairs (four ACE:0's), the aggregate rate is 2.2 GB/second (Figure 3.2). The aggregate data rate is not affected by multiple rings, since no pair crosses a ring boundary in our pairing scheme.

However, if one member of each pair is on a different ring ($P_1 \rightarrow P_{32}, P_2 \rightarrow P_{33}, \ldots$), then the single-pair prefetch rate drops to 18 MB/second. For multiple cross-ring pairs, the aggregate rate increases linearly up to 90 MB/second for 10 processors (5 pairs), but then flattens and declines as the 100 MB/second link between the ACE:0 and ACE:1 becomes saturated (Figure 3.3).

Another test was an exchange of data between processor pairs using prefetch ($P_1 \leftrightarrow P_2, P_3 \leftrightarrow P_4, \ldots$). In this test, a processor both provides and requests data. In the exchange, the provider data rate drops from 36 MB/second to 25 MB/second, but the aggregate rate (provider + requester) climbs to 50 MB/second. (Note, if the two processors are on separate rings, the aggregate data rate for an exchange drops to 32 MB/second.) For 16 pairs (one ACE:0) doing exchanges, the aggregate rate is 811 MB/second. Again, the rate scales linearly even for multiple rings, since a pair does not cross a ring boundary.

If we measure the data rate of a shift operation between processors using prefetch ($\rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \cdots \rightarrow P_n \rightarrow$), then we detect a slight reduction in the aggregate rate when a ring-crossing is required. For example, for 34 processors the aggregate exchange data rate is 850 MB/second, but the aggregate shift data

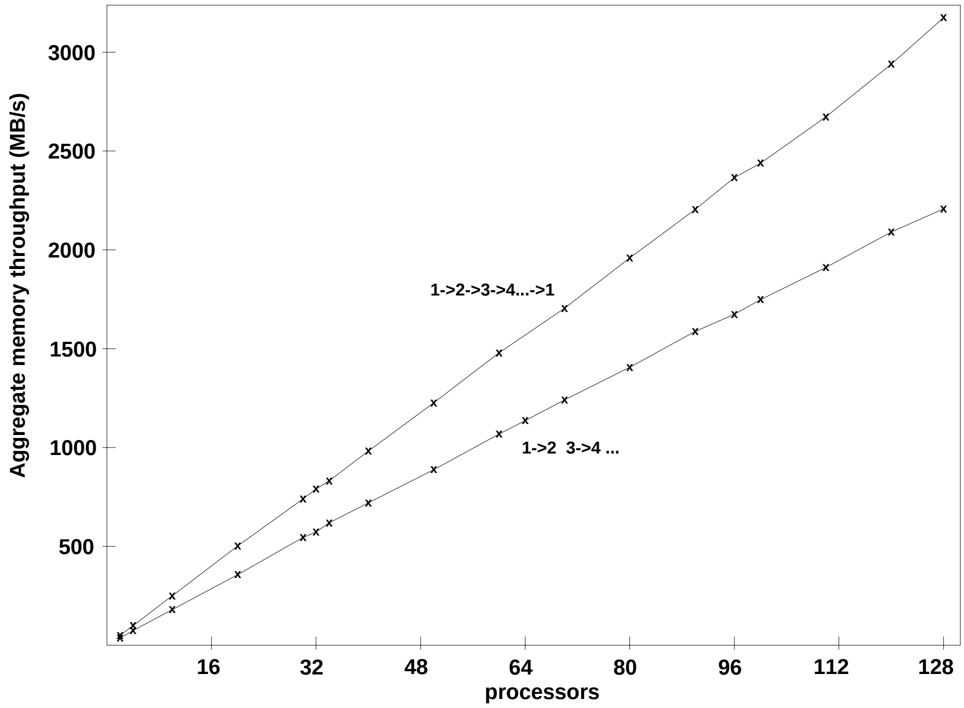

Figure 3.2: *Aggregate memory throughput for concurrent prefetch.*

rate is 834 MB/second. The shift data rate between processor 32 and 33 and between 34 and 1 is only 16 MB/second because of the ring crossing.

Additional tests were constructed to measure the speed that a single processor can service memory requests from other processors. From our prefetch measurements, the maximum data rate between two processors on the same ring is 36 MB/second. If we have multiple processors requesting distinct data from a single processor, we measure the maximum service rate for a processor at about 75 MB/second (Figure 3.4). The figure shows the aggregate data rate for various numbers of requesting processors with and without prefetch. We performed various tests with the server processor idle and with it touching local pages. The activity of the server processor seemed to have little effect on the rate at which it serviced memory requests from other processors. However, Figure 3.4 shows that the service processor is slowed by the memory requests of the other processors. Notice that the prefetch server's data rate (the rate at which it touches local pages) is slower than the server without prefetch. Since the data is already local, the extra prefetch instructions have only a detrimental effect on the server processor. Thus prefetch may not always benefit an application, and the optimal

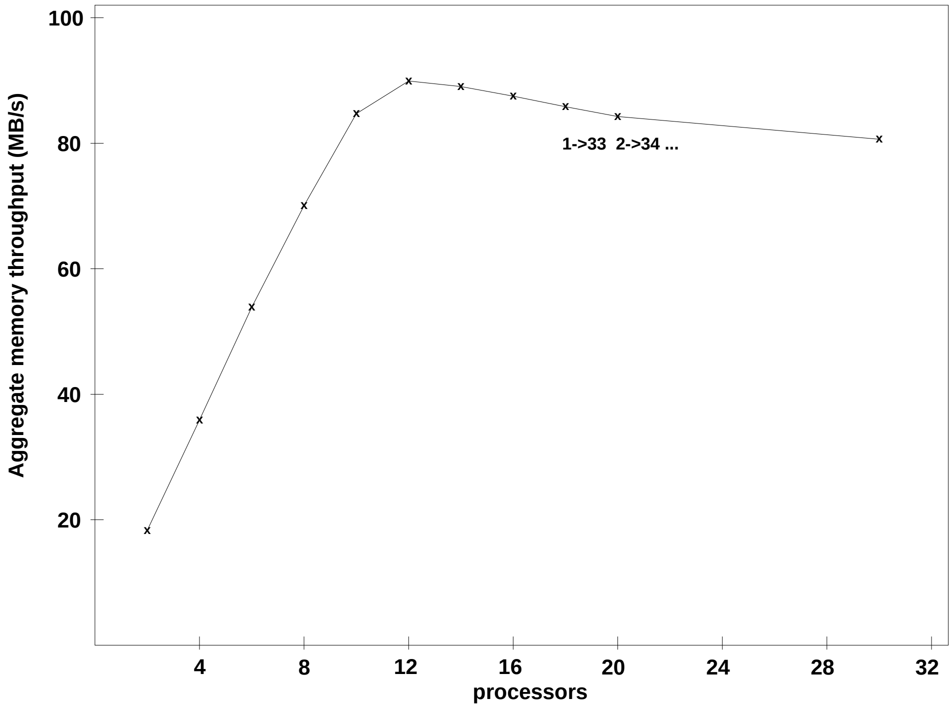

Figure 3.3: *Aggregate memory throughput for concurrent prefetch across two rings.*

use of prefetch is still an open research question [15]. In some cases, poststore can also degrade overall performance [21].

In summary, with the present speed of the processors and ring, the speed of the processors seems to be the limiting factor in memory performance. The one-gigabyte-per-second ring can sustain eight million transactions per second [19], and none of our concurrency tests were able to saturate the ring.

Memory contention

The previous tests had processors competing for the same ring or the same "server" processor, but not for the same memory location. A number of tests were used to see how the KSR scales when multiple processors try to update one or more shared locations or hot spots. Our worst-case contention test is p processors continually updating the same shared location. Like other shared memory multiprocessors (Sequent and BBN), the average time for the KSR to update a single shared location grows linearly with p [8] even across multiple rings. The shared location bounces from processor to processor as each processor

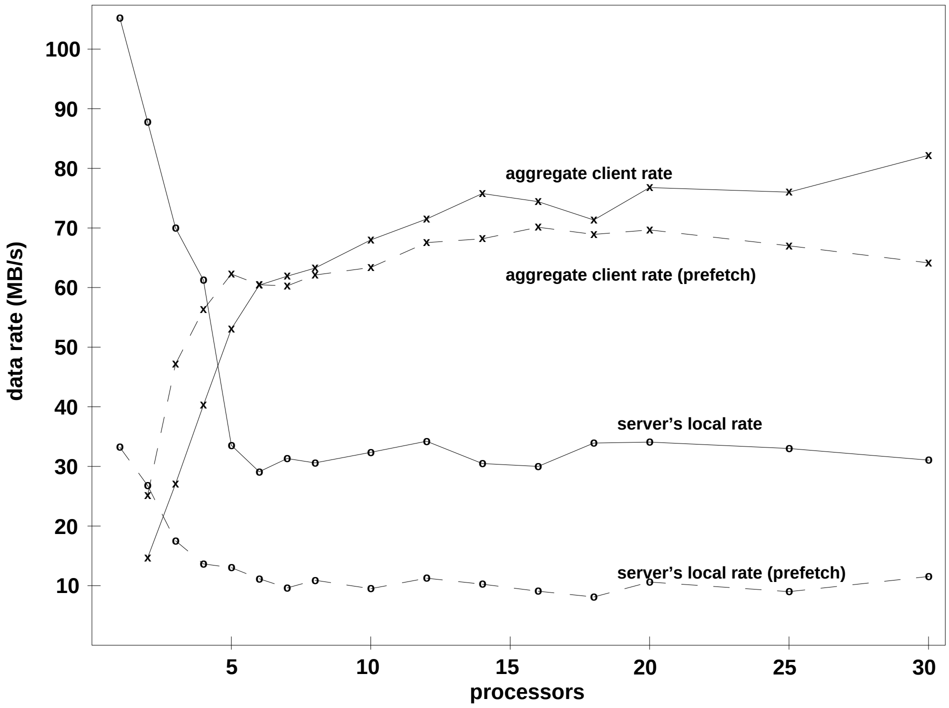

Figure 3.4: *Aggregate data rate for multiple processors faulting independent data from a single "server" processor.*

updates the value. A more realistic test is having p processors update m shared locations some fraction of the time, f, the remaining time being used to access local variables. We developed a simple, probablistic, hot-spots test where p processors continuously update the m shared variables. Each processor selects the m variables in a random order. The results of this test were consistent with the results of our implementation of Nanda's parameterized shared-memory test [16]. Nanda's test permits us to select the fraction, f, of the workload that will be shared-variable updates and to randomly select which shared variable to update for each pass through the workload. A workload is an amount of local computing randomly interspersed with a given fraction, f, of shared memory updates to $m = p$ shared variables. Figure 3.5 shows the average workload time versus the number of processors for a 1% and 10% shared update fraction using p shared variables on p processors. Nanda argues that the workload time grows as $1 - (1 - f/m)^p$. Our measurements on the Sequent, BBN, and the single-ring KSR confirm this relationship [8]. However, for the multi-ring configuration illustrated in Figure 3.5, the contention curve is a series of curves. The access times

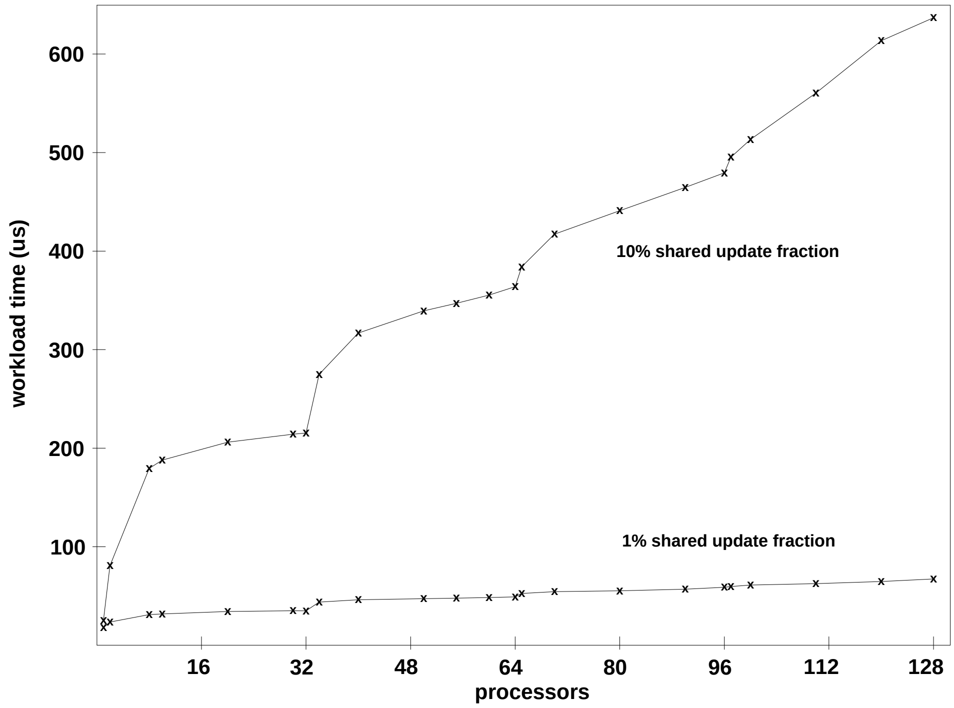

Figure 3.5: *Work load time for p processors and p shared variables.*

climb rapidly at ring crossings, as processors on the "new" ring will most likely find the shared variable they need to update on another ring and will experience the longer multi-ring latency. The average workload time increases with p even though the total amount of computation is increasing linearly with p as well, that is, the per-processor amount of computation is constant. This suggests that applications with shared-memory updates need to have a computational component that increases faster than p for performance to scale.

Locks and barriers

Updates to shared variables are usually controlled by locks. On the KSR a hardware lock instruction, *gsp*, is provided to lock a 128-byte block of memory. The *gsp* is the basis of the slower, but more socially acceptable, *mutex* library routines. In the absence of contention, the average time for lock and unlock is 2.5 μs using *gsp*. If two processors on the same ring are contending for the lock, then the average time is 14 μs. If each processor is on a different ring, the lock-unlock time is 32 μs. Figure 3.6 shows the lock-unlock times when p processors contend for the same lock.

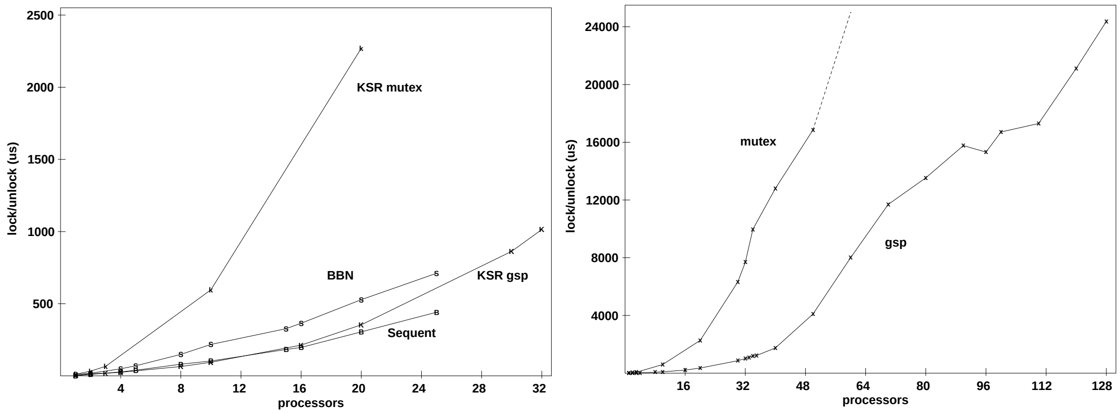

Figure 3.6: *Lock-unlock time.*

Nanda [16] argues that the expected lock-unlock time is proportional to $(p-1)(k+t_c)$, where k is the time the lock is held and t_c is the time to access the shared location that is the lock. As we noted earlier, the time to update a single shared location by p processors grows linearly with p, so the lock-unlock time grows quadratically with respect to the number of processors. For the KSR, the coefficients of the quadratic change for each additional ring of processors contending for the lock. Clearly, an application that contends for a single lock may not scale well.

KSR provides a barrier subroutine to synchronize processes or threads. A simple implementation would use a single lock, but as the preceding paragraph shows, such an implementation would not scale well. KSR provides the programmer with an n-ary tree barrier which scales roughly linearly with the number of processors. Figure 3.7 shows average barrier delay for a four-ring KSR using a tree width of four. Barrier times for the iPSC/860 and the Delta are provided for comparison. By contrast, barrier synchronization is provided by hardware on the CM5 and requires only a few microseconds.

4. Scalability Experiences

A simple parallel implementation of a dense Cholesky matrix factorization in C exhibits the effect of ring-crossings on performance. Figure 4.1 shows the megaflop rate for factoring a 1024×1024 and 2048×2048 matrix on a 4-ring KSR. This implementation uses a single lock (*gsp*) to control a queue of columns to be done, and a set of spin-locks to control when a column is ready for access by other

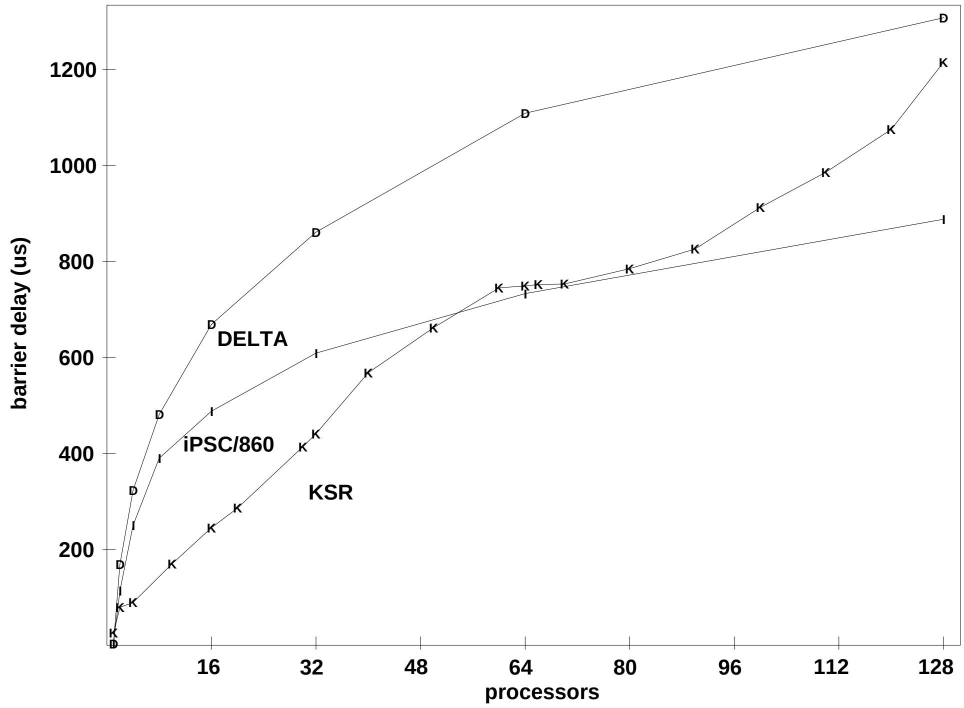

Figure 3.7: *Average barrier delay.*

processors. Poststore had no noticeable effect on the spin-locks, probably because the processors were already spinning on the lock by the time it was released by the owning processor. Performance for the 1024×1024 matrix flattens for larger number of processors because the amount of computation per processor is not sufficient to hide the overheads of parallelism and memory latency.

By contrast to the simple dense Cholesky, the 1000×1000 LINPACK represents the vendors best effort at parallelizing a code. Figure 4.2 compares KSR's performance (coded by Nick Camp of KSR) to the Intel Delta and iPSC/860 [3]. On a single processor, the machines achieve about 31 Mflops. The difference in performance among the machines is mostly attributed to bandwidth (the Delta and the iPSC/860 have roughly the same latency) and to the lower latency of the KSR. Again note the dip in performance at the ring-crossing. Table 4.1 compares LINPACK performance of the three multiprocessors for the largest matrix size that gives the optimal performance. With more computation and larger messages, the Delta performs slightly better than the KSR.

The first parallel application to be ported was a 19,000 line FORTRAN code that calculates energy densities for high temperature superconducting materials

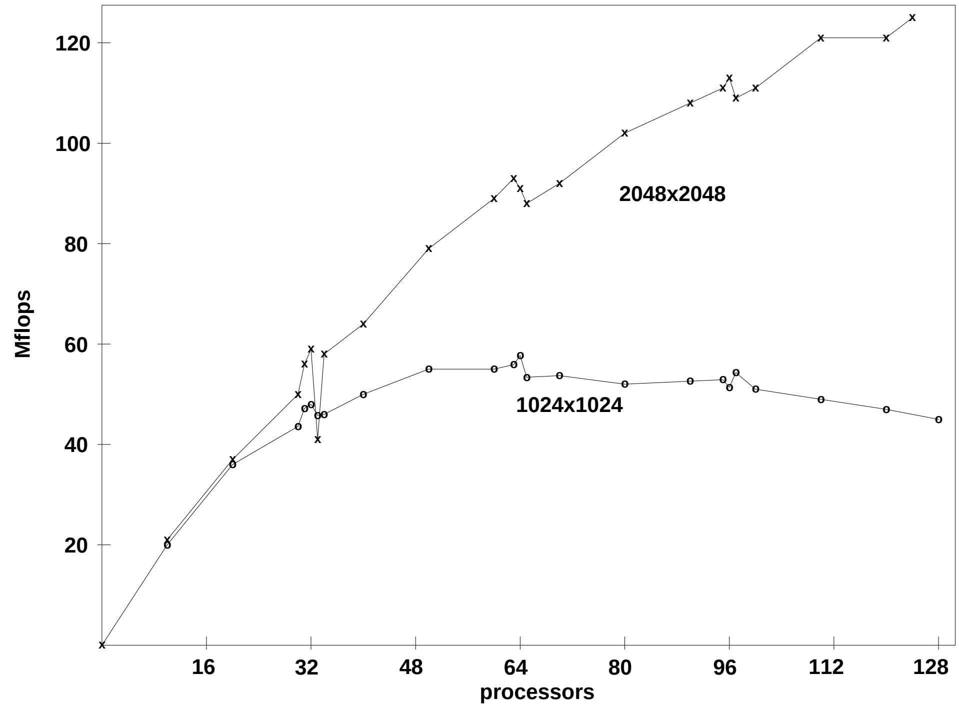

Figure 4.1: *Megaflops for dense C Cholesky.*

[9]. The code already contained explicit Cray parallel micro-tasking directives, so porting to the KSR merely required changing the names and arguments for thread creation and joining and for lock management. The parallel version exhibited near linear speedup and achieved 540 Mflops on 60 processors. The application is embarrassingly parallel, so it performs well on most parallel processors. The computational kernel (based on a complex ZAXPY) runs at 9.1 Mflops on a single KSR processor. (For comparison the kernel runs at 24.4 Mflops on an Intel Delta processor and at 39.7 Mflops on the new Intel Paragon processor.) We used only 60 of 64 processors on a two-ring KSR because KSR performance usually scales

Massively Parallel LINPACK				
	64 CPUs		128 CPUs	
Multiprocessor	N	Gflops	N	Gflops
KSR	8K	1.2	10K	3.4
Delta	8K	1.7	12.5K	3.5
iPSC/860	9K	1.4	12K	2.6

Table 4.1: *Massively parallel LINPACK for 64 and 128 processors.*

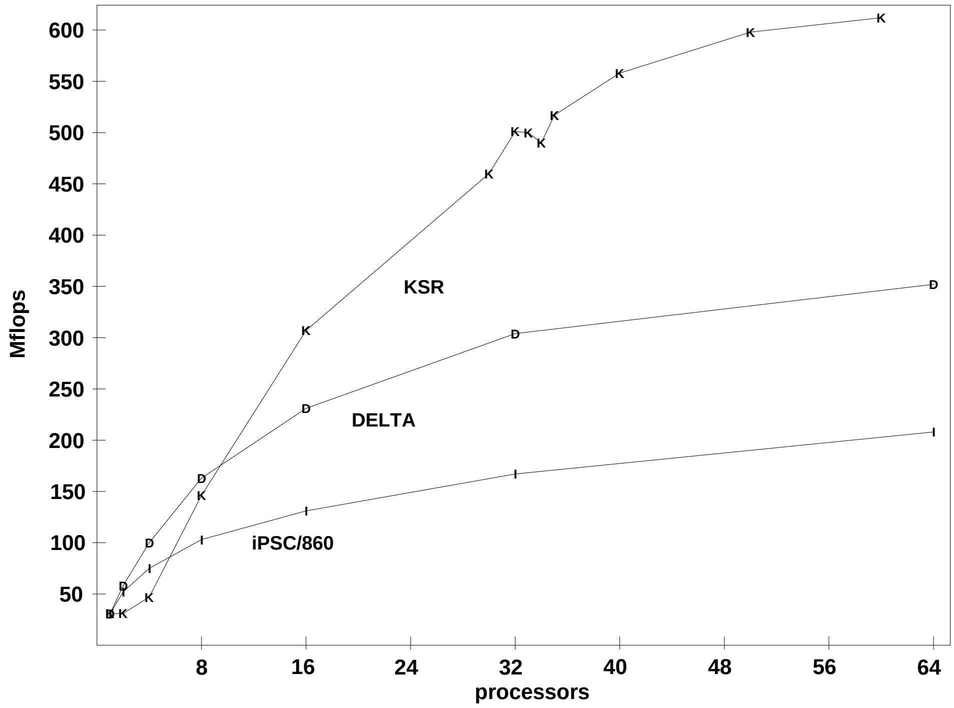

Figure 4.2: *Megaflops for* 1000 $\times$ 1000 *double-precision LINPACK.*

better if the user avoids the processors with disks and network connections.

Serial and parallel versions of a sparse-matrix Cholesky using SPARSPAK ([17]) were ported to the KSR. The sparse Cholesky code includes implicit parallel directives for the Cray and Sequent, and those directives map nicely into corresponding KSR directives. The latency of the KSR memory prevents the KSR from achieving the speedups of the Sequent or Cray (Figure 4.3), but the memory architectures of the Cray and Sequent are not scalable. (For the same number of processors, the KSR outperforms the Sequent when measuring actual run-time of the sparse Cholesky.) A global climate modeling code (CCM2) based on Cray directives was ported and scaled well up to 32 processors on the KSR.

A parallel implementation of a hypercube simulator was ported to the KSR[5]. The simulator uses *fork()* to create sub-tasks and System V shared memory and semaphores to communicate among the sub-tasks. Performance for this simulator was poor and did not scale well on the KSR compared to the Sequent implementation. Only a single shared buffer was used for message exchange, so the longer latency of the KSR (compared to the Sequent) and the contention for a single lock degraded performance. By contrast, the implementation of Argonne's

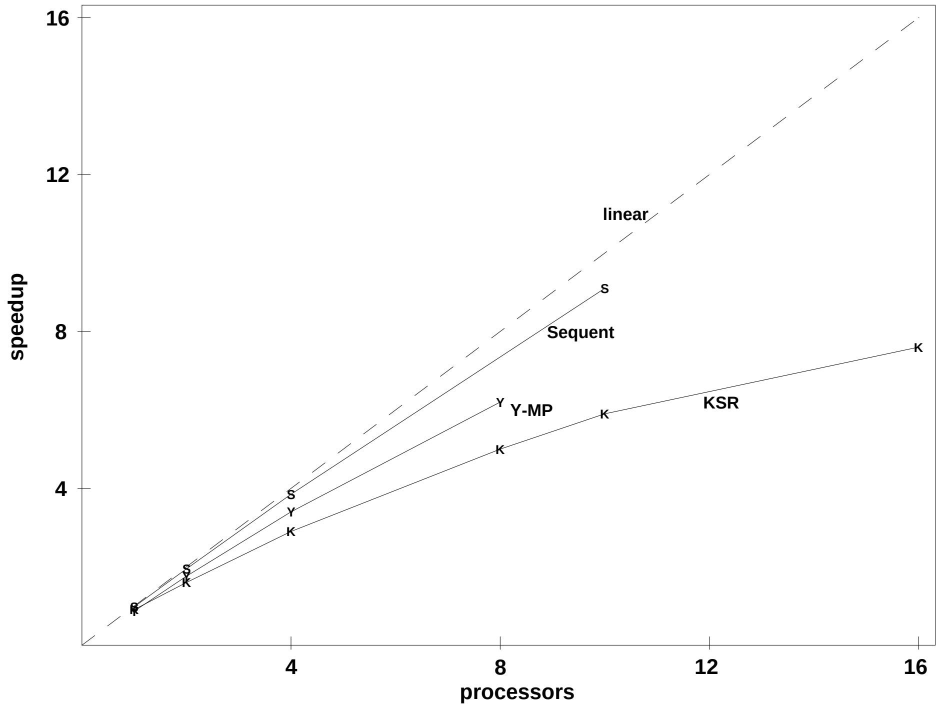

Figure 4.3: *Speedups for sparse Cholesky factorization.*

tcgmsg message-passing library performs well on the KSR. *Tcgmsg* uses a pool of shared-memory message buffers for each processor and has separate locks for each buffer. Two KSR processors on the same ring passing messages using *tcgmsg* have a message-passing latency of 71 μs and a bandwidth of 7.7 MB/second. If the two processor are not on the same ring, the latency climbs to 162 μs and the bandwidth drops to 3.4 MB/second. (By comparison, the Delta has a latency of 60 μs and an 8 MB/second bandwidth; see Appendix A.)

Figure 4.4 shows the KSR performance for three of the benchmarks from the Stanford Parallel Applications for Shared Memory (SPLASH) suite [23]. This suite has been used to compare simulations of other scalable shared-memory multiprocessors (DASH and DDM) with the Encore (a bus-based shared-memory multiprocessor similar to the Sequent). The *water* code simulates the movement of 384 water molecules. The *mp3d* code simulates a wind tunnel with 3,000 particles for 1,000 time steps. The *Cholesky* is a sparse matrix factorization using the *bcsstk15* input matrix. The KSR speedups are somewhat less than the Encore [23] and appear comparable to the simulated results for the DASH [23] and the DDM [10]. The problem sizes are small and need to be increased for larger number of processors.

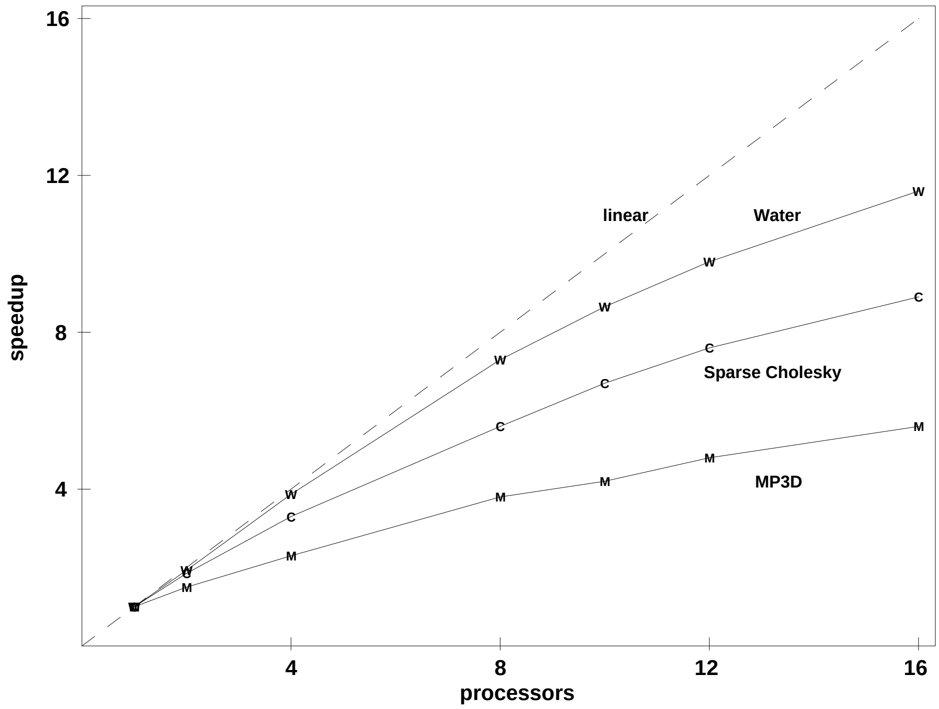

Figure 4.4: *Self-relative speedups for various SPLASH benchmarks.*

5. Summary

The low level tests described in this report have measured the performance of the KSR memory system with and without contention for the ring, for a processor, and for one or more shared locations, demonstrating that the memory system performance is limited by the speed of a processor. Applications that do not contend for shared locations should scale well, but applications that contend for shared locations will probably see performance drop as processors are added, particularly at ring crossings. For applications whose computational component scales faster than p, performance may continue to improve with increasing p. Applications ported from non-scalable shared-memory multiprocessors will likely exhibit smaller speedups on scalable shared-memory multiprocessors like the KSR, particularly multi-ring applications. We found that applications from smaller shared-memory multiprocessors often used a single lock. For those applications to scale to the larger number of processors available on a KSR, the applications needed to be modified to use a hierarchy of locks and often required a re-design of the use of shared variables. Performance may be further increased by using prefetch or poststore and by improving data locality.

Acknowledgements

A special thanks to the Advanced Computing Research Facility at Argonne National Laboratory for providing access to the BBN TC2000 and Sequent Symmetry and to the Theory Center at Cornell University for stand-alone time on their 128 processor KSR. Arun Nanda of Michigan State University graciously provided source to the workload program used in [16].

6. References

[1] R. Bisiani and M. Ravishankar. PLUS: A distributed shared-memory system. In *International Symposium on Computer Architecture*, pages 115–124, 1990.

[2] G. S. Delp. The architecture and impelmentation of MEMNET: A high-speed shared-memory computer communications network. Technical report, University of Delaware, 1988. Ph.D. Dissertation.

[3] J. Dongarra. Performance of various computers using standard linear equations software. Technical report, University of Tennessee, January 1991. CS-89-85.

[4] M. Dubois, C. Scheurich, and F. Griggs. Memory access buffering in multiprocessors. In *13th International Symposium on Computer Architecture*, pages 434–442, 1986.

[5] T. H. Dunigan. A message-passing multiprocessor simulator. Technical report, Oak Ridge National Laboratory, Oak Ridge, TN, 1986. ORNL/TM-9966.

[6] T. H. Dunigan. Hypercube clock synchronization. Technical report, Oak Ridge National Laboratory, 1991. ORNL/TM-11744.

[7] T. H. Dunigan. Performance of the Intel iPSC/860 and Ncube 6400 hypercubes. *Parallel Computing*, 17:1285 – 1302, 1991.

[8] T. H. Dunigan. Kendall Square multiprocessor: Early experiences and peformance. Technical report, Oak Ridge National Laboratory, 1992. ORNL/TM-12065.

[9] G. A. Geist, B. W. Peyton, W. A. Shelton, and G. M. Stocks. Modeling High-Temperature Superconductors and Metallic Alloys on the Intel iPSC/860. In David W. Walker, editor, *Proceedings of the Fifth Distributed Memory Computing Conference*, pages 504–512, 1990.

[10] E. Hagersten. Toward scalable cache only memory architectures. Technical report, Swedish Institute of Computer Science, 1992. Ph.D. Dissertation.

[11] E. Hagersten, A. Landin, and S. Haridi. Ddm – a cache-only memory architecture. *Computer*, September:44–54, 1992.

[12] L. Lamport. Solved problems, unsolved problems, and non-problems in concurrency. *Operating Systems Review*, 19:34–44, 1985.

[13] R. P. LaRowe and C. S. Ellis. Experimental comparison of memory management policies for numa multiprocessors. Technical report, Duke University, April 1990. CS-1990-10.

[14] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. In *International Symposium on Computer Architecture*, pages 148–159, 1990.

[15] T. Mowry and A. Gupta. Tolerating latency through software-controlled prefetching in shared-memory multiprocessors. *Parallel and Distributed Computing*, 12:87 – 106, 1991.

[16] A. K. Nanda, H. Shing, T. Tzen, and L. M. Ni. Resource contention in shared-memory multiprocessors: A parameterized performance degradation model. *Parallel and Distributed Computing*, 12:313 – 327, 1991.

[17] E. Ng and B. Peyton. Block sparse cholesky algorithms on advanced uniprocessor computers. Technical report, Oak Ridge National Laboratory, Oak Ridge, TN, 1991. ORNL/TM-11960.

[18] Kendall Square Research. *KSR1 Principles of Operations*. Kendall Square Research, Waltham, MA, 1991. KSR 8/1/91 Rev 5.5.

[19] Kendall Square Research. *KSR Technical Summary*. Kendall Square Research, Waltham, MA, 1992.

[20] R. D. Rettberg, W. R. Crowther, P. P. Carvey, and R. S. Tomlinson. The monarch parallel processor hardware design. *Computer*, 23:18–30, April 1990.

[21] E. Rosti, E. Smirni, T. Wagner, A. Apon, and L. Dowdy. The KSR1: Experimentation and modeling of poststore. Technical report, Oak Ridge National Laboratory, Oak Ridge, TN, 1992. ORNL/TM-9966.

[22] J. Rothnie, KSR, 1992. personal communication.

[23] J. Singh, W. Weber, and A. Gupta. SPLASH: Stanford parallel applications for shared-memory. Technical report, Stanford University, Stanford, CA, 1991. CSL-TR-91-469.

Appendix

A. Comparative Architectures

The KSR is compared with a number of other processors in this report. This appendix summarizes the architectures and configurations used in the comparisons. The processor architecture of the IBM RS/6000 and the Intel i860 share several common characteristics with the KSR processor: independent integer and floating point units and pipelined independent adder/multipliers in the floating point units. The Sequent and BBN parallel processors provide contrasting shared-memory architectures. Finally, the TMC and Intel distributed-memory parallel processors provide contrast to KSR's shared-memory model.

BBN TC2000

The BBN TC2000 at Argonne National Laboratory (ANL) is a 45 processor shared-memory parallel processor. Each processor is a Motorola 88000 running at 20MHz with 16 MB of memory fronted by a 16KB data cache and a 16KB instruction cache. All of the memories are interconnected by a 2-stage 8-way switch. The system can be expanded up to 512 processors. The Uniform programming environment (under nX 2.0.6) provides the program with both local and explicitly allocated shared memory. The shared memory may be allocated in another processor's memory, and thus a non-uniform memory access (NUMA) model is supported. In the absence of contention, a remote reference typically takes less than two microseconds, and a single channel of the switch has a bandwidth of 40 MBs [20]. The architecture could be used with other memory management policies [13]. Compiles on the BBN were done with -O -lus. LINPACK 100×100 double-precision was 1.0 Mflops using -OLM -autoinline. Dhrystone (v1.0) was 19.4 Mips.

Intel iPSC/860 and DELTA

The Intel iPSC/860 hypercube and DELTA mesh distributed-memory parallel processors both use the 40 MHz i860 processor. The i860 has an 8KB data cache and 8 MB of memory (16 MB on the DELTA) with a memory bandwidth of 160 MBs. The processor has independent integer and floating point units, and the floating point unit has an independent pipelined adder and multipler for a peak rate of 64 Mflops. The iPSC/860 has a maximum configuration of 128 processors. The processors are interconnected with a hypercube network with a latency of about 60 microseconds and a bandwidth of 2.8 MBs per channel [7]. The DELTA is a mesh connected parallel processor located at Caltech with a maximum configuration of 512 processors. The mesh has a latency of about 50 microseconds and a measured bandwidth of about 17 MBs/channel [6]. The processors run NX 3.3 and compiles were done with -O3 -Knoieee on a separate "host" processor. LINPACK 100 $\times$ 100 double-precision was 6.5 Mflops [3]. Dhrystone (v1.0) was 29.4 Mips.

Sequent Symmetry

The 26 processor Sequent Symmetry located at ANL is based on 80386/387 processors (16 MHz) with a Weitek 3167 floating point co-processor. Each processor has a 64KB cache, and 32 MB of memory is shared by all processors on a 54 MBs bus. The maximum configuration is 30 processors. The processors run Dynix 3.1.2, and compiles were done using -O. LINPACK 100 $\times$ 100 double-precision was 0.37 Mflops [3]. Dhrystone (v1.0) was 3.6 Mips.

TMC CM-5

A Thinking Machines CM-5 processor nodes consists of a 32 MHz SPARC RISC processor with four vector units and 16 MB of memory. The nodes are connected by a 20 MB/second hypertree data network. A separate control network provides support for broadcast, reduction, and synchronization. Message-passing latency and bandwidth times were measured using CMMD 2.0.

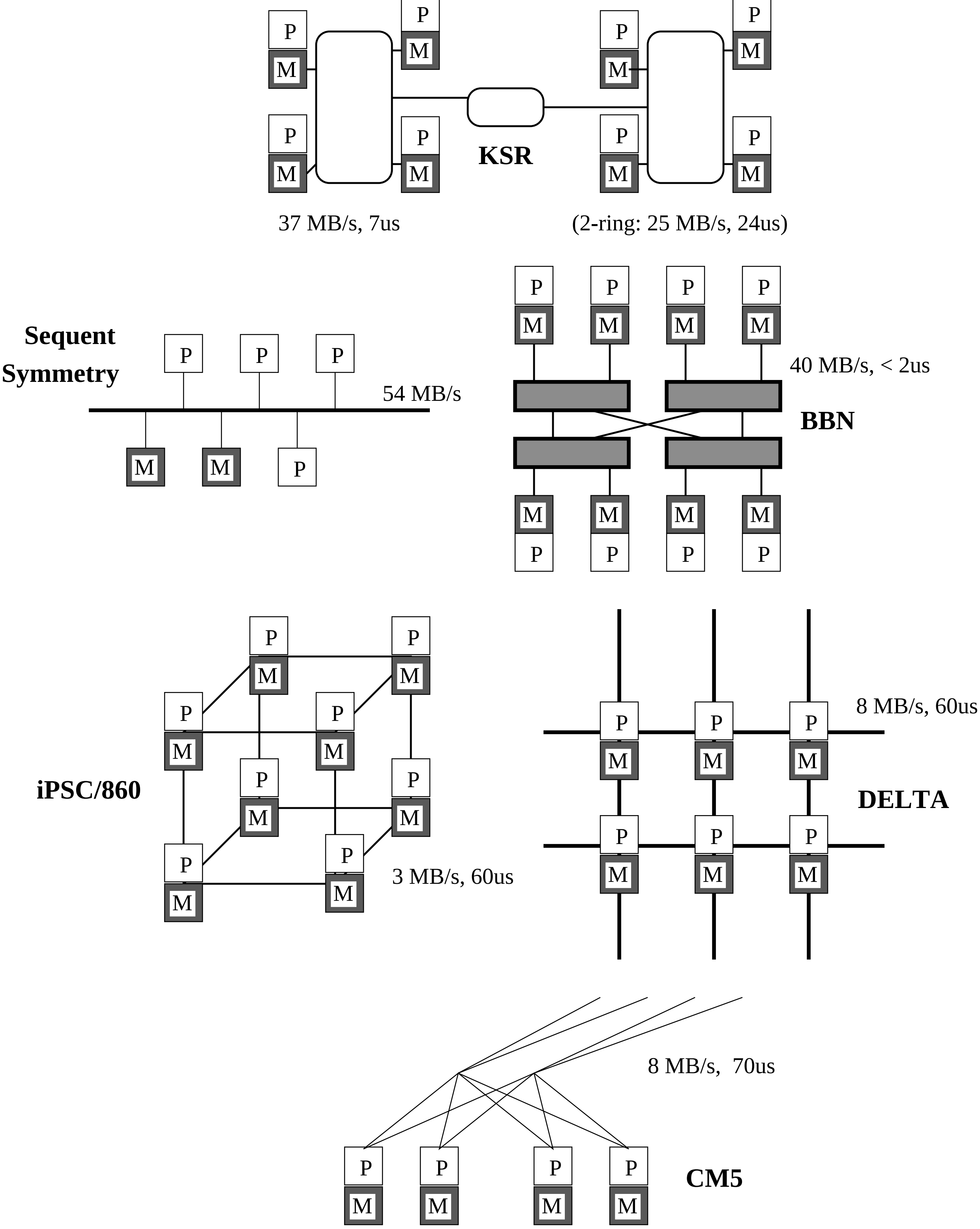

Figure A.1: *Multiprocessor latency and bandwidth.*