



3 4456 0374303 7

ORNL/TM-12287

oml

**OAK RIDGE
NATIONAL
LABORATORY**

MARTIN MARIETTA

The KSR1: Experimentation and Modeling of Poststore

E. Rosti
E. Smirni
T. D. Wagner
A. W. Apon
L. W. Dowdy

OAK RIDGE NATIONAL LABORATORY
CENTRAL RESEARCH LIBRARY
CIRCULATION SECTION
4009N ROOM 125
LIBRARY LOAN COPY
DO NOT TRANSFER TO ANOTHER PERSON
If you wish someone else to see this
report, send in name with report and
the library will arrange a loan.
UCR 798 (0 577)

MANAGED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

This report has been prepared for the use of the Government.

Available to DOD and DOD contractors under the provisions of the Freedom of Information Act for a fee. Call (703) 605-4800, ext. 8430; 576-8401, ATG 828 8411.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 528A Patuxent Ferry, Maryland 20740.

This report was prepared as part of the work performed by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of its employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not constitute an endorsement, recommendation, or approval by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Engineering Physics and Mathematics Division
Mathematical Sciences Section

405
32

THE KSR1: EXPERIMENTATION AND MODELING OF POSTSTORE

E. Rosti •
E. Smirni †
T. D. Wagner †
A. W. Apon †
L. W. Dowdy †

• Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Via Comelico 39
20135 Milano, Italy

† Computer Science Department
Vanderbilt University
Box 1679, Station B
Nashville, TN 37235

Date Published: February 1993

This work was partially supported by sub-contract 19X-SL131V from the Oak Ridge National Laboratory, and by grant N. 92.01615.PF69 from the Italian CNR "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo - Sottoprogetto 3."

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

| | | |
|-----|--|----|
| 1 | Introduction | 1 |
| 2 | Architectural Overview of the KSR1 | 2 |
| 2.1 | System Hardware | 2 |
| 2.2 | Memory Organization | 4 |
| 2.3 | System Configuration | 6 |
| 3 | Experimental Analysis | 6 |
| 3.1 | The Workload | 6 |
| 3.2 | The Experiments | 7 |
| 3.3 | The Results | 9 |
| 4 | Modeling and Validation | 14 |
| 4.1 | Detailed Model | 14 |
| 4.2 | Approximate Load Dependent Model | 17 |
| 4.3 | Theoretical Model/Experimental Comparisons | 19 |
| 5 | Generalizations | 21 |
| 6 | Summary | 22 |
| 7 | References | 25 |

THE KSR1: EXPERIMENTATION AND MODELING OF POSTSTORE

E. Rosti
E. Smirni
T. D. Wagner
A. W. Apon
L. W. Dowdy

Abstract

Kendall Square Research introduced the KSR1 system in 1991. The architecture is based on a ring of rings of 64-bit microprocessors. It is a distributed, shared memory system and is scalable. The memory structure is unique and is the key to understanding the system. Different levels of caching eliminates physical memory addressing and leads to the ALLCACHETM scheme. Since requested data may be found in any of several caches, the initial access time is variable. Once pulled into the local (sub)cache, subsequent access times are fixed and minimal. Thus, the KSR1 is a Cache-Only Memory Architecture (COMA) system.

This paper describes experimentation and an analytic model of the KSR1. The focus is on the poststore programmer option. With the poststore option, the programmer can elect to broadcast the updated value of a variable to all processors that might have a copy. This may save time for threads on other processors, but delays the broadcasting thread and places additional traffic on the ring. The specific issue addressed is to determine under what conditions poststore is beneficial. The analytic model and the experimental observations are in good agreement. They indicate that the decision to use poststore depends both on the application and the current system load.

1. Introduction

Traditionally, the scalability of shared memory multiprocessors has been limited due to memory access path contention. However, the KSR1 system, recently developed by Kendall Square Research, demonstrates that scalable shared memory multiprocessors are feasible. From a measurement and modeling perspective, the KSR1 and its architectural paradigm deserve an in-depth analysis.

One novel feature of the KSR1 is its memory management scheme, ALLCACHETM. Each processor has its own local memory that is managed as a cache, and a valid copy of a data item must exist in the local cache of the processor in order to be accessed. Data items are not bound to any particular memory, but migrate dynamically to a processor when they are accessed. The entire memory is shared and the memory is viewed as a hierarchy of caches. Upon writing, a requesting processor writes the data item to its local cache and marks it as valid. All other copies of the item in other processor caches are marked as invalid. Prior to reading, a requesting processor must have a valid copy of the item in its local cache. If a valid copy of the item is not in the local cache of the requesting processor, then a valid copy is migrated from the local cache of another processor. Depending on which cache contains the requested data item at any particular time, the time required to perform this migration may vary. However, once a valid copy of the requested item is moved into the local cache, all subsequent accesses are to the local copy. Thus, the KSR1 has a Cache-Only Memory Architecture (COMA) [5].

To take advantage of the architecture, programmers are provided with a *poststore* option. When a variable is updated by a write, using *poststore* will cause a valid copy of the variable to be sent to all caches which contain a copy of that variable. This will shorten the access time for any future reads on those other processors, since each will have a valid copy of the item in its local cache. Without *poststore*, whenever a future reader requests the variable, it must first pull a valid copy into its cache. Clearly, a tradeoff exists since using *poststore* will shorten the time for future reads, but lengthens the time for the write.

This paper presents an experimental and modeling study of the KSR1. The focus is on the *poststore* option. The stated goals and outline of this work are:

- to understand and describe the KSR1 architecture,
- to run controlled experiments on the KSR1, using a simple readers-and-writers workload, to observe performance with and without *poststore*,
- to construct and validate an analytic model of the system which could be used for predicting the general behavior of the KSR1 and for predicting the specific behavior of *poststore*, and
- to outline generalizations and summarize our findings.

The purpose of this paper is to study the effects of poststore for a particular reader/writer workload. The results show that relatively simple models accurately indicate the effects of poststore. Also, results show that poststore is more effective as the number of reader threads in one application increases, but becomes less effective as the total number of applications increases. Therefore, the effective use of poststore depends on both the programmer's application code as well as the system load.

2. Architectural Overview of the KSR1

2.1. System Hardware

The general KSR architecture is a multiprocessor system composed of a hierarchy of rings. The lowest level, ring:0, consists of a 34 slot backplane connecting 32 processing cells (processing elements) and two cells responsible for routing to the next higher layer ring, ring:1. A fully populated ring:1 is composed of the interconnecting cells from 32 ring:0 rings. A fully configured KSR1 is composed of two layers containing 1024 processing cells along with two ring interconnecting cells on each ring:0. The general KSR architecture provides for a third layer which connects 32 ring:1 rings into a ring:2 layer. Figure 1 shows the hierarchical ring structure of the KSR multiprocessor.

This study deals with a KSR1 multiprocessor with a single ring:0 installed. The description that follows is of the general KSR architecture with specific attention given to the memory structure and management of a single ring:0.

Each processing cell is constructed from 12 custom CMOS chips:

- The Co-Execution Unit (CEU) fetches all instructions, controls data fetch and store, controls instruction flow, and does arithmetic required for address calculations.
- The Integer Processing Unit (IPU) executes integer arithmetic and logical instructions.
- The Floating Point Unit (FPU) executes floating point instructions.
- The eXternal Input/output Unit (XIU) performs DMA and programmed I/O.
- Four Cache Control Units (CCU) are the interface between the 0.5MB subcache and the 32MB local memory (referred to as the local cache).
- Four Cell Interconnect Units (CIU) are the interface between a processing cell and the ring:0 ring.

In one instruction cycle an instruction pair is executed. One member of the pair is an instruction for the CEU or XIU and the other member is an instruction for the FPU or IPU. The clock

speed is 20 MHz. As in other superscalar processors, the KSR processor operates in a pipelined fashion with two pipelines, one for the FPU/IPU and one for the CEU/XIU. The pipelining and 20 MHz clock yield a peak 40 MFLOPS for each cell. Using shared data structures and optimized code, early implementations of a 1000 X 1000 double precision LINPACK running on a 32 processor system resulted in over 500 MFLOPS total capacity [3].

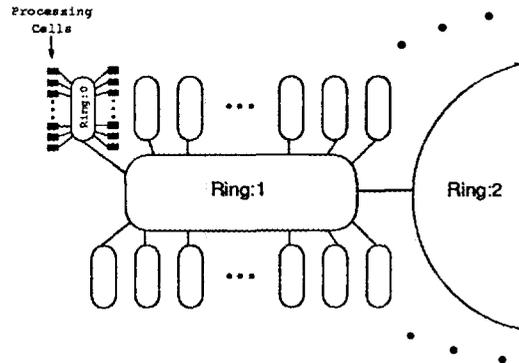


Figure 1: KSR hierarchy of rings.

Each processing cell also contains a 256KB data cache and a 256KB instruction cache. The on-board data and instruction caches are referred to as *subcaches*. A daughter board connected to each processing cell contains 32MB of memory referred to as *local cache*. The word size of the KSR is 64 bits and all functional units are based on 64 bit operands. All execute and control operations are register oriented. Each processor has 64 floating point registers, 32 integer registers, and 32 addressing registers. All registers are 64 bits wide. (The KSR1 implementation uses 40 bit addressing registers.)

In addition to the 32 processing cells, each ring:0 also contains 2 ALLCACHE Routing and Directory (ARD) cells. One of the ARD cells is an uplink from the ring:0 to ring:1. The other ARD is a downlink from the ring:1 to ring:0. The ARDs participate in the transfer of data between ring:0s across ring:1.

All of the local caches, together with the interconnecting rings, make up the ALLCACHE memory system. Addressing in the KSR architecture is based on the translation of a Context Address (CA) into a System Virtual Address (SVA). Context addresses are composed of a segment and offset and are translated into System Virtual Addresses via fully associative hardware Segment Translation Tables (STTs) on each processor. There are two STTs, one for data and one for instructions. The System Virtual Address space consists of all of the local caches. The ALLCACHE memory system and the organization and management of System Virtual Address (SVA) space is the major difference between the KSR architecture and other architectures. When a processor references an SVA, a *search engine*, which is the collection of CIUs and the ARD on each ring:0 along with the ring interface, locates the SVA and moves its

contents to the local cache of the referencing processor.

2.2. Memory Organization

ALLCACHE stores data in units of pages and subpages. Pages contain 16K bytes divided into 128 subpages of 128 bytes each. Each local cache can hold 2,048 pages. The memory system allocates storage in the local caches on the basis of pages and each page of SVA space is either entirely allocated in the caches or not allocated at all. The local caches share data in units of subpages. Whenever a page of SVA space is allocated in the system, there may be more than one copy present. This would be the case when several threads running on different processors are all referencing shared memory. It is possible that each local cache that has allocated a page may not contain a copy of all of the subpages in that page. That is, space in the local caches is allocated on a page basis, but data is transferred on a subpage basis. Each local cache maintains a cache directory in a 16-way set associative memory with 128 sets that maps physical pages in that cache to SVA pages. All of the pages of SVA space are divided into 128 equivalence classes, each associated with a cache directory set. Since there are 16 elements in each set in the cache directory, a cache can contain no more than 16 pages in the same equivalence class.

The subcaches are allocated on the basis of blocks (2K bytes) and data is moved into and out of the subcaches in subblocks of 64 bytes each. A two way set associative subcache directory maintains the mapping between subcache blocks and SVA pages with one descriptor for each block. The subcaches replace blocks as needed using a random replacement scheme.

In the cache directory of each cell, additional information is maintained that represents the *state* of each subpage in the local cache. There are four states that a subpage can be in:

- **Exclusive owner:** Indicates that this is the only valid copy of the subpage in all of the local caches (i.e., in the entire system). The contents can be read or modified.
- **Atomic:** Like exclusive, this is the only valid copy and the subpage can be modified. This state also provides a flag to allow synchronization by multiple processors. Thus, this state provides for locks.
- **Read-Only:** Indicates that there are two or more valid copies of this subpage among all of the local caches. The contents of this subpage cannot be modified until its state is changed to exclusive or atomic.
- **Invalid:** The contents of this subpage are not to be accessed (i.e., read or modified). Newly allocated pages set all subpage descriptors to invalid. This state is analogous to the setting of a “dirty bit.”

The subcaches also maintain state information at the subblock level. The instruction subcache allows each subblock to be in either the invalid state or the read-only state. In addition to

invalid and read-only, the data subcache allows a block to be in the exclusive owner state to allow for modification. The data subcache also maintains modification information for each subblock. The state of a subblock in the subcache is not allowed to be stronger than the state of the corresponding subpage in the local cache. Thus, it is not possible for a subblock's state to be exclusive in the subcache while read-only in the local cache.

When a processor references an SVA address it continues execution for two cycles, which is the latency of the subcache. If the address is not contained in the subcache, the processor is stalled and a request is presented to the CCUs to locate the subpage containing the requested address in the ALLCACHE memory. If the subpage containing the address is not present in the local cache (and in the state requested by the processor), then the CCUs make a request of the local CIUs to format a request message and place it on ring:0. The ring:0 communication interconnect is a slotted pipelined ring with a total bandwidth of 1GB. There are 13 slots on the ring:0 ring. Each message on the ring consists of a 16 byte header followed by one subpage (128 bytes) of data. As a request message passes each processing cell, the cell's CIU determines if the request can be satisfied from its local cache. If it can be satisfied, the request message is extracted from the ring and a response message is inserted. Also attached to each ring:0 is an ALLCACHE Router and Directory (ARD) cell that contains a directory of the entire ring:0 cache (i.e., all of the local caches). If the ARD determines that a request message cannot be satisfied on the local ring:0, it extracts the message and inserts a request on the next higher ring in the hierarchy, ring:1. When the response message to the original request is inserted on the ring, the requesting processor copies the message and fills the original request from the local CCU. If a request message returns to the requesting processor unanswered, a hard page fault is generated and the subpage is brought in from the disk. The latency and total capacity of the ALLCACHE memory system hierarchy is shown in Table 1 [6].

Table 1: Latencies and capacities

| Location of subpage | Total capacity (MB) | Latency in cycles (5ns) |
|---------------------|---------------------|-------------------------|
| Local subcache | 0.5 | 2 |
| Local cache | 32 | 18 |
| Ring:0 | 1,024 | 175 |
| Ring:1 | 34,816 | 600 |
| Disk | | 400,000 |

The hardware management of the KSR memory system assures that the ALLCACHE memory is both *sequentially consistent* [7] and *strongly ordered* [2]. The state of a subpage in local cache or a subblock in subcache is changed in response to requests from processing cells in the system. When a load instruction is issued, it can specify the state that the subblock should

possess. A store instruction always requires that a subpage have an exclusive ownership state. Whenever a request for exclusive ownership is made, all copies of the subpage in other cells are marked as invalid. One distinction between the ALLCACHE memory and NUMA shared memory architectures is that no processor is the designated “home” of a subpage of memory. There can be multiple local caches that have allocated space for a subpage and the ownership travels around the rings as required, to satisfy state requests by the multiple processors.

One problem that floating ownership can cause is that as fetch requests are made, it is possible that the local cache of the processor issuing the request may have an invalid copy. There are two methods by which the inefficiencies created by this approach are moderated. First, whenever a copy of a subpage is sent across the ring to satisfy a request, any local cache that has a descriptor for the subpage (i.e., has allocated space) but does not have a valid copy, can pick up a read-only copy of the subpage if the cell is not too busy. This *automatic prefetching* is a function of the hardware. Second, there are two instructions, `pcsp` (prefetch subpage to cache) and `pstsp` (poststore subpage), that provide the programmer with some control over the locality of specific subpages. The prefetch instruction allows for the specification of the state that should be acquired when a subpage is fetched. The poststore instruction simply relinquishes exclusive ownership and broadcasts the contents of a subpage on the ring. All cells that have a descriptor for the subpage will take a copy from the ring if they are not too busy. If no advance copy is obtained by a cell, then a new request is issued whenever the cell requires a valid copy.

2.3. System Configuration

The KSR operating system is an implementation of OSF-1 and provides a standard UNIX interface. Built on top of the Mach threads of OSF-1 is a pthreads interface based on the IEEE POSIX draft standard, P1003.4a. The KSR pthreads interface includes extensions to enable an application to manage ring traffic and the geometry of thread placement for optimizing the performance of cooperating threads. The experiments described here were run using version R1.0.5 of the KSR OS. The system includes a fully configured ring:0 with 32 processing cells. The timings reported in the experimental section were collected using the two sub-microsecond timers on each cell, one which reports user time, the other system time.

3. Experimental Analysis

3.1. The Workload

In order to study the advantages and disadvantages of using poststore after an update, various workloads consisting of a parallel version of a readers/writers workload are constructed. Each

workload performs the following steps:

- Initialization Phase

1. A number of reader and writer threads are spawned, each bound to a specific processor.
2. Each reader and writer reads a predetermined portion of a given data set. This ensures that a copy of the shared data set is in the local cache of each participating thread, and that no disk accesses will be required during the measurement phase.

- Measurement Phase

1. Timing begins for each writer.
2. Each writer updates its portion of the data set. Writing is done with or without poststore, depending on the experiment.
3. Timing ends for each writer.
4. Timing begins for each reader.
5. Each reader sequentially reads its portion of the data set one time.
6. Timing ends for each reader.

The emphasis of the experiments is to determine under which conditions the use of poststore is an advantage. If the writers broadcast their updates with poststore, then each reader should find a valid copy of the data in its local cache during the reading phase. If the updates are done without poststore, then no valid copy is available in the reader's local cache during the reading phase. In this case, every read is a cache miss and generates a request on the ring. Readers are allowed to read only after all the writers have finished. In all the experiments, readers and writers are implemented by distinct threads, and are mapped onto distinct processing cells, so that no two threads in the same application access the same local cache.

3.2. The Experiments

The parameters to be varied in the experiments are:

1. the amount of data requested per subpage access,
2. the amount of delay between accesses,
3. the read access pattern,
4. the number of writers,
5. the amount of data set sharing among readers, and

6. the number of concurrent reader/writer workloads.

Several experiments were run using different values for each of these parameters. Table 2 lists the experiments reported here with their parameter values. Three data set sizes were used: small (13K subpages), medium (52K subpages), and large (100K subpages). Different sizes test the effect of processing for longer periods of time. Each experiment was run for a varying number of readers.

Table 2: Experiment parameter values

| Experiment | Granularity | Delay | Access Pattern | Writer | Sharing | Workloads |
|------------|----------------|-------|----------------|----------|---------|-----------|
| A | 1 per subpage | N | same | single | global | single |
| B | entire subpage | N | same | single | global | single |
| C | 1 per subpage | Y | same | single | global | single |
| D | entire subpage | N | different | single | global | single |
| E | 1 per subblock | N | same | multiple | global | single |
| F | 1 per subblock | N | same | multiple | private | single |
| G | 1 per subpage | Y | same | single | global | multiple |

Different access granularity levels affect the rate at which read requests are made to the ring. The access granularity may be one access per subpage, one access per subblock (i.e., two accesses per subpage), or the entire subpage. In the experiments reported, each read is a 64 bit word. Each subpage contains 16 words. When one word per subpage is read, without intervening processing, the rate at which requests for invalid pages are made is maximized. When one word per subblock is read, then the rate of ring requests decreases, since every other read is a local cache hit. When an entire subpage is read there will be one request to the ring (to acquire the subpage initially), one hit to the local cache (to get the first word of the second subblock), and fourteen hits to the subcache (to get the remaining 14 words of the subpage). The subcache and local cache latencies of 2 and 18 processor cycles, respectively, increase the time between requests to the ring. Experiments A and B show the effect of different access granularities.

When no additional time is used for processing (i.e., pure read requests), the single request to the ring outweighs the other delays since it is an order of magnitude greater than the local cache latency. The rate at which read requests are made to the ring may be slowed further by introducing a variable delay between read accesses to simulate data processing. Experiment C shows the effect of introducing delay between read accesses.

In Experiment D the access pattern is varied in order to study the effect of automatic prefetching. If many subpages are copied from the ring before they are requested, then the number of ring requests will be reduced. This has the effect of reducing total execution time.

Experiments E and F show the effects of multiple writers. With multiple writers, the data set is divided equally among the writers so that each writer has the valid copy of a distinct (private) portion of the data set. When multiple writers own different parts of a shared data set and multiple readers read different parts as well, the composition of read requests being placed on the ring changes and the read time per subpage changes. Readers may or not may not be allowed to share data sets. Two extremes are considered:

1. Full sharing, where each reader reads the entire data set. This is termed *global* readers.
2. No sharing, where the data set is divided equally into distinct portions among the readers, and each reader accesses only its portion. This is termed *private* readers.

Experiment E investigates the effects of multiple global readers. It is possible that a single writer could become the system bottleneck. Multiple writers can reduce this bottleneck effect. Also, it is possible for a reader to obtain a valid copy of a subpage through automatic prefetching because of a request made by another reader.

Experiment F shows the effect of multiple writers and private readers. With private readers, readers cannot take advantage of automatic prefetching since each reader is the only thread accessing the data for which it has put a request on the ring. With multiple writers and private readers, read requests are served by different writers at the same time, which reduces the demands on the writer process.

Since poststore reduces the execution time of the reader threads while increasing the execution time of writer threads, both thread types should be considered when making the decision of when to use poststore. It is expected that for a low reader-to-writer ratio the expense to the writers would dominate, indicating that poststore should not be used. Conversely, for a high reader-to-writer ratio, it is expected that the benefits to the readers would dominate, indicating that poststore should be used. Also, as the number of reader/writer workloads (i.e., heavyweight threads, multiprogramming level) changes, the relative benefit of poststore can be affected. Experiment G examines these issues.

3.3. The Results

The results of the 7 experiments are presented here. Except for Experiment G the performance metric used is the average access time per subpage by an average reader thread.

In Experiments A, B and C there is a single writer and progressively longer times between read requests. In each of these, the average read time per subpage is shown as the number of readers varies from 1 to 30. The results of Experiment A are shown in Figure 2. Experiment A has the highest rate of ring requests (one per subpage). Results are shown for the three data set sizes. When poststore is used, read time per subpage is constant, since every read is a hit in

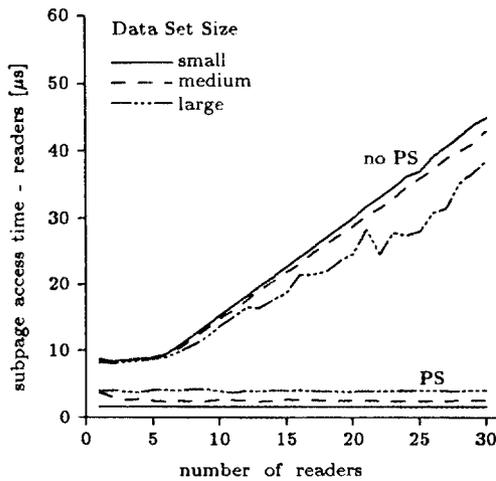


Figure 2: Granularity: 1 per subpage, Experiment A.

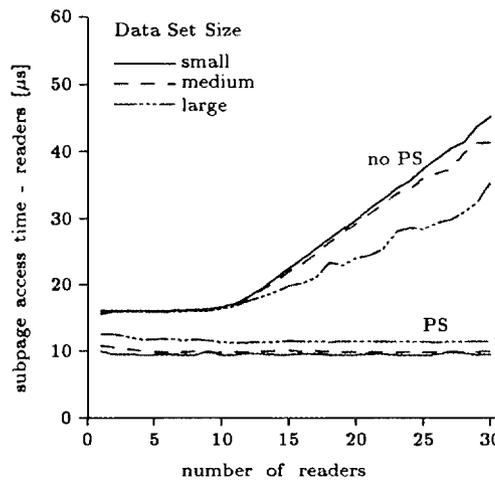


Figure 3: Granularity: entire subpage, Experiment B.

the local cache. With a larger data set, the average time to read a subpage increases because of the extra overhead incurred due to more subcache turnover. When no poststore is used, the average time to read a subpage increases as the number of readers increases. Regardless of the size of the data set, when more than six readers are executing, the time to read a subpage increases linearly due to delays at the cell of the writer thread which must handle all requests. Larger data sizes yield better performance because they allow for better exploitation of the pipelined execution, and the subcache turnover overhead is overlapped with the time the processor is waiting for the requested subpage. Furthermore, a longer global execution time favors automatic prefetching. This is because the longer readers execute, the more their executions are staggered from the initial synchronized start, increasing the probability that one reader will request a subpage that will be needed in the future by another reader.

In Experiment B every word in each subpage is read. The results are similar to those of Experiment A, as shown in Figure 3. Again, when no poststore is used, the average time to read a subpage increases as the number of readers increases. The increase becomes linear with the same slope as before but begins with a higher number of readers, since the request rate is smaller. The point where the curve reaches the asymptote is 11 readers, as Figure 3 shows. The absolute value of the average read time per subpage is larger than with Experiment A due to the extra accesses performed per subpage. However, when the system is not saturated, the difference between the average read time with poststore and the average read time without poststore is the same, and is equal to the measured ring latency.

Experiment C shows the effect of including a variable delay to represent processing time between each read, which further reduces the ring request rate. In this experiment, one word per subpage is read, so that with no poststore, every read generates a ring request. The curves

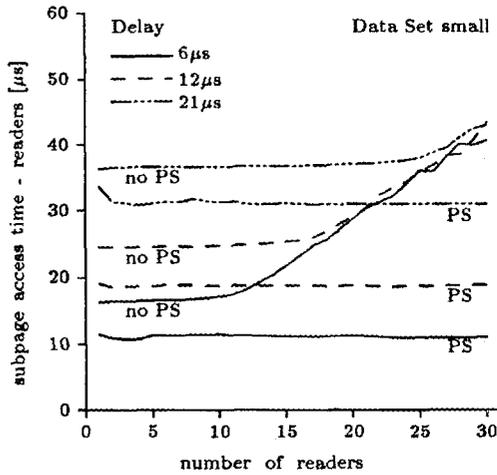


Figure 4: Readers with extra processing delay, Experiment C.

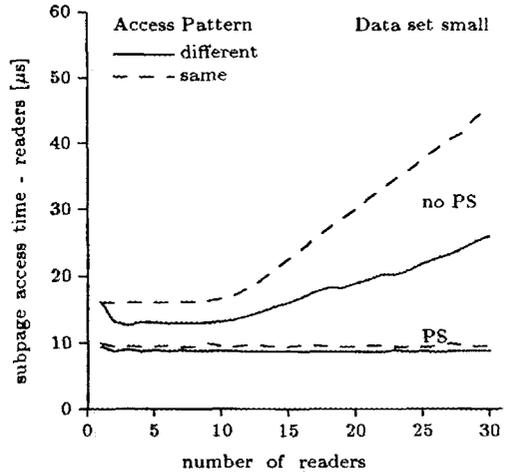


Figure 5: Access patterns, Experiment D.

in Figure 4 show the average read time per subpage for different delays between reads, as the number of readers increases from 1 to 30. The base case for a delay of $6\mu s$ yields the same performance as when an entire subpage is read and there is no delay between reads, as in Experiment B with the small data set. As the delay increases, the number of readers that it takes to saturate the system is larger. At saturation, the slope of the asymptote is the same as before for all curves, but the location of the saturation point is a function of the delay. Again, when the system is not saturated, the difference between the average read times with and without poststore is the same, and is equal to the measured ring latency. Experiment C shows that as the delay between reads increases, the ring latency and writer response time have less effect on total execution time.

Experiment D shows that performance improves if readers use different access patterns, as illustrated in Figure 5. In this experiment there is one writer, readers are global, and the number of readers is varied from 1 to 30. Half of the readers read the entire data set sequentially forward, and half of the readers read the entire data set sequentially backward. Figure 5 shows that the slope of the saturation asymptote for the average read time without poststore is about 50% of the slope for the corresponding experiment where all readers use the same access pattern (Experiment B). The performance improvement is due to the automatic prefetching of subpages that have not yet been requested as they pass by on the ring. This effect of prefetching is noticeable from one to two readers. The read time drops because there is a high probability that subpages requested by the second reader are copied by the first reader also, and vice versa. This is an instance of "anomalous" behavior where performance improves as the workload increases. When both readers have read half of the data set, the probability of generating ring requests is very low. Additional readers do not give any advantage, since their

read pattern is the same as one of the first two. As more reader threads are added, performance degrades less severely than in the other cases because during the second half of the execution the number of ring requests is reduced.

In Experiments E and F the number of readers varies from 1 to 29, and the number of writers varies from 29 down to 1, with the number of active threads fixed at 30. The results of Experiment E are shown in Figure 6. In this experiment, every reader reads the entire shared data set with the same reference pattern, and requests are satisfied by one writer at a time. At different times during execution, different writers supply the requested subpages. Because all readers tend to access similar parts of the data set at the same time, the trend is for a single writer at a time to be responding to reader requests. Thus, the expected improvement in execution time by spreading the requests among multiple writers is not realized. Figure 6 shows that average read time per subpage follows the same trend as in Experiment A, where there is a single writer and multiple global readers reading one word per subblock.

The results of Experiment F are shown in Figure 7. In this experiment, no two readers read the same piece of data, so no duplicated requests for the same subpage are seen on the ring. The data is distributed evenly among the writers. Thus, the readers segregate their read requests. Each reader will read data from a different set of writers, unlike Experiment E where each reader makes requests of each writer. When readers access distinct parts of the data set, the saturation behavior and the low load behavior are different. The slope of the asymptote is much steeper and occurs at a much higher number of readers due to the load balancing which occurs at the writers. The effect of many writers using poststore to a very few readers is also shown in this graph. With 29 writers and 1 reader the time to access a subpage is higher because not all poststore instructions were effective. The single reader was saturated with poststores from 29 writers and could not process all of the poststores.

The effectiveness of poststore is a tradeoff between the total time it takes the writer to update and poststore the data, and the reduction in read time for the readers. Figure 8 shows the sum of average access time per subpage for readers and the average time to write a subpage for the writer as a function of the number of readers, for the medium data set. The data is taken from the Experiment B runs for both with and without poststore. When the number of readers is small, the additional time it takes the writer to poststore is not offset by the savings in average access time for the readers. However, as the number of readers increases, the average access time of the readers in the without poststore case increases, while the write time is always constant. After approximately 15 readers the savings in access time for the readers with poststore is greater than the extra time required for the writer to perform the poststore.

Experiment G illustrates a similar tradeoff as the number of workloads (i.e., reader/writer sets, heavyweight threads, multiprogramming level) increases. The data was collected by si-

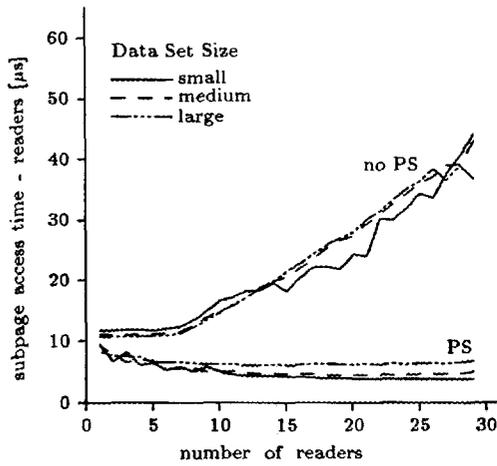


Figure 6: N global readers, $30 - N$ writers, Experiment E.

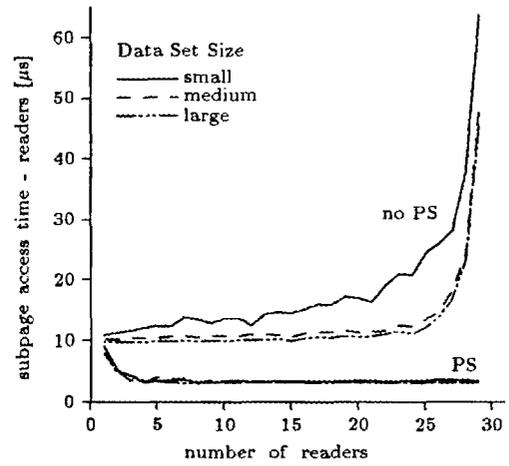


Figure 7: N private readers, $30 - N$ writers, Experiment F.

multaneously executing multiple copies of the workload from Experiment C with 29 readers with a processing delay of $9\mu s$, and 1 writer. Figure 9 graphs the average over all workloads of the combined access time for the readers and writers as the number of concurrently executing workloads increases. As the number of workloads increases (i.e., as the system load increases), the advantage of using poststore decreases. With 4 or more workloads, the average response time is lower without poststore. One possible reason is that retrieval of subpages from the ring can occur during the time that a thread is suspended due to context switching between workloads. When poststore is used, the time a thread is suspended (because it has been context switched with threads of other workloads) cannot be overlapped with data fetching. This tends to nullify the advantage of broadcasted updates. The higher the number of workloads, the more evident this effect becomes. In Figure 8 the advantage of poststore is more significant when there are more reader threads. The tradeoffs shown in Figure 8 and in Figure 9 explain why the decision of when to use poststore should be shared by the programmer and the system. As the system load increases, programmed poststores should be ignored by the system.

In general, the higher the rate at which non-local shared data is read, the greater the advantage of poststoring, especially when many other threads share that data. However, the number of threads which access the same data, and their access patterns, are other important factors to consider. When strict serialization of writes and reads cannot be ensured *a priori*, the use of poststore should be limited. When there are pending requests for a subpage for which a poststore has been issued, the poststore instruction is started but not completed, so no update broadcast is performed. This results in pure overhead for the writer.

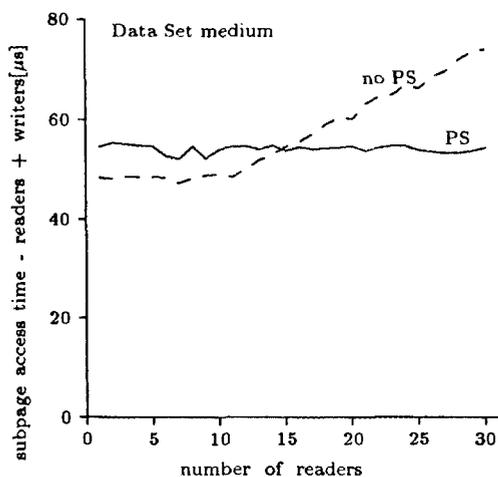


Figure 8: Combined reader and writer access time, single reader/writer workload.

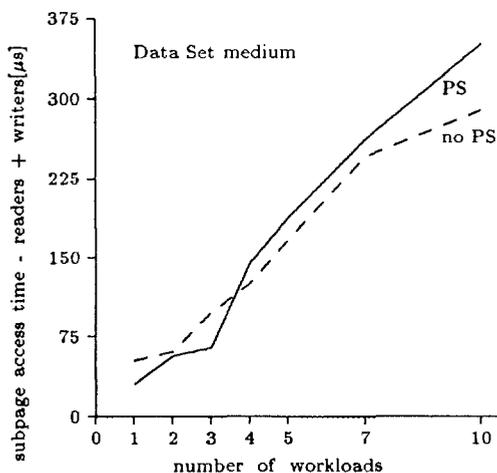


Figure 9: Combined reader and writer access time, multiple reader/writer workloads, Experiment G.

4. Modeling and Validation

4.1. Detailed Model

In this section, analytical models of the system and the workloads presented in Section 3.2 are presented. The workloads modeled are the various applications of readers and writers with and without the use of poststore. The analytic model illustrates the processing which occurs in the subcaches, local caches, and the ring under the selected workloads. The following modeling assumptions are made:

1. Initial modeling will include only the effects of subcaches, local caches, and ring:0 traffic. However, the models could be extended to include disk accesses and ring:1 traffic.
2. No cache inconsistencies or synchronization occur among the reader/writer threads. Specifically, all writing completes before any reading occurs. The hardware guarantees cache consistency and the modeled workloads have no synchronization.
3. Access times for the subcache, local cache, and ring are exponentially distributed, with a mean given by the hardware specifications of the KSR1 (see Table 1).
4. Each processor may make a memory request to the subcache, local cache, or ring:0 based on probabilities which are determined by the specific workload running on the processor.
5. A request placed on the ring and the removal of a request may be effectively modeled probabilistically. That is, it is not necessary to track the exact path of every request on the ring and that modeling average path behavior is sufficient.

For the workloads modeled, each cell does some processing, followed by a memory request. When a memory request is made from a processor, the item may be located in either the subcache, the local cache, or the local cache of another processor. If the item is in the subcache, then it is transferred directly to the processor. If the item is found in the local cache, then it is transferred to the subcache, and then to the processor. If the item is not found locally in either the subcache or local cache of the processor, then a request is issued on ring:0 for the data item. When the response arrives, the data item is placed first in the local cache, then the subcache, then sent to the processor.

A Generalized Stochastic Petri Net (GSPN) [9,8] was selected to model the system. The detailed GSPN model includes a subnet for each of the 32 processing cells and subnets for the two ARDs which model the ring propagation only. Each subnet models the cell's processing time, and subcache (sc), local cache (lc), and ring interactions. The subnets are connected together to form the complete ring:0.

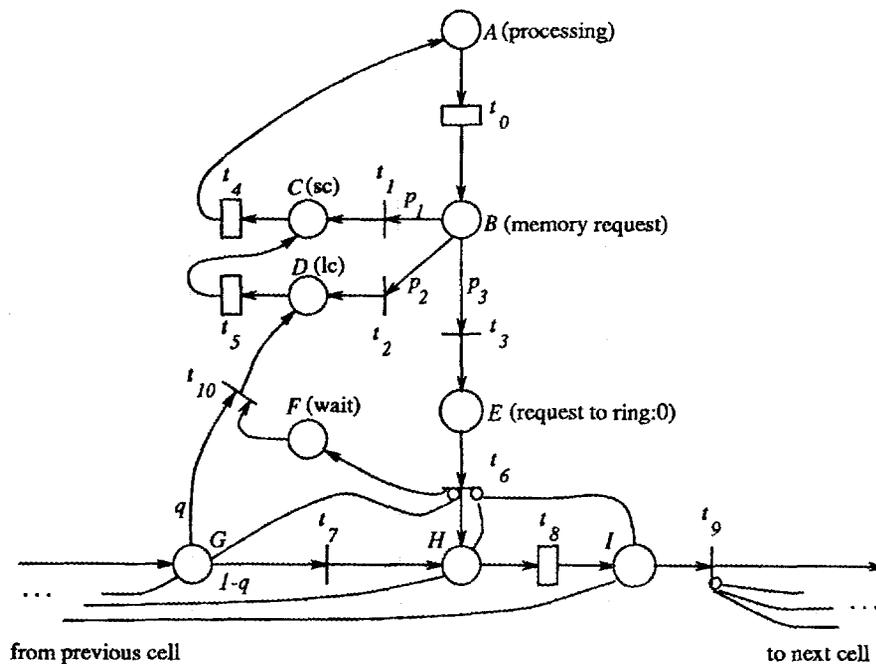


Figure 10: A subnet of one cell of the KSR1.

Figure 10 illustrates the detailed model of the subnet of one cell of the system. Places in each processor are labeled A through I. Transitions are numbered t_0 through t_{10} at each cell. The traffic on the ring is expressed by the number of occupied slots. Each cell has access to one slot, and this single slot is represented by the three places G, H, and I, in Figure 10. Inhibitor arcs on transition t_6 ensure that a cell can only place a message into an empty ring slot. Inhibitor arcs on transition t_9 from each of the places G, H, and I of the next cell ensure

that a message on the ring will only be passed to the next cell if the slot for that cell is empty. Throughput on the ring at a processor can be measured as the throughput of transition t_8 at that processor. Throughput of a processor can be measured as the throughput of transition t_0 , since all transactions at the processor must pass through that transition.

The subnet of a reader (i.e., a cell where the executing thread is a reader) operates as follows: Place A represents processing that occurs between memory requests. Transition t_0 is a timed transition which represents this processing time, and its rate depends on the volume of computation/processor cycles the reader is executing between two consecutive read requests. If a token is in place B , a memory request has been issued. After the memory request is issued, one of the immediate transitions t_1 , t_2 , or t_3 is fired with probability p_1 , p_2 , or p_3 , respectively. If the requested item resides in the cell's subcache, t_1 fires. If the subpage containing the item resides in the cell's local cache, transition t_2 fires. If the subpage containing the item resides in the local cache of another cell, t_3 is fired. The firing probabilities of transitions t_1 , t_2 , and t_3 depend on the workload type. The modeling of automatic prefetching is approximated by adjusting the probabilities p_1 , p_2 , and p_3 .

A token in place E represents a pending request to the ring. As soon as a slot becomes available, transition t_6 will fire, representing a request which is propagated on the ring. At the same time, the processor will go into a wait state, represented by place F , until the request is satisfied. Upon arrival of the response to place G , transition t_{10} is fired and the packet (i.e., the requested subpage) is received from the ring. The probabilities that a reader acquires the subpage from the ring or not are q and $1 - q$, respectively. Transitions t_4 , t_5 , and t_8 are timed transitions with firing times equal to the hardware latencies given by the manufacturer for the subcache, local cache, and the rate of ring propagation, respectively. Reader cells are initialized by placing a token in place A of each cell which represents an active reader process. This indicates that a read request is about to be made.

The subnet of a writer (i.e., a cell where the executing thread is a writer) operates similarly, except that the probabilities and transition rates are different. In each writer cell, q and $q - 1$ represent the probabilities that a writer does or does not own the subpage requested from the ring. Transitions t_0 , t_4 , and t_5 represent the total time for a writer to respond to a request. The probabilities p_1 and p_2 are zero for a writer thread, since no additional processing takes place, and the writer immediately issues a response on the ring as soon as a slot is available. Writer cells are initialized by placing a token in place F of each active writer thread, indicating that the writer is waiting to respond to a request.

The detailed model is a description of the interactions between threads and ring:0 of the KSR1. However, the detailed model contains 294 places and 358 transitions. Even a simple workload of 1 reader and 1 writer generates a reachability set containing over 800 states. Since

the addition of each new active thread causes the number of states to increase exponentially, the model quickly becomes intractable with just a few active threads. Since this detailed model cannot be solved easily, simulation or an approximate model must be used. The latter option is chosen.

4.2. Approximate Load Dependent Model

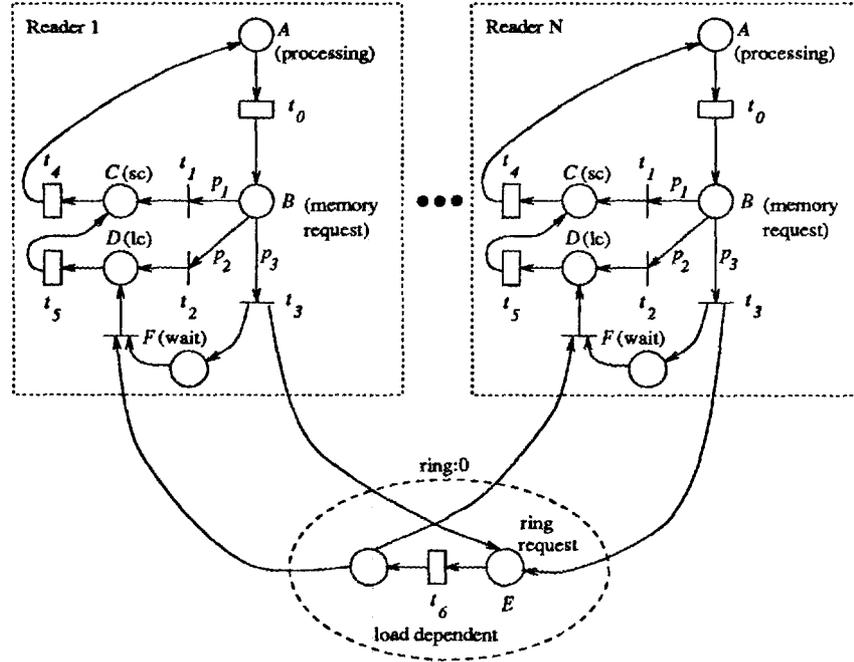


Figure 11: Load dependent GSPN model of ring:0.

The readers are modeled with the approximate load dependent model illustrated in Figure 11. In this model, transitions t_0 through t_5 , and places A through F are as described for the detailed model. However, the ring delay (i.e., ring and writer activity) is modeled as a single load dependent server, and all processes interact through this single resource. Figure 12 illustrates the experimentally measured service rate of the ring and a single writer thread. Since the access rate increases linearly up through six readers, and flattens thereafter, an M/M/6 server is used in the approximate model.

If the assumption is made that a single thread executes at a time on each processing cell and each cell is statistically identical (i.e., single class), then the model can be reduced. The equivalent model shown in Figure 13 results after collapsing all subnets that represent the readers of the approximate load dependent model into a single subnet. This model is initialized by placing a number of tokens in place A equal to the number of readers. This model gives the same global performance metrics for the reader threads as the model in Figure 11. The

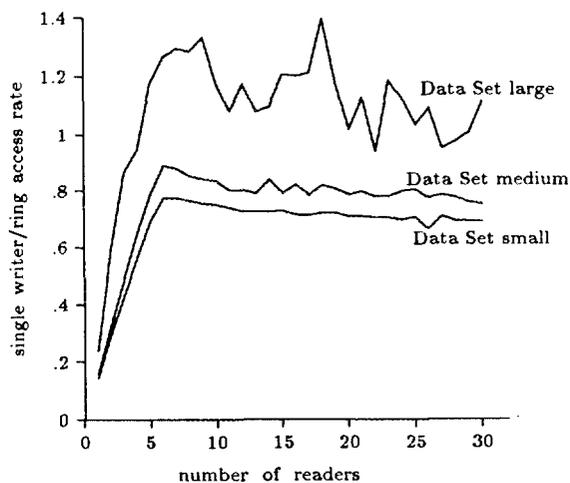


Figure 12: Service rate of the ring versus the number of active readers/threads.

response time measured with this model is the access time of a word.

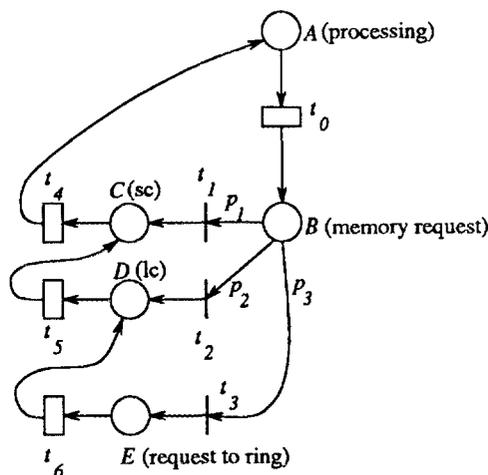


Figure 13: Reduced load dependent GSPN model for ring:0.

The number of tokens in this model indicates the number of readers. As before, tokens in place A represent internal cell processing. Tokens pass to place B when processing is complete and a memory request is issued. Transitions t_1 , t_2 , and t_3 are immediate transitions, with the same functionality as that of the detailed model. A token in place C indicates that the requested word is in the subcache. A token in place D indicates that the word is in the local cache. A token in place E indicates that a fault to ring:0 has occurred. Transitions t_4 , t_5 , and t_6 are timed transitions. Transitions t_4 and t_5 are infinite servers, representing the subcache and local cache of each reader thread, again with rates equal to the hardware rates specified by

the manufacturer for subcache and local cache access, respectively. Transition t_6 is an M/M/6 server, with a rate equal to the hardware rate specified by the manufacturer for ring:0 access. As more processes attempt to place messages on the ring, the server becomes saturated and processes requests at a fixed maximum rate. As before, p_1 , p_2 , and p_3 depend on the modeled workload.

4.3. Theoretical Model/Experimental Comparisons

In this section, comparisons between the theoretical response time curves (dashed lines) and the experimentally observed response time curves (solid lines) of various workloads are presented. The theoretical results are based upon the reduced load dependent GSPN model. The model was programmed and solved using SPNP [1], an analytic GSPN solver. The service rate of t_0 is workload dependent and depends on the amount of delay or computation performed after each read. The granularity of access of the experimental workload is reflected in the transition probabilities p_1 , p_2 , and p_3 . The size of the experimental workload affects both the amount of overlap of subcache overhead with processing and the effectiveness of pipelining. Experiments with different data sets yield different response time curves. However, the model does not incorporate any information about these types of overhead. The analytically predicted response times apply to the workload, regardless of the size of the data set. The medium data set is selected as representative and is used for comparisons to the theoretical model. The performance metric of interest is the average read time per subpage.

Figure 14 shows comparisons for Experiment A. For the average read time without poststore, the transition probabilities p_1 , p_2 , and p_3 are set to 0.0, 0.0, and 1.0, respectively (i.e., all reads generate a ring request). For the average read time with poststore, the transition probabilities are set to 0.0, 1.0, and 0.0 (i.e., all reads are a subcache miss, but a hit to the local cache). The model prediction is quite good. The analytical model overestimates performance by at most by $5\mu\text{s}$ (12.5%). In this case, the writer (i.e., the load dependent server) becomes the system bottleneck.

Figure 15 shows comparisons for Experiment B, where the global readers read the entire subpage. For the without poststore curve, the transition probabilities t_1 , t_2 , and t_3 are set to $\frac{14}{16}$, $\frac{1}{16}$, and $\frac{1}{16}$, since each subpage consists of two subblocks of 8 words each and as soon as a request is made to the ring, the subpage is moved to the local cache. For the with poststore curve, the transition probabilities are set to $\frac{14}{16}$, $\frac{2}{16}$, and $\frac{0}{16}$, since all read requests are satisfied in either the subcache or local cache. The theoretically predicted response time overestimates the experimental results by about 15%.

Similar comparisons between the analytic model and the experiments can be observed for Experiment C, as shown in Figure 16. (In the remaining figures, the with poststore curves do

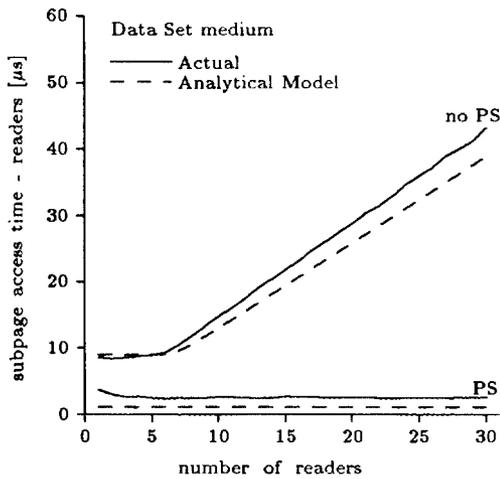


Figure 14: Model prediction for Experiment A.

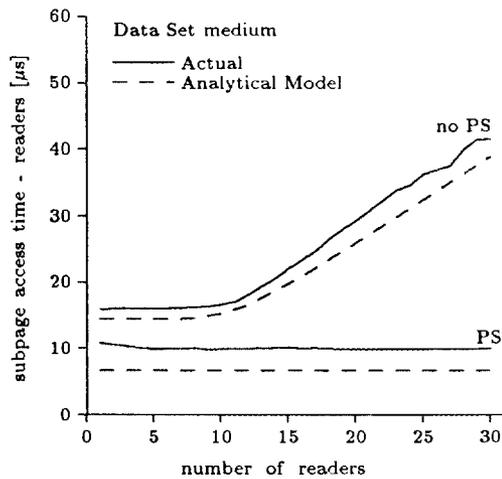


Figure 15: Model prediction for Experiment B.

not provide any additional insight and have been deleted for clarity.) In Experiment C, delays of $6\mu s$, $12\mu s$, and $21\mu s$ are added after each read to simulate processing time. The rate for the timed transition t_0 is adjusted to account for this in the model. In this experiment, the rate of ring requests is the slowest (in contrast to Experiment A where the relative rate of generating a ring request is the highest possible). As before, the model predictions follow the trend of the experimental response time curves.

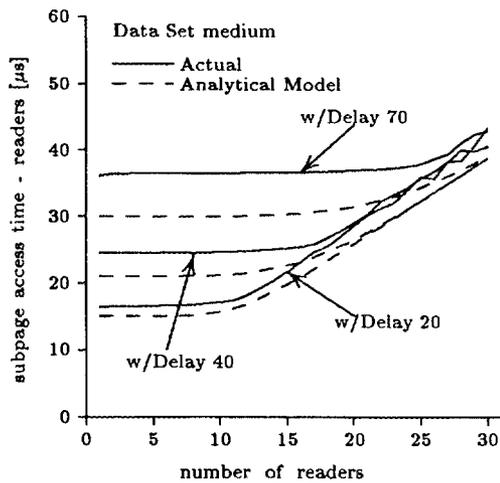


Figure 16: Model prediction for the "no poststore" case, Experiment C.

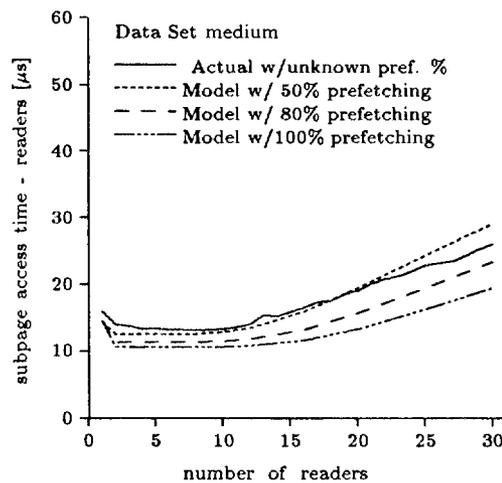


Figure 17: Model prediction for the "no poststore" case, Experiment D.

Figure 17 shows the predicted performance of the model along with the actual performance of Experiment D. For Experiment D (i.e., in the single writer/multiple reader case where half of the readers read sequentially forward, from the beginning to the end, the other half read sequentially backwards), the performance improves (i.e., compared against Experiment B where

all readers read in the same direction). Improvement results due to readers collecting subpages that they have not yet referenced but for which they have subpage descriptors. By adjusting the probabilities p_1 , p_2 , and p_3 it is possible to capture the effect of automatic prefetching of some percentage of the circulating subpages. The model indicates that the actual system is prefetching roughly 75% of the circulating subpages.

5. Generalizations

The model presented here may be generalized in a number of ways. These generalizations include such features as a more accurate load dependent server, a multiclass model, and less extreme workloads.

In Experiments A through D there is only one writer, with multiple readers which all behave similarly. This makes it possible to build a simple load dependent model of the readers on ring:0. The model reflects the readers' interactions with the "system", which is viewed as the combination of the ring and single writer, and is modeled as an M/M/6 server. The parameters of each model reflect the different behavior of the readers in each of the Experiments A through D. However, the simple load dependent model is not as accurate for Experiments E and F. Experiments E and F are different from the first four because there is more than one writer. In particular, when the readers are private, as in Experiment F, each reader is accessing data from a different writer at any one time.

Figure 18 shows the predicted performance using two different analytical models along with the actual performance for Experiment F. The first model is the one used in the previous section and uses a single M/M/6 server to model the ring/writer behavior. As seen, the model is a poor predictor for the case of multiple writers and private readers. The second model uses multiple M/M/6 servers, one for each writer. The behavior of this model is very close to that of the actual system. When there are 29 writers and 1 reader, the large number of writers (M/M/6 servers) can easily handle the number of requests from the single reader. Both the actual system curve and that from the analytic model are flat up to 25 readers (and 5 writers). If each writer behaves as an M/M/6 server, then there are equivalently 30 servers to handle the requests of the 25 readers. The system is behaving as an infinite server up until that point, since the number of readers is smaller than the total number of servers. For 26 readers there are 4 writers, with the number of total servers equal to 24. At that point, the read time per subpage begins to increase dramatically, since the number of readers is greater than the number of servers. With 29 readers and 1 writer, the writer is saturated, as in the earlier experiments.

The current model approximates the effect of automatic prefetching by adjusting the probabilities p_1 , p_2 , and p_3 . A multiclass model could be used to show this effect more accurately by modeling each request as a separate class. Each reader would issue a ring request for a class

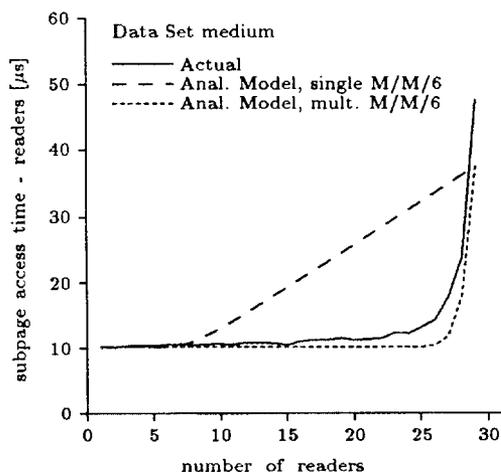


Figure 18: Model prediction for Experiment F.

of data not previously seen on the ring. Each reader would access future data on the ring with some probability dependent on other processor activity. The Petri net model would have a place for each class of data in the local cache and subcache. A more general model could also take into consideration subcache overhead and pipelining effects.

The current experimental workload was selected with the goal of illustrating the worst case behavior with and without the use of poststore. This workload is extreme in that all writing is completed before any reading starts, and the workload ensures that either all data is available in the local cache for each individual reader, or no data is available in the local cache for each individual reader at the time the reading occurs. Further work includes monitoring actual codes to acquire model parameters for the processing rates and the probabilities of memory requests of less extreme workloads.

6. Summary

The primary contributions of this paper are listed below, relative to the stated goals in the introduction.

- A description of the basic KSR1 architecture has been given. The key elements are: the ring of rings structure, the hierarchical (ALLCACHE) caching scheme, and the address resolution search engine. Attention is focused on the poststore option.
- Results from a suite of sensitivity analysis experiments have been reported. A simple readers/writers workload was used. Each experiment was run both with and without the poststore option. Performance sensitivity results were given with respect to: data

granularity, reader delay, data access patterns, reader/writer ratio, data sharing, total system load, data set size, and number of readers.

- Analytic models, both a detailed model and an approximate model, have been constructed. Validation experiments indicate that the basic trends are accurately captured using relatively simple models.

The experimental results indicate where poststore is most effective. Figure 8 shows that as the number of reader threads increases, poststore is more advantageous. However, if the number of reader threads is small, poststore should not be used since the benefit to the readers does not offset the extra incurred overhead of the writers. Thus, the number of reader threads influences the decision of whether or not to use poststore, and this number is a parameter of the programmer's application code. Figure 9 shows that as the number of application workloads (i.e., the number of sets of reader/writer codes) increases, poststore becomes less advantageous. That is, as contention increases at the processing cells, poststore is not beneficial and should not be used. The system load is controlled by the operating system scheduler. Therefore, neither allowing a programmer to use poststore without knowledge of the system load, nor allowing the operating system to determine the use of poststore without knowledge of the application code, is advisable.

Using poststore is analogous to a sender-initiated transfer. Sender-initiated transfers are most beneficial under light load [4]. Using prefetch (i.e., another programmer option not addressed in this paper), or allowing the readers to pull in subpages as requested when poststore is not used, is analogous to a receiver-initiated transfer. Receiver-initiated transfers are most beneficial when the system load is heavy. Although applied here in a different context, the results in Figure 9 confirm these general findings.

As mentioned in Section 5, several improvements to the model are possible and other features of the KSR architecture warrant further study. The intent here was not to model all aspects of the KSR's memory or ring hardware. However, building on the basic understanding of the architecture, such a modeling effort would be useful. These modeling and experimentation efforts are continuing.

Acknowledgements

The helpful information, criticisms, and suggestions provided by Tom Dunigan, Rich Stirling, and Jim Rothnie have significantly improved this paper.

7. References

- [1] G. CIARDO AND J. K. MUPPALA, *Manual for the SPNP Package, Version 3.1*, Department of Electrical Engineering, Duke University, Sept. 1991.
- [2] M. DUBOIS, C. SCHEURICH, AND F. GRIGGS, *Memory access buffering in multiprocessors*, in 13th International Symposium on Computer Architecture, 1986, pp. 434-442.
- [3] T. H. DUNIGAN, *Kendall Square multiprocessor: Early experiences and performance*, Tech. Report ORNL/TM-12065, Oak Ridge National Laboratory, Apr. 1992.
- [4] D. L. EAGER, E. D. LAZOWSKA, AND J. ZAHORJAN, *A comparison of receiver-initiated and sender-initiated adaptive load sharing*, Performance Evaluation, 6 (1986), pp. 53-68.
- [5] E. HAGERSTEN, A. LANDIN, AND S. HARIDI, *DDM- a cache-only memory architecture*, IEEE Computer, 25 (1992), pp. 45-54.
- [6] KENDALL SQUARE RESEARCH, *KSR1 Principles of Operation*, Revision 5.5, Waltham, Ma., Oct. 1991.
- [7] L. LAMPORT, *How to make a multiprocessor computer that correctly executes multiprocess programs*, IEEE Transactions on Computers, C-28 (1979), pp. 690-691.
- [8] M. A. MARSAN, G. CONTE, AND G. BALBO, *A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems*, ACM Transactions on Computer Systems, 2 (1984), pp. 93-122.
- [9] M. K. MOLLOY, *Performance analysis using stochastic Petri nets*, IEEE Transactions on Computers, C-31 (1982), pp. 913-917.

INTERNAL DISTRIBUTION

- | | |
|--------------------|--|
| 1. B. R. Appleton | 17. T. H. Rowan |
| 2. A. S. Bland | 18-22. R. F. Sinovec |
| 3-4. T. S. Darland | 23-27. R. C. Ward |
| 5. J. J. Dongarra | 28. P. H. Worley |
| 6. T. H. Dunigan | 29. Central Research Library |
| 7. G. A. Geist | 30. ORNL Patent Office |
| 8. K. L. Kliewer | 31. K-25 Applied Technology Li- brary |
| 9. M. R. Leuze | 32. Y-12 Technical Library |
| 10. R. A. Manning | 33. Laboratory Records - RC |
| 11. C. E. Oliver | 34-35. Laboratory Records Department |
| 12-16. S. A. Raby | |

EXTERNAL DISTRIBUTION

- 36-40. Amy W. Apon, Computer Science Department, Vanderbilt University, Nashville, TN 37235
- 41. Donald M. Austin, 6196 EECS Building, University of Minnesota, 200 Union Street, S.E., Minneapolis, MN 55455
- 42. Clive Baillie, Physics Department, Campus Box 390, University of Colorado, Boulder, CO 80309
- 43. Edward H. Barsis, Computer Science and Mathematics, P. O. Box 5800, Sandia National Laboratory, Albuquerque, NM 87185
- 44. Robert E. Benner, Parallel Processing Division 1413, Sandia National Laboratories, P. O. Box 5800, Albuquerque, NM 87185
- 45. Donna Bergmark, 745 E & TC Building, Hoy Road, Cornell University, Ithaca, NY 14853
- 46. Roger W. Brockett, Harvard University, Pierce Hall, 29 Oxford Street Cambridge, MA 02138
- 47. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
- 48. Maria Calzarossa, Dipartimento di Informatica e Sistemistica, Università Degli Studi di Pavia, Via Abbiategrosso 209, I-27100 Pavia, Italy
- 49. Brian M. Carlson, Computer Systems Research Institute, University of Toronto, Toronto, Ontario M5S 1A1, Canada
- 50. Jagdish Chandra, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709

51. Melvyn Ciment, National Science Foundation, 1800 G Street N.W., Washington, DC 20550
52. John J. Dorning, Department of Nuclear Engineering Physics, University of Virginia Reactor Facility, Charlottesville, VA 22901
- 53-57. Lawrence Dowdy, Computer Science Department, Vanderbilt University, Nashville, TN 37235
58. Derek Eager, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195
59. Edward Felten, Department of Computer Science, University of Washington, Seattle, WA 98195
60. Geoffrey C. Fox, NPAC, 111 College Place, Syracuse University, Syracuse, NY 13244-4100
61. Offir Frieder, George Mason University, Science and Technology Building, Computer Science Department, 4400 University Drive, Fairfax, Va 22030-4444
62. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47401
63. C. William Gear, NEC Research Institute, 4 Independence Way, Princeton, NJ 08540
64. W. Morven Gentleman, Division of Electrical Engineering, National Research Council, Building M-50, Room 344, Montreal Road, Ottawa, Ontario, Canada K1A 0R8
65. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
66. Gene Golub, Computer Science Department, Stanford University, Stanford, CA 94305
67. Andy Grant, Computer Graphics Unit, Manchester Computing Centre, University of Manchester, Oxford Rd, Manchester M13 9PL, United Kingdom
68. Eric Grosse, AT&T Bell Labs 2T-504, Murray Hill, NJ 07974
69. John L. Gustafson, Ames Laboratory, 236 Wilhelm Hall, Iowa State University, Ames, IA 50011-3020
70. Robert M. Haralick, Department of Electrical Engineering, Director, Intelligent Systems Lab, University of Washington, 402 Electrical Engineering Building, FT-10, Seattle, WA 98195
71. Michael T. Heath, *National Center for Supercomputing Applications*, 4157 Beckman Institute University of Illinois, 405 North Mathews Avenue, Urbana, IL 61801-2300
72. John L. Hennessy, CIS 208, Stanford University, Stanford, CA 94305
73. Charles J. Holland, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332

74. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550
75. Gary Johnson, Office of Scientific Computing, ER-7, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585
76. Lennart Johnsson, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142-1214
77. Harry Jordan, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309
78. Malvyn Kalos, Cornell Theory Center, Engineering and Theory Center Building, Cornell University, Ithaca, NY 14853-3901
79. Kenneth Kennedy, Department of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77001
80. Michael Langston, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301
81. Richard Lau, Office of Naval Research, Code 111MA 800 Quincy Street, Boston Tower 1, Arlington, VA 22217-5000
82. Robert L. Launer, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709
83. E. D. Lazowska, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195
84. Tom Leighton, Lab for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139
85. James E. Leiss, Rt. 2, Box 142C, Broadway, VA 22815
86. Heather M. Liddell, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, England
87. Rik Littlefield, Pacific Northwest Laboratory, MS K1-87, P.O.Box 999, Richland, WA 99352
88. Ivo de Lotto, Dipartimento di Informatica e Sistemistica, Università Degli Studi di Pavia, Via Abbiategrosso 209, I-27100 Pavia, Italy
89. Allen D. Malony, Department of Computer and Information Science, University of Oregon, Eugene, OR 97403
90. Oliver McBryan, University of Colorado at Boulder, Department of Computer Science, Campus Box 425, Boulder, CO 80309-0425
91. James McGraw, Lawrence Livermore National Laboratory, L-306, P. O. Box 808, Livermore, CA 94550
92. Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801

93. Richard Muntz, Computer Science Department, University of California at Los Angeles, Los Angeles, CA 90024
94. David Nelson, Director, Office of Scientific Computing, ER-7, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585
95. Randolph Nelson, IBM, P.O. Box 704, Room H2-D26, Yorktown Heights, NY 10598
96. James M. Ortega, Department of Applied Mathematics, Thornton Hall, University of Virginia, Charlottesville, VA 22901
97. Merrell Patrick, Department of Computer Science, Duke University, Durham, NC 27706
98. David A. Poplawski, Department of Computer Science, Michigan Technological University, Houghton, MI 49931
99. Daniel A. Reed, Computer Science Department, University of Illinois, Urbana, IL 61801
- 100-104. Emilia Rosti, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39, 20135 Milano, Italy
105. Diane T. Rover, 155 Engineering Building, Department of Electrical Engineering, Michigan State University, East Lansing MI 48824
106. Ahmed H. Sameh, Department of Computer Science, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455
107. Robert B. Schnabel, Department of Computer Science, University of Colorado at Boulder, ECOT 7-7 Engineering Center, Campus Box 430, Boulder, CO 80309-0430
108. Robert Schreiber, RIACS, MS 230-5, NASA Ames Research Center, Moffet Field, CA 94035
109. Martin H. Schultz, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520
110. David S. Scott, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
111. The Secretary, Department of Computer Science and Statistics, The University of Rhode Island, Kingston, RI 02881
112. Charles L. Seitz, Department of Computer Science, California Institute of Technology, Pasadena, CA 91125
113. Giuseppe Serazzi, Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza Leonardo da Vinci 32, 20133 Milano, Italy
114. Kenneth C. Sevcik, Computer Systems Research Institute, 10 King's College Road, University of Toronto, Toronto, Ontario M5S 1A1, Canada
115. Horst D. Simon, NASA Ames Research Center, Mail Stop T045-1, Moffett Field, CA 94035

- 116-120. Evignia Smirni, Computer Science Department, Vanderbilt University, Nashville, TN 37235
- 121. Burton Smith, Tera Computer Company, 400 North 34th Street, Suite 300, Seattle, WA 98103
- 122. Marc Snir, IBM T.J. Watson Research Center, Department 420/36-241, P. O. Box 218, Yorktown Heights, NY 10598
- 123. Rick Stevens, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
- 124. Paul N. Swarztrauber, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
- 125. Anne Trefethen, Engineering & Theory Center, Cornell University, Ithaca, NY 14853
- 126. Mary Vernon, Computer Sciences Department, University of Wisconsin, 1210 W. Dayton Street, Madison, WI 53706
- 127. Robert G. Voigt, National Science Foundation, Room 417, 1800 G Street N.W., Washington, DC 20550
- 128-132. Thomas Wagner, Computer Science Department, Vanderbilt University, Nashville, TN 37235
- 133. Mary F. Wheeler, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251
- 134. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
- 135. John Zahorjan, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195
- 136. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P. O. Box 2001, Oak Ridge, TN 37831-8600
- 137-146. Office of Scientific & Technical Information, P. O. Box 62, Oak Ridge, TN 37831