ORNL/TM-12065

Engineering Physics and Mathematics Division

Mathematical Sciences Section

KENDALL SQUARE MULTIPROCESSOR: EARLY EXPERIENCES AND PERFORMANCE

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-AC-05-84OR21400

Contents

KENDALL SQUARE MULTIPROCESSOR: EARLY EXPERIENCES AND PERFORMANCE

Thomas H. Dunigan

Abstract

Initial performance results and early experiences are reported for the Kendall Square Research multiprocessor. The basic architecture of the shared-memory multiprocessor is described, and computational and I/O performance is measured for both serial and parallel programs. Experiences in porting various applications are described.

1. Introduction

In September of 1991, a Kendall Square Research (KSR) multiprocessor was installed at Oak Ridge National Laboratory (ORNL). This report describes the results of this initial field test. The performance of the KSR shared-memory multiprocessor is compared with other shared-memory and distributed-memory multiprocessors, using synthetic benchmarks and real applications. Performance figures must be considered preliminary, since the KSR system was in its first field test.

The KSR multiprocessor runs a modified version of OSF/1 (Mach). To the user, the KSR system appears like typical UNIX^{TM}, but providing performance advantages similar to those provided by the Sequent Symmetry and BBN TC2000 multiprocessors and providing scalability similar to the Intel iPSC/860 and DELTA. Piped processes and background jobs can utilize the multiprocessor architecture to provide improved throughput and response time.

A programmer on the KSR system is provided with a parallel *make* and with automatic parallelization for FORTRAN. The programmer can assist the automatic parallelization (a FORTRAN pre-processor from Kuck Associates) with compiler directives, or can do explicit parallelization using the *pthread* subroutine library. The *pthread* library is provided to the C programmer along with language extensions to manage shared variables.

Shared Memory

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 a Ring:0. Larger systems are formed by connecting Ring:0's to an interconnecting Ring:1, providing up to 1,088 processors. The memory of all of the processors is 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. The KSR architecture and chip set are designed specifically to support a shared-memory multiprocessor. Section 2 and [18] provide more detail on the actual implementation.

The KSR shared-memory architecture is similar to the bus-based Sequent systems in that there is one cached address space, but it differs from the Sequent in that the Sequent does not have a notion of "local cache," and the KSR architecture is extensible beyond 30 processors. The BBN shared-memory multiprocessors share KSR's extensibility, but under the BBN's Uniform system there is no caching, rather 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 systems DASH [14] and PLUS [1] in that these systems 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 system 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 system is quite similar to MEMNET [2], except that MEMNET still has a local memory for each processor 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.

Additional details of the implementation of the shared-memory architecture are provided in Section 2 along with a summary of the processor architecture and implementation. Section 3 compares the computational performance of a single KSR processor to other superscalar processors and compares KSR's UNIX performance to other UNIX systems. Section 4 measures the parallel performance of the KSR multiprocessor and compares it to other shared-memory and distributed-memory multiprocessors. Section 5 relates our early experiences in porting various applications to the KSR.

2. Implementation

The KSR ring:0 consists of a 34 slot backplane, populated with 32 processor boards, or *cells*. The remaining two slots are used for ring:1 interconnect boards. Each cell consists of 12 custom CMOS chips. The shared-memory is managed by 4 Cell Interconnect Units (CIU) and 4 Cache Control Units (CCU). 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 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 being either an FPU or IPU instruction. 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 Megabyte daughter board is attached to the back of each processor board. KSR calls the local memory on each processor *cache* and refers to the 256KB data cache as the *sub-cache*.

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 cache then the data is fetched from the on-cell memory. 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 is stalled until the data arrives. The latencies and capacity of each level of the cache hierarchy are listed in Table 2.1 [18]. The hardware cache (sub-cache) is two-way set associative with random replacement and write-back and uses a 64-byte cache line. The memory cache is 16-way set associative with a 128-byte cache line 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
ring 0	126	1GB
ring 1	600	34GB

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

The programmer or compiler can use a non-blocking pre-fetch instruction (up to four may be in progress from each processor) and a post-store instruction to reduce the latency. Synchronization, or locking, is provided by instructions to lock and unlock a 128-byte subpage.

The KSR configuration at ORNL is a 32 cell-system. An Ethernet and Exabyte 8mm tape drive are connected to the I/O port of cell 1. A Multi-channel Disk (MCD) controller is attached to cell 3. The MCD has 5 SCSI controllers, each with two 1-gigabyte drives. These drives are presently mounted as independent UNIX disk partitions. In the future, the drives can be configured as RAID arrays and as one logical volume with the files striped across the drives. Appendix A summarizes the configurations of other machines (BBN TC2000, IBM

RS/6000-530, Intel iPSC/860, and Sequent 80386-Symmetry) used for comparison in the following sections.

For the tests described in the following sections, the KSR software release used was PR1.14. Unless otherwise noted, -O2 optimization was used. Timings were provided by either the UNIX *time* command, or by timer calls within the application. The KSR supports a "global" time-of-day clock with a 10 millisecond resolution and two sub-microsecond timers on each cell. One timer provides user time, and the other is a free-running timer. The timers all run at the same frequency, but the free-running timer is initialized as each cell is started. Each cell is started serially after cell 1, so all of the free-running timers are offset from each other. Thus if a process/thread migrates to another cell, timings reported by the free-running timer cannot be trusted. We used the free-running timer for many of our tests, but we always bound the thread to the cell for the test, preventing the scheduler from moving it.

3. Single Cell Performance

The single processor performance of the KSR functional units was measured with several widely used benchmarks. Floating point performance was measured with the FORTRAN Livermore Loops, SLALOM (version 2) [11], and the 100×100 double-precision LINPACK. As of this writing, KSR FORTRAN codes performed somewhat faster than the equivalent C programs. As a rough measure of integer performance the C Dhrystone (version 1) was used. Figure 3.1 shows the results of these benchmarks. For comparison, results from the Intel i860 and IBM RS/6000-530 processor are displayed as well (see Appendix A for configurations and compiler options). The 20 MHz KSR is competitive with the faster clocked i860 and 530. The KSR compiles were done with -O2 optimization, except "auto-inline" was used for LINPACK. Unfortunately, with "auto-inline" the LINPACK compile takes more than an hour. Without "auto-inline", the compile still takes several minutes and performance slows from 15 Mflops to 11 Mflops.

The KSR compilers have not yet been optimized for compile-time speed. The KSR takes over 6 minutes to compile the 3000-line Livermore Loops FORTRAN code with -O2 optimization. Compile times for the i860 (a Sun 4/390 cross-compiler) and the IBM RS/6000-530 are under one minute. A similar disparity in performance is exhibited by the BYTE benchmark suite, a set of C programs and shell scripts that exercise various UNIX features including multiple processes, pipes, and compiles. The time for a BYTE run on the KSR was more than five minutes, compared with under one minute for the IBM 530. (The BBN TC2000

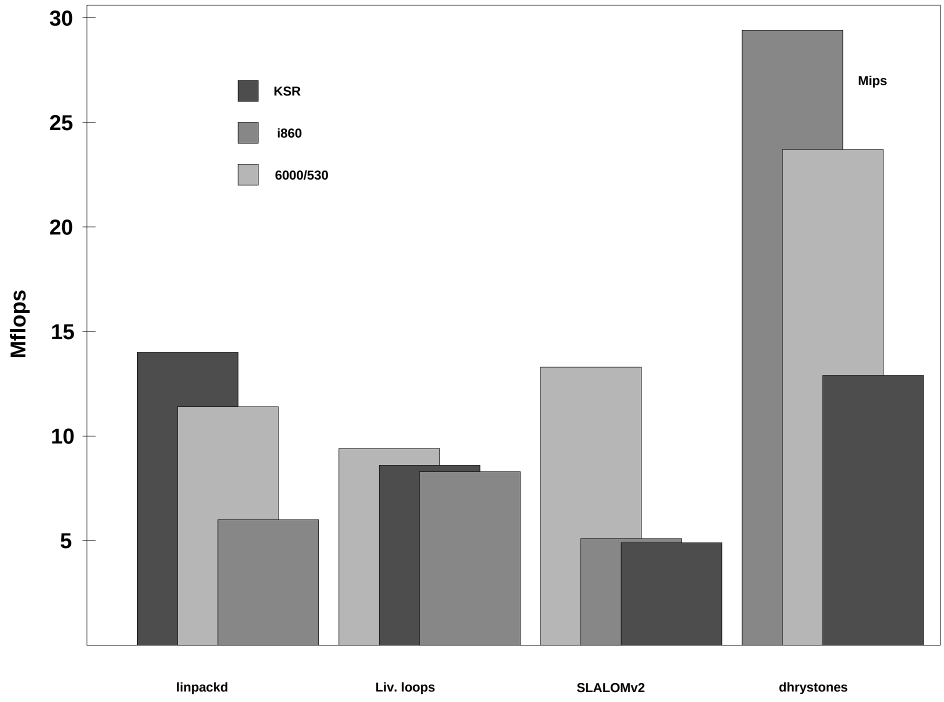

Figure 3.1: *Single processor performance.*

ran the BYTE suite in 113 seconds, the Sequent Symmetry in 117 seconds.) Some of the slowness can be attributed to the development stage of the OS and I/O subsystem. The disk subsystem will eventually support a RAID organization with striping, but at present each disk is a separate UNIX partition.

Basic I/O data rates from the disk subsystem measured with a file system exerciser (FSX) and simple write/read tests are competitive with data rates from the IBM 530. There was some measurable performance difference if the I/O test was performed on the cell attached to the disk subsystem. Write times dropped from nearly 1 Megabtye/second on the I/O cell to 0.31 MBs on other cells. Read times were about 1 MBs and showed little variation from cell to cell, presumably due to disk buffer caching. (A 16 Megabyte file was written/read using 16 KB blocks.) Concurrent I/O tests, multiple processes writing/reading independent files on separate disks, showed promising results with a 2.4 MBs aggregate read rate on four cells — results competitive with concurrent I/O rates on the Intel hypercube file system (CFS) [7]. More extensive I/O tests will be performed when the disk system is more optimally configured.

The performance of a single KSR processor in executing some simple process control primitives is given in Table 3.1. The average time for the creation, execution, and termination of an empty sub-process using *fork()/wait()* can be contrasted with the light weight thread initiate/join. For comparison, the equivalent tests were performed on the BBN TC2000 and Sequent Symmetry. (One could argue that a simple *send/recv* on a distributed-memory system is equivalent to the thread initiate/join. So for further comparison, the time for a *send/recv* on the iPSC/860 is about 160 μs and about 125 μs on the Intel DELTA [8].) The table also shows the time for a single processor to do a lock/unlock. The KSR time is based on using the *gsp* instruction, using the more general mutex library call results in an average performance of 12.1 μs. Performance of concurrent lock/unlock tests are described in the next section.

Process Control Primitives (μs)			
	KSR	BBN	Symmetry
fork/wait	108,000	44,000	14,000
thread/join	100	79	26
lock/unlock	3	8	10

Table 3.1: *Time for process control primitives on a single processor.*

The computational performance of the KSR depends on the effectiveness of the user's program in utilizing the memory hierarchy. The large number of registers and dual instruction streams permit the compiler to generate code to do computations in one instruction stream while loading and storing data in the other. The large register set makes it feasible to unroll loops to a greater depth. A hand-unrolled FORTRAN double-precision (64-bit) matrix multiply achieved 33.3 Mflops.

Data for the registers are fetched from a 256 KB data cache (sub-cache). This large cache sustains high performance over larger vector sizes. Figure 3.2 illustrates the performance of a repeated double-precision complex *zaxpy* vector computation for various vector sizes. The *zaxpy* is repeated 10,000 times on the same two vectors for various vector sizes. Although this test is not representative of any application, it does serve to illustrate cache behvior. When the cache can no longer contain all of the data, performance drops as data has to be fetched from the slower main memory. The advantage of the larger cache is evident when compared with the smaller caches of the i860 (8 KB data cache) and 530 (64 KB data cache). The 256 KB cache actually will hold all of the data for the 100×100 LINPACK. Performance for a 128×128 matrix drops to 5 Megaflops for the unmodified FORTRAN code. However, by using a blocked algorithm as KSR has done for the 1000×1000 LINPACK, performance reaches 31 Megaflops

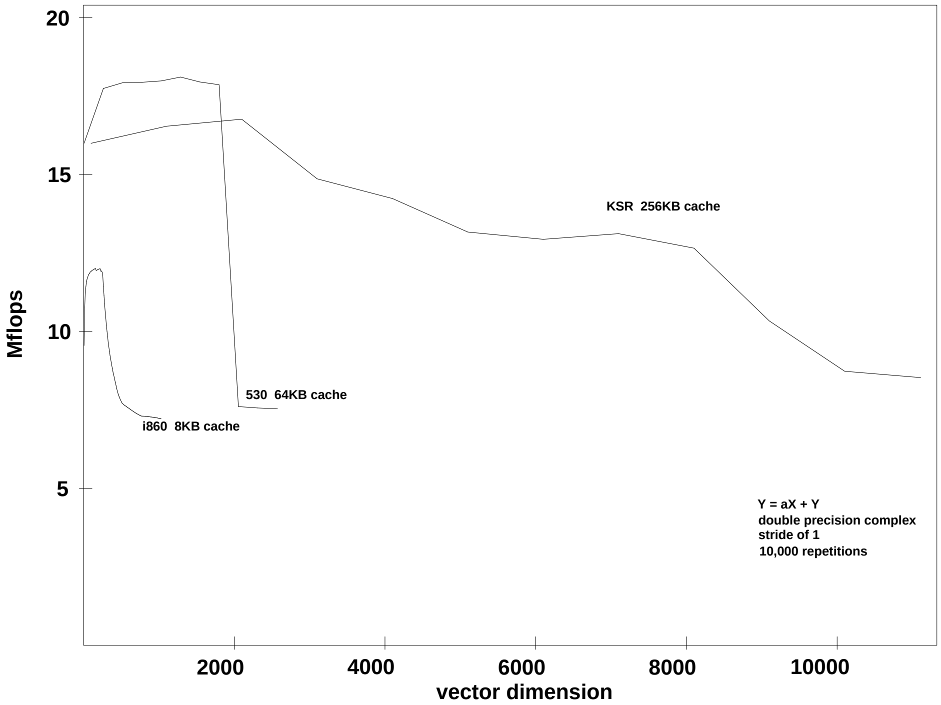

Figure 3.2: *Cache capacity and performance for repeated zaxpy.*

on a single processor. (Additional results for the 1000×1000 LINPACK are presented in the next section.)

If the KSR processor fails to find a data item in the local memory, it must issue a request to the ring to fetch the data from one of the other processors. In the absence of other activity on the ring, we measured this latency to average about 6.7 microseconds (μs). For the BBN TC2000, a remote access takes less than 2 μs, but on the BBN the remote access is not cached to the requesting processor. By contrast, on the KSR, subsequent references will be local (in the absence of other exclusive requests for that location from other processors). A remote access on the iPSC/860 or DELTA would require a *send/recv* and would take roughly 150 μs. Faulting a large vector from one KSR cell to another, using a 128-byte stride, resulted in a data rate of 19.5 MBs. Using the prefetch instruction (up to four may be in progress at once), the measured data rate increases to 34 MBs. By comparison, the peak data rate for iPSC/860 is 2.8 MBs, and the measured peak for the DELTA is about 17 MBs [8]. In the following section we run these memory tests concurrently on multiple processors and measure both single processor and aggregate data rates.

4. Parallel Performance

To measure the parallel performance of the KSR system, we ran a number of the tests in the previous section concurrently on multiple processors. In addition, we measured parallel performance of the memory system under various loads. Parallel tests of various synchronization primitives were conducted as well. The parallel tests were conducted using the *pthread* library and "binding" each thread to a separate processor.

Concurrent Memory Tests

The prefetch test was run concurrently on independent pairs of processors. There was little or no interference among the pairs, each pair averaged about 30 MBs. The aggregate memory throughput increased linearly to 490 MBs for 16 pairs (Figure 4.1). The data rate decreases to about 23 MBs if each processor both fetches and supplies data concurrently. That is, cell i is prefetching data in from cell $i+1$ while cell $i-1$ is faulting in data from cell i. The data rate for this test increases linearly to 731 MBs for 32 cells (Figure 4.1). The linear response and aggregate data rate are quite good, but these tests were not able to achieve the vendor-stated peak of 1 GBs.

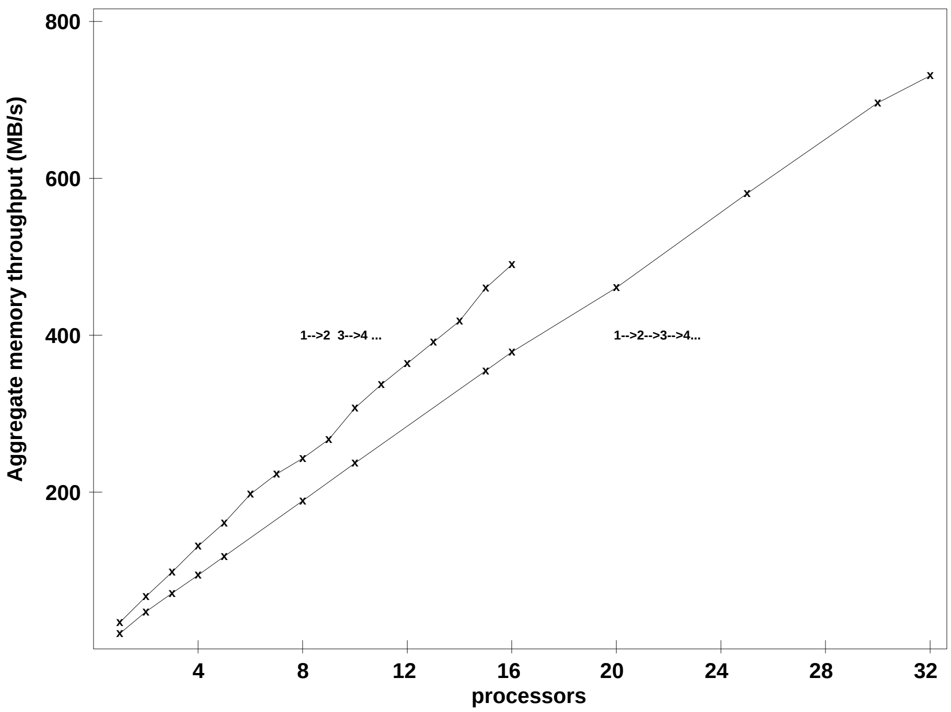

Figure 4.1: *Aggregate memory throughput for concurrent prefetch.*

To stress the memory subsystem, we measured the average time for doing an unrestricted update of a shared variable with varying number of processors. The unrestricted update is unrealistic, since in a real application such an update would be coordinated with a lock. However, the test is adequate for our intent of measuring the response of the memory subsystem to a very hot spot. For comparison, the same test was performed on the BBN and Sequent systems. For all three machines, the compute time is comparable and increases linearly with the number of processors (Figure 4.2). Though the Sequent has a slower CPU, its

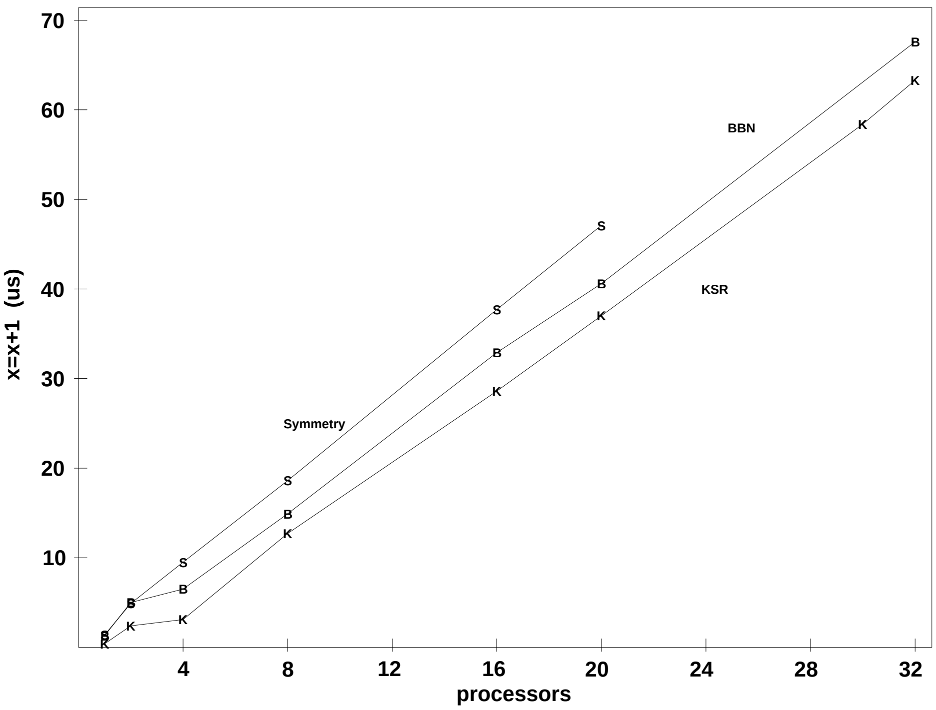

Figure 4.2: *Average time for* $x = x + 1$.

memory latency is better than either the BBN or KSR, so the compute times for this test are comparable. For both the BBN and KSR, the memory subsystem does not reach saturation until more than four processors are contending for the shared location. For further comparison, we conducted the hot-spot test on the distributed memory iPSC/860 and DELTA. Multiple processors send a message to the owning processor requesting the current value, followed by a message updating the value. For 32 processors, the average update time was 5.7 ms for iPSC/860 and 5.9 ms for the DELTA compared to 63 μs for the KSR.

To further study the effects of a hot spot in a shared memory, we used the workload generator described in [16]. An input file to the generator describes the various workload characteristics for exercising a shared-memory system. One can specify the number of shared locations, the percentage of shared references to local references, and whether locking is required. We ran the workload using a single shared memory location and no locking for various percentages of shared-to-local references. The occurrence of the shared reference within the workload can be deterministic or probabilistic [16]. The tests were run on the KSR, BBN, and Sequent systems. Figure 4.3 shows the efficiency of each system for a 1% and 10% shared access ratio using the probabilistic model. Efficiency is measured as the average time for executing the workload on a single processor (the "shared" location is local in this case) divided by the average time for executing the workload concurrently on p processors, T_1/T_p.

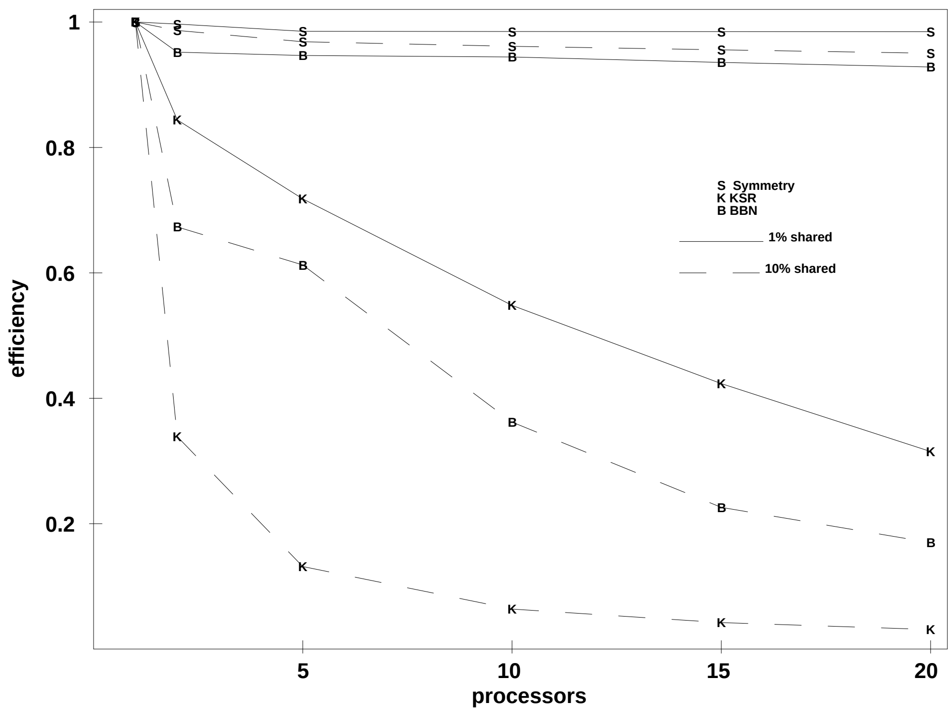

Figure 4.3: *Memory efficiency (T_1/T_p) for referencing a shared location as 1% and 10% of the workload.*

Although the three systems performed comparably when the memory subsystems were saturated (Figure 4.2), their behavior under lighter loads is markedly different. The Sequent shared-bus can easily keep up with the demand from the workloads. The efficiency for the KSR falls off faster than for the BBN, but response of the memory subsystems (shape of the curve) are roughly the same. The KSR has a faster processor and longer remote memory latency than the BBN which accounts for most of the performance difference. Figure 4.4 are the same workloads with performance measured as the average time to complete a

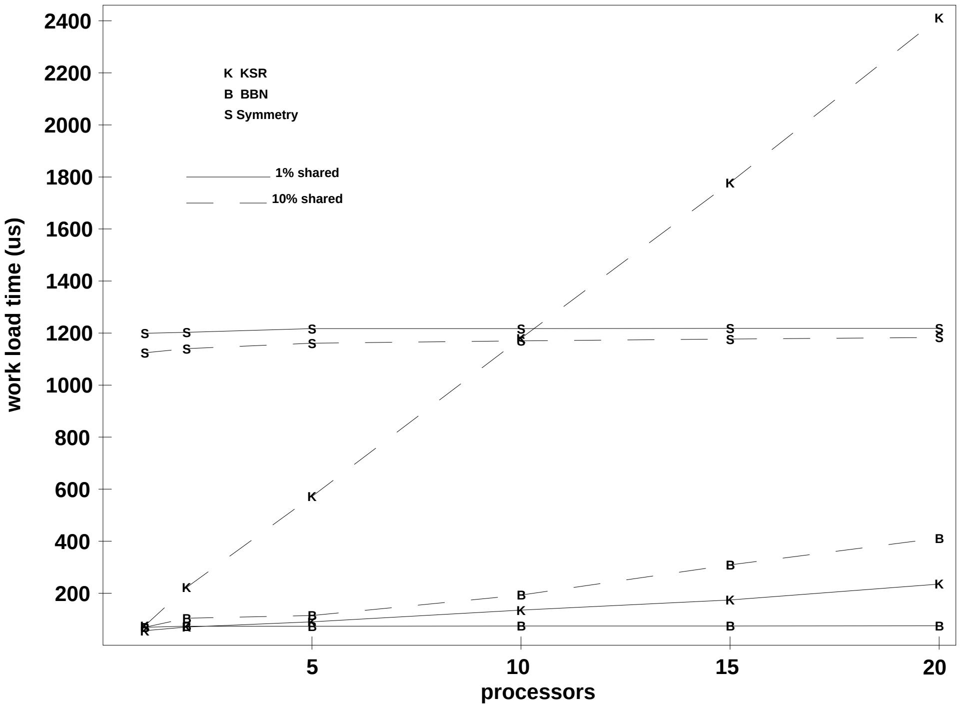

Figure 4.4: *Average workload time for referencing a shared location as 1% and 10% of the workload.*

workload. The KSR is noticeably slowed in relation to the other two systems, suggesting the need for coarser-grained applications for the KSR shared-memory system.

Locks and barriers

Access to a shared location is usually controlled by an atomic locking operation. A synthetic lock/unlock test was run on the three shared-memory systems to measure the performance of locking operations on a single lock (Figure 4.5). The performance of the KSR hardware lock instruction, *gsp* (blocking version), is better than the *mutex* library routine for a few processors, but *gsp* performance degrades rapidly for more than 15 processors. The *mutex* version is thus preferred and performs well compared to the BBN and Sequent.

The lock controls data access, the barrier controls synchronization of processes or threads. Figure 4.6 compares the barrier times for both shared-memory and distributed-memory systems for varying number of processors. If the barrier is implemented with a single lock, then performance degrades linearly with the number of processors. The BBN, Sequent, and the KSR (the solid line in Figure 4.6) use a single-lock barrier. The hypercube and mesh use a spanning

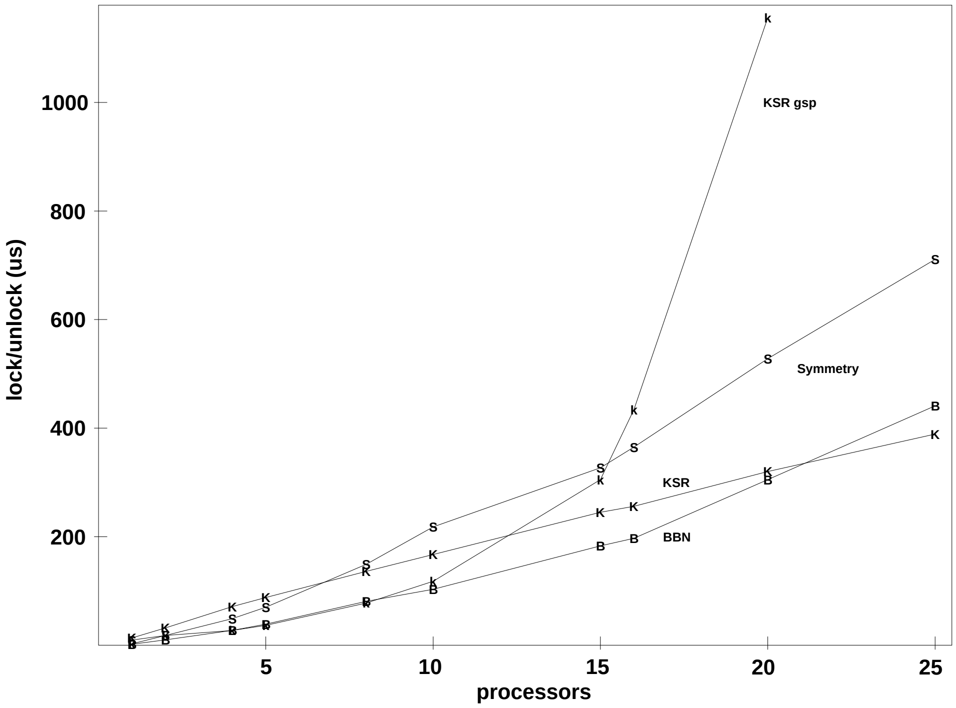

Figure 4.5: *Average lock/unlock time for a single lock.*

tree to implement the barrier, so performance goes as the log_2 of the number of processors. The KSR barrier function also provides an option for a spanning-tree like implementation. The dashed line in Figure 4.6 shows the improved KSR performance using a tree of width four. (Presumably a similar implementation for the BBN would improve its barrier performance as well.) The bus-based Sequent shared-memory system provides the best performance, but the architecture is not extensible beyond 30 processors. Memory (or message-passing) latency, bandwidth, and contention account for most of the difference in barrier performance for the different machines. Since we are using wall-clock time, the barrier times may also be affected by the OS overhead on one or more processors on each system. OS timer interrupts typically occur every 10 ms. The timer-interrupt overhead on the Intel nodes is only about 50 μs, but for the UNIX-based systems (KSR, BBN, and Sequent) the overhead is on the order of 500 μs.

Parallel applications

The next class of benchmarks we used in comparing the KSR with other architectures consisted of small C applications that utilize shared memory, threads, barriers, and locks. The applications do simple numeric integration using spatial decomposition (static allocation), matrix multiply using spatial decomposition

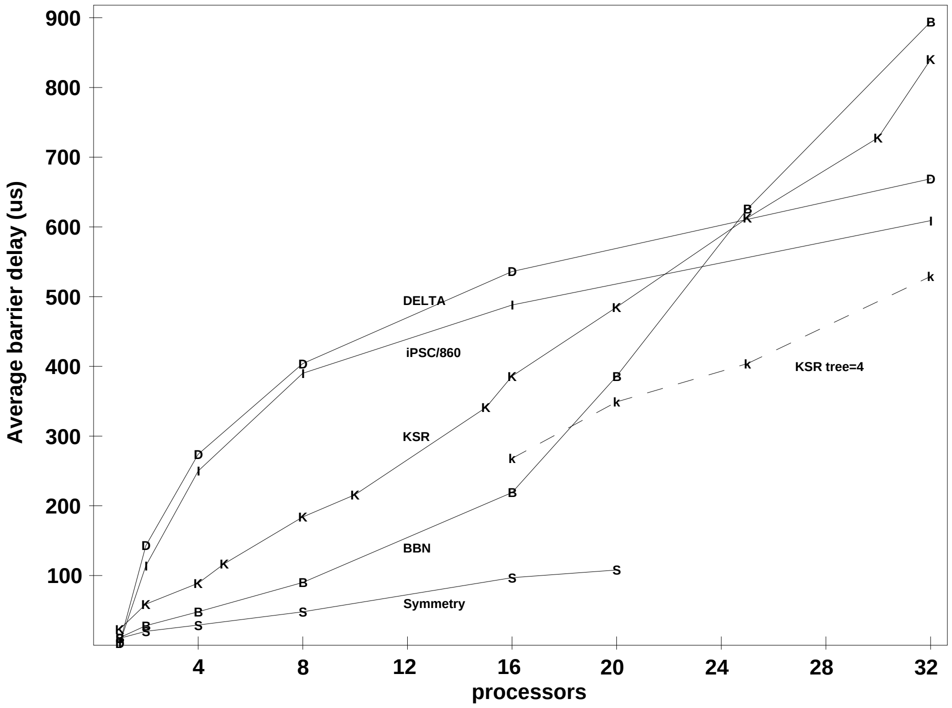

Figure 4.6: *Average time for a barrier synchronization.*

(static allocation), finite difference using chaotic Jacobi iterative method with static spatial decomposition, a parallel quick sort using a queue-of-tasks model (dynamic allocation), and solve a linear system using Cholesky factorization (dynamic allocation). The codes use explicit parallelization and were easily ported to the KSR from the Sequent version. The main objective was to compare the shared-memory architectures running identical source programs (except for the translation of the calls that manage the parallelism).

Figure 4.7 illustrates the Cholesky performance for the shared-memory multiprocessors and for the Intel distributed-memory multiprocessors. The shared-memory code could not be run on the Intel multiprocessors, so the Intel performance includes the effects of a different algorithm — the program must explicitly move portions of the matrix among the various processors. The performance of the serial code is represented as processor 0 in the figure. The BBN outperformed the KSR in the parallel (and serial) quick sort and numerical integration. The quicksort is integer work and the BBN also performed the dhrystones faster than the KSR (Appendix A). The numerical integration is dominated by floating-point divides which the KSR does in software and the BBN does in hardware.

The performance of these tests was consistent with the underlying speed of the individual processors and memory subsystem. In general, the Sequent was slower

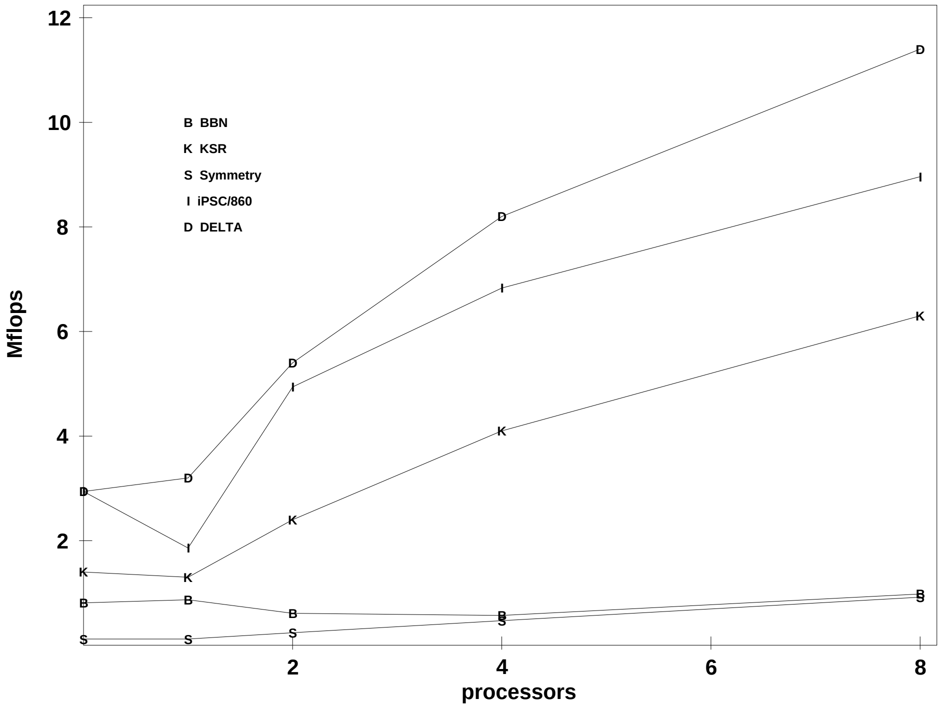

Figure 4.7: *Megaflops for* 400×400 *double-precision C Cholesky.*

in absolute time, but maintained a higher efficiency (speedup divided by the number of processors) with increasing processors. The Sequent's low latency bus architecture accounts for the high efficiency, but the architecture is not extensible to more than 30 processors. The KSR was faster than the BBN in most tests and maintained a higher efficiency. Even though the BBN memory hierarchy has a lower latency than the KSR ring, the KSR's ability to "fault" a remote reference into a local reference results in higher performance for these tests. (A BBN-tuned application would see that work was assigned to a processor that "owned" the distributed portions of the global data structures — such tuning is not required for the KSR, though it too can profit from such tuning.)

To compare the architectures with a larger problem, optimized for each architecture, we used the 1000×1000 double precision LINPACK [3]. The KSR implementation was based on a block algorithm implemented by KSR's Nick Camp in FORTRAN with some assembly language. The matrix is manipulated in groups of columns to optimize the use of the 256KB cache, and post-store's are used to reduce ring latency. Figure 4.8 shows the results over 32 processors along with results from the iPSC/860 and DELTA. (The Intel implementation is also based on a block algorithm implemented by Robert van de Geijn.) A different algorithm is used for 4 or fewer KSR processors, which explains the first

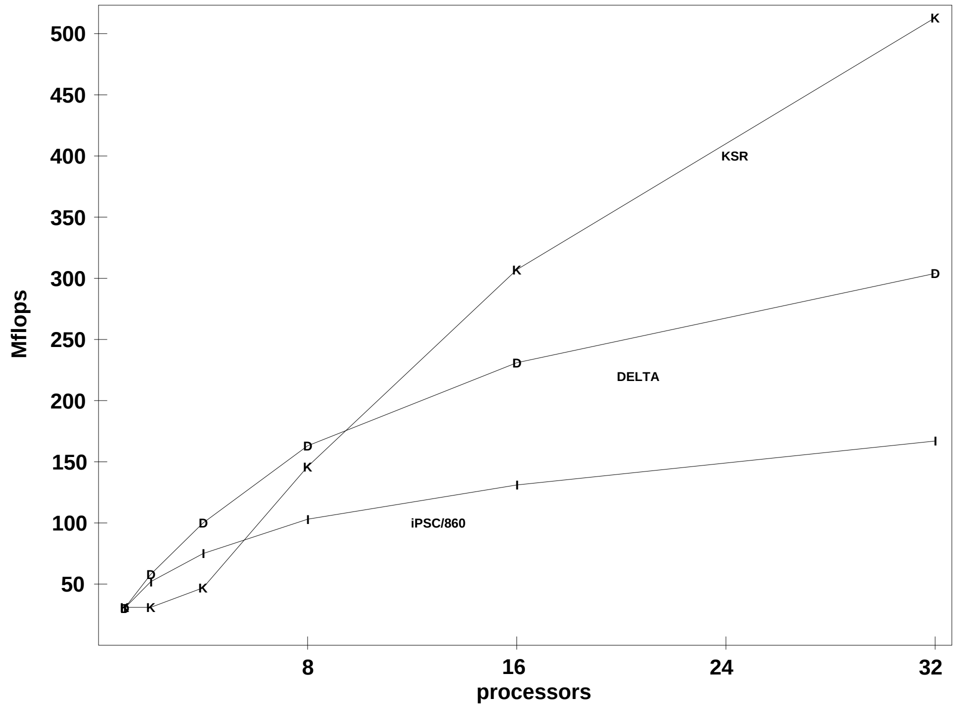

Figure 4.8: *Megaflops for* 1000 × 1000 *double-precision LINPACK.*

few data points of the KSR performance curve. The two Intel machines share the same processor and roughly the same message latency, thus the difference in their performance is due to the higher bandwidth of the DELTA mesh. The KSR outperforms the Intel multiprocessors because it has both higher bandwidth and lower latency. (Performance figures are not available for the BBN and Symmetry, but since their single-processor performance is low, their parallel performance would not be competitive for this test.)

5. Early Experiences

As the various benchmark kernels were being developed and tested other users were working on porting applications to the KSR. The KSR multiprocessor is designed to make porting applications easy and that has been our initial experience, both for serial and parallel codes. The first parallel application to be ported was a 19,000 line FORTRAN code that calculates energy densities for high temperature superconducting materials [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 243 Mflops on

32 processors.

Serial and parallel versions of a sparse-matrix library (SPARSPAK, [17]) and a large FORTRAN global climate modeling code are also being ported to the KSR. Each of these large FORTRAN applications has usually uncovered one or more bugs in the -O2 optimization of the FORTRAN compiler. These bugs were usually fixed quickly. SPARSPAK includes implicit parallel directives for the Cray and Sequent, and those directives map nicely into corresponding KSR directives. The climate modeling code also has Cray parallel directives.

A number of UNIX C codes were ported as well, including the Network Time Protocol (NTP) [15], a variety of hypercube simulators [5], and PVM [10]. Some of the C codes had to be modified to account for 64-bit *long*'s. The hypercube simulators use *fork()* to create sub-tasks and then use pipes, sockets, or System V shared memory to communicate among the sub-tasks. Performance for these simulators was poor, since the scheduler presently runs only one sub-task at a time.

Hardware reliability has been very good, with only two board failures during the first four months. The compilers and operating system have improved with each release, and KSR support has been very responsive. The OS still lacks several features for full multi-user support, but those features will be available in the first production release of the OS.

We will continue tracking KSR performance with the new releases and hope to expand the system to include a second ring. A second ring would permit us to better understand the extensibility of the architecture. We would like to develop analytical models of the performance of the memory hierarchy in terms of latency, hit ratio, and contention. A hardware memory event monitor will be installed on each cell in early summer. Data from the event monitor will permit us to better measure architecture and application performance. Finally, the user community will be expanded, providing more applications and a better understanding of the ease of use of the KSR multiprocessor.

Acknowledgements

A special thanks to the Advanced Computing Research Facility at Argonne National Laboratory for providing access to the BBN TC2000 and Sequent Symmetry. 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. Communication performance of the Intel Touchstone DELTA mesh. Technical report, Oak Ridge National Laboratory, 1992. ORNL/TM-11983.

[9] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. A Users' Guide to PICL A Portable Instrumented Communication Library. Technical report, Oak Ridge National Laboratory, October 1990. ORNL/TM-11616.

[10] G. A. Geist and V. S. Sunderam. Network Based Concurrent Computing on the PVM System. Technical report, Oak Ridge National Laboratory, June 1991. ORNL/TM-11760.

[11] John Gustafson, Diane Rover, Stephen Elbert, and Michael Carter. The design of a scalable, fixed-time computer benchmark. Technical report, Ames Laboratory, 1990.

[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] D. L. Mills. Network time protocol (version 2) specification and implementation. Technical report, DARPA Network Working Group, September 1989. RFC-1119.

[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] 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.

A. Comparative Architectures

The KSR is compared with a number of other processors. This appendix summarizes the architectures and configurations used in this report. 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 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 [19]. 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.

IBM RS/6000-530

The IBM RS/6000-530 uses a 25 MHz processor with a 64 KB data cache and a 400 MBs memory bandwidth. The processor has an independent integer and floating point unit, and the floating point unit has an independent adder and multiplier. The peak performance is thus 50 Mflops. The workstation used in the tests was running AIX 3.1 in 16 MB of memory. Compiles used -O optimizations. LINPACK 100×100 double-precision was 11 Mflops [3]. Dhrystone (v1.0) was 23.7 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 Cal Tech 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 100×100 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 100×100 double-precision was 0.37 Mflops [3]. Dhrystone (v1.0) was 3.6 Mips.

System Metrics						
	KSR	BBN	Seq	530	i860	DELTA
clock rate (MHz)	20	20	16	25	40	40
data cache (KB)	256	16	64	64	8	8
memory size (MB/CPU)	32	16	32(T)	16	8	16
memory bandwidth (MBs)	160	?	?	400	160	160
remote mem. bandwith (MBs)	34	40	54	n.a.	2.8	17
remote mem. latency (μs)	6.7	2	<1	n.a.	150	150
peak mflops (64-bit)	40	?	?	50	64	64
linpack mflops (100×100)	15	1	0.4	11	6.5	6.5
dhrystone mips (v1.0)	12.9	19.4	3.6	23.7	29.4	29.4
max processors	1088	512	30	n.a.	128	512

Table A.1: *System metrics for systems used in this report.*