The Effects of Thread Placement on the KSR1*

T.D. Wagner, E. Smirni, A.W. Apon, M. Madhukar, and L.W. Dowdy
Department of Computer Science
Vanderbilt University
Nashville, TN 37235

Abstract

This paper describes a measurement study of the effects of thread placement on memory access times on the Kendall Square KSR1 multiprocessor. The KSR1 uses a conventional shared memory programming model in a distributed memory architecture based on a ring of rings of 64-bit superscalar microprocessors. Memory consists of local cache memories attached to each processor and is managed in a Cache-Only Memory Architecture (COMA) fashion.

Experiments run on the KSR1 across a variety of thread configurations show that shared memory access is accelerated through strategic placement of threads which share data. The experiments "stress test" the automatic prefetching feature of the hardware. Strategies to keep the KSR1 memory access times nearly constant even when the number of participating threads increases are proposed.

1 Introduction

Typically, as the number of processors increases in a shared memory multiprocessor, memory access times also increase due to contention on a common communication path or at the shared memory. The KSR1 system has a distributed memory architecture that makes it possible for average memory access times to remain roughly constant as the number of processors grows. The architecture is based on a ring of rings of processing cells. Each processing cell has a 64-bit microprocessor and a local memory that is managed as a cache. Up to thirty-two processing cells are connected on a level 0 ring. Up to thirty-four rings may be connected in a level 1 ring, for a total of 1088 processing cells.

The KSR1 has a shared memory programming environment, but a valid copy of a data item must exist in the local cache of a processor in order to be accessed. Many processors may have a valid read-only copy of a data item in their local cache. The ALLCACHE Engine is the distributed mechanism responsible for finding a valid copy of a data item, copying it to a processor's local cache when referenced by that processor, and maintaining sequential consistency between caches.

One processor is the designated owner of each data item, but this ownership is not bound to any particular processor. When a processor writes a data item, it first obtains ownership of the data item in its local cache and all other copies of this item in other processors' caches are marked as invalid. When a processor reads a data item and a valid copy is not available in its local cache, then a read-only copy of the data item is obtained via the ALLCACHE Engine. Each subsequent read of the same data item will be to the local copy and will require a minimal amount of time. The memory architecture offers three ways to move a read-only copy of a data item to a local cache prior to a request: two programmer options *poststore* and *prefetch*, and one architectural feature *automatic prefetching.* Poststore and prefetch have been addressed elsewhere [1,2].

The focus of this paper is to examine the effects of automatic prefetching and thread placement in reducing average memory access times for shared data in a multi-threaded application. The results indicate that strategic thread placement across multiple rings of the KSR1 can substantially reduce memory access time for shared data items. Related work includes studies which focus on the implementation of specific applications on the KSR1 [3, 4], analysis using micro benchmarks [5], and performance studies of the KSR1 [6,7].

2 Memory architecture

The general KSR architecture is a multiprocessor system composed of a hierarchy of rings of proces-

*This work was partially supported by sub-contract 19X-SL131V from the Oak Ridge National Laboratory managed by Martin Marietta Energy Systems, Inc. for the U.S. Department of Energy under contract no. DE-AC05-84OR21400.

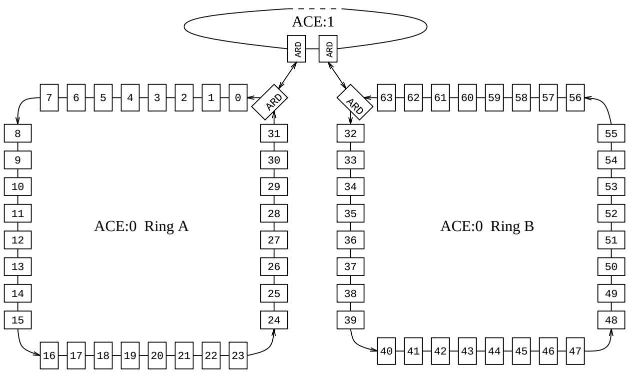


Figure 1: KSR1 with two ACE:0 Rings

sors [8, 9]. At the lowest level is the processor cell, which contains a 64-bit superscalar processor and 32 MB of local cache memory. Each processor has .5 MB of traditional cache referred to as subcache memory. Each processor cell is connected to two neighbors to form the lowest level ring. The lowest level ring, termed ALLCACHE Engine:0 (ACE:0), is a slotted ring with a peak data transfer rate of 1 GB per second. Each ACE:0 ring includes at least one special cell, termed the ALLCACHE Routing and Directory (ARD) cell, which is responsible for routing to the next higher level ring. Up to 32 processing cells may be connected on a single ACE:0 ring. The next higher level ring, termed ACE:1, is composed of up to 34 ACE:0 rings.

The KSR1 system used in this study consists of two identical ACE:0 rings, each containing 32 processing cells, linked on an ACE:1 ring (see Figure 1). Memory capacities and latencies are shown in Table 1 [8].

Each local cache has a directory which contains a three bit descriptor for each subpage in the cache that give the state of the subpage. There are three classes of subpage states: Invalid, Read-only, and Owner. A subpage is *invalid* if the page containing the subpage has a descriptor in the local cache, but the subpage is not present. If any processor holds a subpage in *read-only* state, then any number of other caches can also hold the subpage in read-only state. Before a subpage may be written to, it must be held in an exclusive *owner* state.

The ARD cells on each ACE:0 ring are responsible for directing requests between the ACE:0 ring and the higher level ACE:1 ring. Each ARD maintains a directory that includes the state of every subpage present on the ACE:0 ring to which it is attached. This state information specifies: 1) whether or not the owner of the subpage is on the local ring, 2) whether or not there are valid read-only copies on the subpage on the local ring, and 3) whether or not there are valid read-only copies on other rings. The state information is sufficient for an ARD to know whether or not a request can be satisfied on its local ACE:0 ring. An important task of the ARD is to serve as a filter for multiple requests to the ACE:1 ring for the same subpage by allowing only one of several identical requests to pass to the ACE:1 ring.

Table 1: Memory capacities and latencies

Location of subpage	Total capacity (MB)	Latency in cycles (50 ns)
Local subcache	0.5	2
Local cache	32	18
ACE:0	1,024	175
ACE:1	34,816	600
Disk		400,000

When there are many shared subpages in the system, the ALLCACHE architecture allows subpages to be copied prior to a request through *automatic prefetching*. In automatic prefetching, when a copy of a subpage is sent through the search engine to satisfy a request, any processor whose cache has a descriptor for that subpage which is invalid may acquire a read-only copy as the subpage passes by on the ACE:0 ring. Automatic prefetching takes place as long as the processor is not "too busy" performing other memory accesses [8]. Automatic prefetching is a powerful mechanism which reduces memory access time in applications with a high degree of read-only sharing. The combined effect of automatic prefetching, ARD filtering, and the placement of processing threads is the focus of the experiments in the following section.

3 Experiments

3.1 Workload description

Five suites of experiments examine the effects of the number and placement of processing threads. A synthetic workload is constructed which is executed in each suite of experiments. Two types of processing threads are used in the synthetic workload. An *owner thread* has the only valid copy in memory of the data set (i.e., is the owner of each subpage) at the start of each experiment. A *reader thread* has a descriptor for every subpage of the data set, but these descriptors are made invalid when the owner thread writes to the subpages. A reader thread requests a read-only copy of each subpage in its portion of the data set after the owner thread has written the entire data set.

The workload performs the following steps:

1. A number of reader and owner threads are spawned, each of which binds to a unique processor for the duration of the experiments.
2. Each reader thread and owner thread reads a predetermined portion of the data set.
3. Each owner writes its portion of the data set.
4. A barrier synchronization is performed for all threads.
5. Timing begins for each reader thread.
6. Each reader thread sequentially reads its portion of the data set.
7. Timing ends for each reader thread.

Steps 1 and 2 represents the overhead required for spawning threads, binding them to a processor, and allocating pages of local cache memory for the data set. Step 2 ensures that each local cache has a valid descriptor of every subpage in the data set so that all subsequent accesses to a subpage require only data movement, and do not require local cache memory allocation. The size of the data set for Suite 0 depends on the number of reader threads, but is at most 16 MB. The size of the data set for Suite I through Suite IV is 50K subpages (6.4 MB). The entire data set fits into the local cache thereby obviating the effects of paging to disk.

Step 3 sets the state of each subpage in each owner thread to exclusive owner and sets the state of each subpage in all other threads to invalid. The first access by a reader thread to a subpage causes the state of the subpage to become non-exclusive owner in the owner thread. In step 6, one word per subpage is read, so that one entire subpage is copied for each read operation. This is the maximum rate of data copying possible, emphasizing the effects of thread placement.

The experiments and the synthetic workload are designed to analyze how memory access times can be improved when only thread placement is varied. This workload exhibits behavior that is intended to test the limits of the hardware prefetch feature.

The suites of experiments progressively illustrate the effects of placement of reader and owner threads on the KSR1. The performance metric of interest in all experiments is the average access time per subpage.

3.2 Suite 0

The goal of Suite 0 is to identify the performance when no automatic prefetching or filtering by the ARD occurs, and all reader threads access the data from the same common owner. The methodology of Suite 0 is to eliminate the advantages of automatic prefetching by partitioning the data set among the reader threads, and measuring average access time per subpage as the number of reader threads varies from 1 up to 63. The owner thread is placed on Ring A in cell 31. The first 31 reader threads are placed on Ring A in processor order. The next 32 reader threads are placed on Ring B in processor order.

The data set is divided into disjoint subsets of size 2K subpages each, and each reader thread accesses a unique subset of the data set. Each reader thread reads the same number of subpages, irrespective of the total number of readers, in order to compare with the experiments in Suite I through Suite IV.

The graph in Figure 2 shows the average access time per subpage for readers on Ring A, the average access time per subpage for readers on Ring B, and the overall average access per subpage for all readers. The graph shows that the owner thread can satisfy requests in nearly constant time until the owner thread saturates at roughly 8 reader threads. As additional reader threads are added the average access time per subpage increases almost linearly.

Each reader thread accesses each subpage in its unique subset of the data set exactly one time, so that each reference to a subpage results in one request being sent to the owner. As the number of reader threads increases, the number of requests increases proportionally, the FIFO extract buffers used for extracting messages from the ring at the owner fill, and requests must be denied. Thus, queueing effects are seen, and the average access times per subpage increase proportionally to the number of reader threads.

3.3 Suite I

In all experiments in Suite I through Suite IV, all readers share a single large data set. The size of this data set is 50K subpages (6.4 MB). All readers read the entire data set.

The goal of the experiments in Suite I is to examine memory access times as the location of the owner of the data set is varied on the same ACE:0 ring. The methodology of this experiment is to measure the average read time per subpage as the number of reader threads varies from 1 to 63. The owner thread is placed on Ring A. The first 31 reader threads are placed on Ring A in processor order. The second 32 reader threads are placed on Ring B in processor order. The location of the owner thread is varied from cell 0 to cell 31 on Ring A.

Figure 3 illustrates the results of Suite I. Results

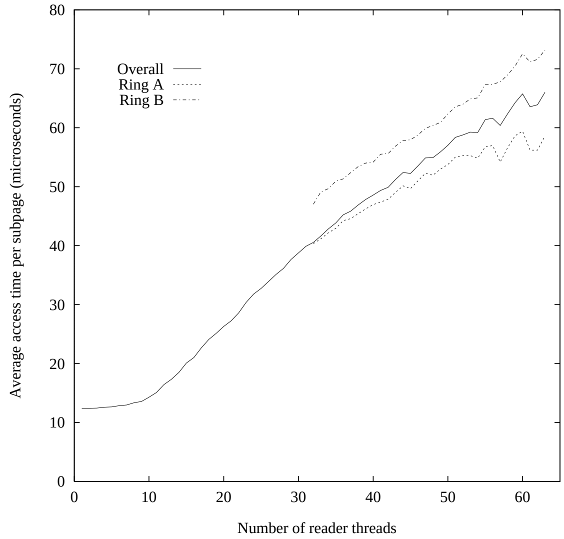


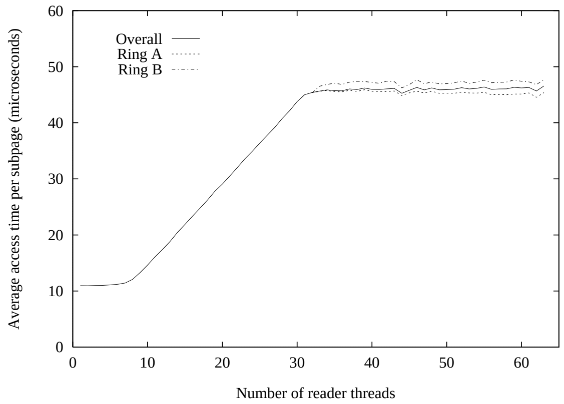
Figure 2: Suite 0: No automatic prefetching, one owner thread

were obtained for each possible location of the owner from cell 1 to cell 31, but are not presented for the sake of brevity. In Figure 3(a) the owner is on cell 1. This figure is characteristic of the performance obtained when the owner thread is placed on cells 1 through 11. In Figure 3(b), the owner is on cell 31 and is characteristic of performance obtained when the owner thread is placed on cells 12 through 31.

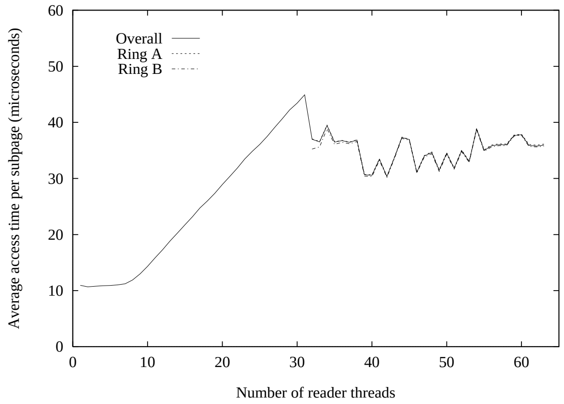
When the number of readers is less than 32, all readers are on the same ACE:0 ring, Ring A. When all readers are on Ring A, the results of this experiment show that average read time per subpage increases as the number of additional threads increases. Further, when all reader threads are on Ring A, the average read time per subpage is insensitive to the placement of the owner thread, as shown in Figure 3. Since all reader threads are accessing the same subpage at about the same time, little automatic prefetching takes place. This situation is similar to that shown in Figure 2. As the number of requests increases, more requests are denied at the owner, and queueing effects are seen.

When the number of reader threads is 32 or more, then at least one reader thread is placed on Ring B. When the owner thread is at cell 1 and at least one reader thread is placed on Ring B, then average access times are constant as the number of readers is increased beyond 32, as shown in Figure 3(a). This demonstrates the effect of the ARD filtering multiple identical requests from Ring B.

When the location of the owner thread is varied



(a) owner on cell 1



(b) owner on cell 31

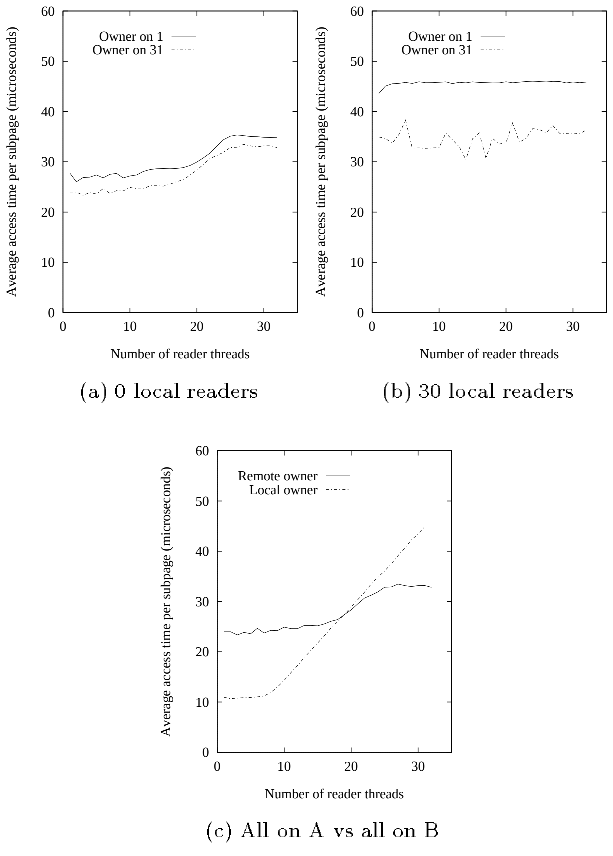
Figure 3: Suite I

around the ring, the average read times *decrease* for all threads on *both* Ring A and Ring B as the number of reader threads is increased beyond 31. This effect is observed in the experiments when the owner cell is on cells 13 through 31. Figure 3(b) shows average access times when the owner thread is on cell 31. In this case, the barrier release mechanism has released some of the remote readers before all of the local readers. This means that a request for the first subpage to be read is sent from a remote reader before all of the requests from local readers have been fulfilled. The net effect is that some of the remote readers can actually get ahead of the local readers and automatic prefetching at the readers local to the owner will be increased.

In Suite I, the owner is on Ring A. Reader threads on Ring A are on a *local* ring with respect to the owner of the data. Reader threads on Ring B are on a *remote* ring with respect to the owner of the data. It was found that placing some number of reader threads on a ring which is remote to the owner can improve the performance of reader threads on both Ring A and Ring B depending on the placement of the owner thread Suite II investigates tradeoffs in the number of remote threads versus the number of local threads.

3.4 Suite II

The goal of the experiments in Suite II is to measure average memory access times for remote threads as the number of local threads is varied. The owner of the data set is placed on Ring A. The methodology of this experiment is to vary the number of remote reader threads (reader threads on Ring B) from 1 up to 32. Four experiments in Suite II set the number of local reader threads (reader threads on Ring A) to be one of 0, 10, 20, or 30. The location of the owner thread is selected to be typical of the two types of performance observed in Suite I, either cell 1 or cell 31.



(a) 0 local readers (b) 30 local readers

(c) All on A vs all on B

Figure 4: Suite II

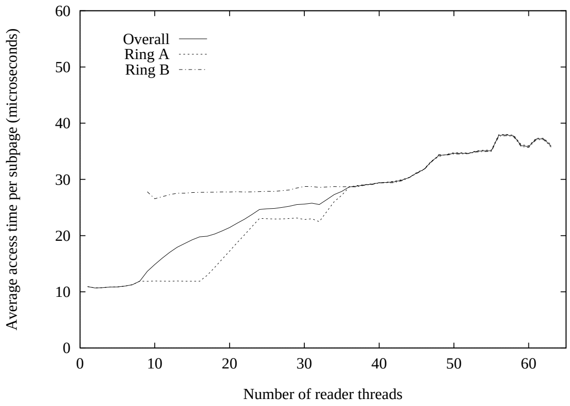
Figure 4(a) illustrates average read times per subpage for Ring B threads as the number of reader threads on Ring B increases from 1 up to 32. The owner is on Ring A, and 0 reader threads are on Ring A. When all readers are remote to the owner of the data, the placement of the owner on Ring A has a small effect on the average read times per subpage.

Figure 4(b) illustrates average read times per subpage for Ring B threads as the number of reader threads on Ring B increases from 1 up to 32. The owner of the data is on Ring A, and 30 reader threads are on Ring A. This graph is similar to the right half of the two graphs in Figure 3. When there are 30 local reader threads on Ring A, then the placement of the owner on Ring A has a significant effect on the average read times per subpage for the reader threads on Ring B. Figure 4(b) illustrates that average read times per subpage improve as much as 30%, depending on the placement of the owner thread. Experiments with 10 and 20 local reader threads give intermediate results.

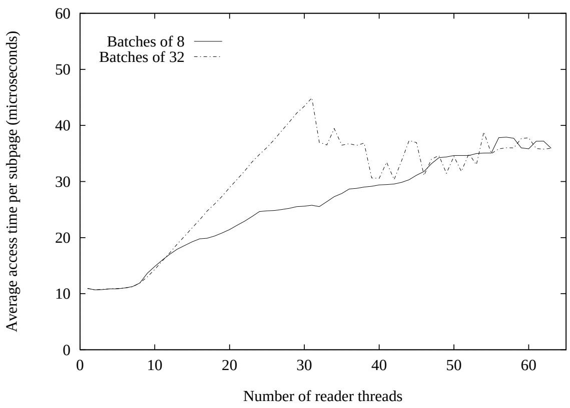
An interesting observation can be made for the graphs in Figure 4(a) and the left half of the graph in Figure 3(b). In both experiments the owner is on Ring A, and the total number of reader threads increases from 1 up to 31. In the left half of Figure 3(b) all readers are on Ring A, which is local to the owner. In Figure 4(a) all readers are on Ring B, which is remote to the owner. Figure 4(c) shows these two curves on the same graph. There is a crossover point in the graph. When more than 20 reader threads are executing, better performance is observed when all reader threads are on the *remote* ring as compared to when all readers are on the same ring as the owner. This effect is due to the behavior of the ARD, which filters requests from Ring B, and the effects of automatic prefetching by the reader threads on Ring B.

3.5 Suite III

The goal of the experiments in Suite III is to fix the location of the owner thread, and examine the effects of various placements of reader threads. Cell 31 on Ring A is chosen as the location for the owner (based on the results of Suite I). The methodology of this suite is to place from 1 up to 63 readers, one at a time, where the choice of ring is determined by the *batch size* of the experiment. The batch size is one of 1, 2, 4, 8, 16, or 32. If the batch size is N, then the first N readers are placed on Ring A, the second N readers are placed on Ring B, the third N readers are placed on Ring A, and so on, until all 63 reader threads are placed. Thus, $N = 32$ corresponds to the Suite I experiments. The results of the run with batch size equal to 8 is shown in Figure 5(a) where the overall average subpage access time and the access time for the processors on each ring is graphed. This case exhibits the most improvement over the original batch size of 32 from Suite I.



(a) Batches of 8



(b) Batches of 8 and 32

Figure 5: Suite III

One result of the filtering performed by the ARD is that many reader threads on Ring B appear to the owner as only one additional reader thread. While the average access time for the first eight readers bound to Ring B is somewhat higher than the average access time for the first eight readers on Ring A, the addition of those eight readers on Ring B have the same effect on the Ring A reader threads as adding only one local reader thread. When the second eight threads are added to Ring A, the Ring A curve takes on the characteristic shape seen in Suite I, but the times for Ring B do not change, as shown in Figure 5(a). Once Ring A is full, the curve is similar to the curve from Suite I, Figure 3(b), with the owner on cell 31. A comparison of the results using a batch size of 8 and the results from Suite I with the owner bound to cell 31 is shown in Figure 5(b). It is clear that the placement of the reader threads across the two rings can affect the overall average subpage access time, especially when the number of reader threads is less than the full complement of processing cells.

3.6 Suite IV

The methodology used in this experiment is that the reader threads do not access the subpages in the same order. Each reader thread starts reading at a different point in the data set. With this *staggered readers* technique each reader thread has a different access pattern from every other reader thread and the effect of automatic prefetching is more pronounced.

Figure 6 illustrates the average subpage access time of staggered readers as the number of reader threads varies from 1 to 63. The owner thread is placed on cell 31 on Ring A. The performance improvement is due to automatic prefetching of subpages as they pass by on the ring. This effect is noticeable from 1 to 2 reader threads. With only one reader thread automatic prefetching will not occur. When a second reader thread is added automatic prefetching occurs and the average access time drops. As the number of reader threads increases, the access time becomes flat, because a larger number of subpages that have not been requested yet are automatically prefetched by each reader thread. Thus, the owner thread can support more reader threads before it begins to saturate.

When the number of readers is larger than 31 (i.e., at least one reader thread is not on the same ring as the owner), the overall average access time increases because of the requests from the second ring. However, the access time of the reader threads on Ring B reduces as the number of readers on Ring B increases because more subpages are brought by the ARD to Ring B and automating prefetching becomes effective on Ring B as well.

Automatic prefetching does not take place between ACE:0 rings. Therefore, the readers on Ring B cannot take advantage of the subpages requested by Ring A processors (the ring local to the owner thread). The access time of the reader threads on Ring A remains flat because the reader threads on Ring A are able to continue automatic prefetching of the responses to requests from Ring B as they circulate on Ring A. After a threshold is crossed (about 57 readers in both rings), the access time on Ring A begins to increase due to queueing effects.

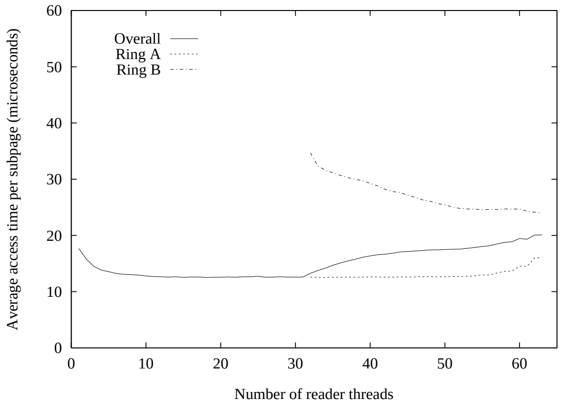


Figure 6: Suite IV: Staggered readers, owner thread on cell 31

4 Programming strategies and future work

The results of the experiments show that with strategic thread placement and coding techniques that take advantage of the architecture, memory access times on the KSR1 can remain fairly constant as the number of threads that share a data set increases. A number of implications for applications programmers can be identified:

- A good understanding of the architecture and its novel features are essential for improving the performance of application codes.
- The placement of the owner thread of a data set affects performance.
- Suite I shows that additional threads that share a data set can actually improve performance, as long as the threads are placed strategically to take advantage of automatic prefetching and ARD filtering.
- The placement of the owner thread of a data set particularly affects the performance of reader threads that are placed on a remote ring, as shown in Suite II.
- The strategic distribution of reader threads across multiple ACE:0 rings can improve performance substantially, as shown in Suite III.
- Code changes that allow a shared data set to be distributed among several owners, or stagger the access pattern among readers, can also substantially improve the performance, as shown in Suite IV.

The experiments illustrate several interesting and unexpected results. These findings indicate that several extensions to this work are appropriate. Such extensions include:

- The testing of actual application codes.
- The testing on KSR1 systems with multiple (i.e., more than two) ACE:0 rings.
- The combined analysis of explicit prefetch (i.e., the programmer option), poststore, and automatic prefetch.
- The testing and analysis of the effects of multiprogramming.

Acknowledgments

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

References

[1] T. Dunigan, A. Hoisie, and A. Trefethen, "Scalability evaluation of the KSR1," tech. rep., Cornell University, 1993.

[2] E. Rosti, E. Smirni, T. Wagner, A. Apon, and L. Dowdy, "The KSR1: Experimentation and modeling of poststore," in *Proceedings of the ACM Sigmetrics Conference*, 1993.

[3] E. Boyd, J. Wellman, S. Abraham, and E. Davidson, "Evaluating the communication performance of MPPs using synthetic sparse matrix multiplication workloads," in *Proceedings of the International Conference on Supercomputing*, 1993.

[4] D. Windheiser, E. Boyd, E. Hao, S. Abraham, and E. Davidson, "KSR1 multiprocessor: Analysis of latency hiding techniques in a sparse solver," in *Proceedings of the 7th International Parallel Processing Symposium*, 1993.

[5] R. Saavedra, S. Gaines, and M. Carlton, "Micro benchmark analysis of the KSR1," in *Proceedings of Supercomputing*, 1993.

[6] T. Dunigan, "Kendall Square multiprocessor: Early experiences and performance," Tech. Rep. ORNL/TM-12065, Oak Ridge National Laboratory, Apr. 1992.

[7] T. Dunigan, "Multi-ring performance of the Kendall Square multiprocessor," Tech. Rep. ORNL/TM-12331, Oak Ridge National Laboratory, Sept. 1993.

[8] Kendall Square Research, Waltham, Ma., *KSR1 Principles of Operation*, Revision 5.5, Oct. 1991.

[9] T. Wagner, E. Smirni, A. Apon, M. Madhukar, and L. Dowdy, "Measuring the effects of thread placement on the Kendall Square KSR1," Tech. Rep. ORNL/TM-12462, Oak Ridge National Laboratory, Aug. 1993.