Experiences with Data Distribution on NUMA Shared Memory Multiprocessors

Sudarsan Tandri and Tarek S. Abdelrahman
e-mail: {tandri,tsa}@eecg.toronto.edu
Department of Electrical and Computer Engineering
The University of Toronto
Toronto, Canada, M5S 1A4

Abstract

The choice of a good data distribution scheme is critical to performance of data-parallel applications on both distributed memory multiprocessors and NUMA shared memory multiprocessors. The high cost of interprocessor communication in distributed memory multiprocessors makes the minimization of communications the predominant issue in selecting data distribution schemes. However, on NUMA multiprocessors other issues such as contention, false sharing and cache affinity also affect performance. In this paper, we present empirical measurements which suggest that these issues cannot be ignored in selecting data distribution schemes. We conclude that existing methodologies used by application programmers and compilers to select data distribution schemes on distributed memory multiprocessors are not suitable for NUMA multiprocessors.

1 Introduction

Non-Uniform Memory Access (NUMA) multiprocessors have recently gained wide acceptance in the parallel computing community. The Stanford DASH [LLG+92], FLASH [JK+94], the KSR1[1] [KSR91], Toronto's Hector [VSWL91], NUMAchine [V+], and the Cray T3D [Gmb93] are some of the NUMA machines currently in use or under development. The shared memory in a NUMA multiprocessor is physically distributed among the processors in modules as shown in Figure 1. The memory modules are connected by a scalable low latency interconnection network, such as a mesh (DASH, FLASH), a hierarchical ring (KSR1, Hector, NUMAchine), or a Torus (T3D) to allow a processor to access any memory module. Memory access latency is non-uniform and is proportional to the distance between processor and memory. Typical access latencies

[1] Although, the KSR1 is a Cache Only Memory Architecture (COMA), we include it here among the NUMA machines because of the hierarchical nature of the latencies.

Architecture	Cache	Local Memory	Remote Memory
Hector	1	10	19/29
DASH	1	29	132
KSR1	2	18	170
CM5	1	10-20	2560[2]

Table 1: Latency for memory read operation in processor cycles

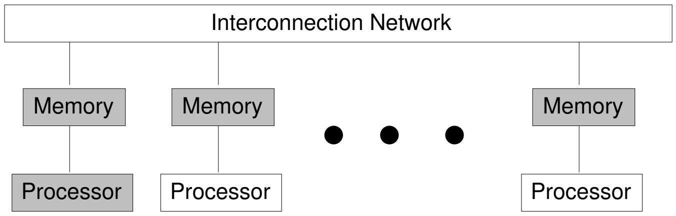

Figure 1: NUMA Multiprocessor

(in processor cycles) for some NUMA multiprocessors are shown in Table 1. Remote memory access latency is 3-10 times the local memory access latency. Nevertheless, in comparison to distributed memory multiprocessors, remote memory access time is relatively small in NUMA multiprocessors; Table 1 shows that remote memory access latency for the CM5 distributed memory multiprocessor is two orders of magnitude higher than local memory access latency.

NUMA multiprocessors support a shared memory programming paradigm but require application programmers to enhance locality of reference in their programs. The relatively high cost of remote memory accesses makes it imperative to manage data so that most accesses by a processor are made to the processor's local portion of shared memory. Since application programmers do not expect to be concerned with the management of data in the shared memory programming paradigm, data management tasks have typically been delegated to the operating system, as part of its page management policies [LWE91]. However, the granularity of a page is too coarse to exploit the fine grain sharing that exists in the parallel applications. Furthermore, operating system policies are designed to be generic across a wide variety of application char-

[2] Calculated based on the elapsed time for a message of 128 bytes Send-Reply(Normalized) [KTR93]

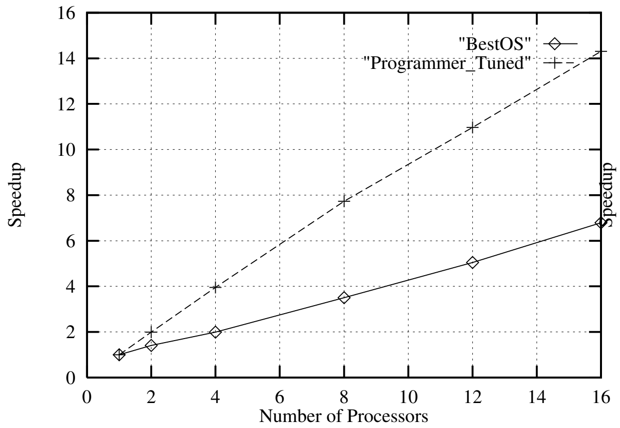

Figure 2. Jacobi SOR

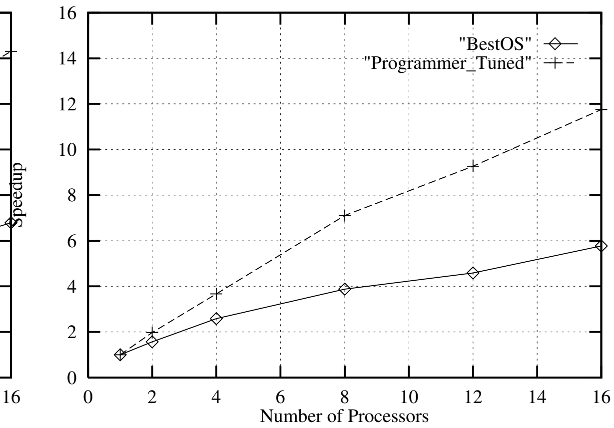

Figure 3. LU Factorization

acteristics. Therefore, the operating system is impervious to the semantics and the data access behavior of a particular application. Figure 2 and 3 compare the speedups obtained on the Hector multiprocessor for two applications, $Jacobi\ SOR$ and $LU\ Factorization.$ It is evident that programmer-tuned data management performs much better compared to operating system based data management. However, the tuning of data management tasks for a specific application is tedious and requires considerable effort on the part of programmers.

We believe that data distributions are good abstractions to specify the locality of data in data-parallel applications [LS93, AW94]. The notion of data distributions has proved successful on distributed memory multiprocessors (DMM)[3], as evident from the large number of compilers and/or environments build around this notion. The programmer specifies a data distribution *scheme* for each array in the data-parallel program by assigning a partitioning attribute and processors to each dimension of the array. A compiler then uses these data distribution schemes to distribute arrays across the memory modules of the processors, to partition computation, and to generate communication primitives. The selection of data distribution schemes in a program is crucial to the performance of the application. In general, on DMM data distribution schemes are selected to minimize interprocessor communication, which is the overwhelming factor affecting performance

[3] Message Passing Machines

on DMM.

In this paper, we argue that interprocessor communication is not the only factor that must be taken into consideration when selecting data distribution schemes on NUMA multiprocessors. Although minimizing communication between processors is important, issues such as contention, false sharing and cache affinity have equal (and at times larger) impact on performance. In addition, the interaction of these issues result in subtle tradeoffs that must be taken into account in selecting data distribution schemes in an application. Existing methodologies for selecting data distributions that only minimize interprocessor communications are tailored for DMM and hence are not suitable for NUMA multiprocessors. In fact, a suitable distribution scheme for an application on a DMM can be the least suitable distribution scheme for the same application on a NUMA multiprocessor.

The remainder of this paper is organized as follows. Section 2 presents an overview of the issues that impact on performance of data-parallel applications on NUMA multiprocessors. In Section 3, we present experimental results that show that the choice of a data distribution scheme is strongly influenced by all these performance factors, and not only by the cost of interprocessor communication. Finally, in Section 4, we present concluding remarks and directions for our future work.

2 Performance Issues

The choice of a data distribution scheme on a distributed memory multiprocessor is primarily motivated by minimizing the cost of interprocessor communication. This typically implies the reduction of the number of messages transferred between processors [AL93, BFKK91], or the reduction of the number of processors to communicate with [GB92]. The choice of a data distribution scheme on a NUMA multiprocessor must in addition be motivated by a number of other factors that affect performance. In this section we describe these factors.

1. **Remote Memory Accesses**. The performance of a parallel program is directly affected by the number of remote memory accesses performed by the processors. Reducing the number of remote memory accesses is one of the strategies for improving performance on a NUMA multiprocessor. The impact of remote memory accesses most significant when the local cache of a processor is large enough to hold the working set of the program. In this case processors make most of their accesses from their local caches and only stall when making remote memory accesses.

2. **Cache Affinity**. On NUMA multiprocessors, the behavior of caches plays a key role in performance. In particular, the problem of cache interference is significant in NUMA multiprocessors because some of the memory accesses that result from interference misses will be to remote memory modules. In contrast, all cache misses on DMM result in local accesses only. Hence it is more critical to consider the impact of caches when selecting an appropriate data distribution.

3. **Memory Contention**. Memory contention occurs when many processors access a single memory module at the same time, which increases response time for satisfying memory requests and degrades program performance. Memory contention can be alleviated or minimized by using data replication, appropriate data distributions and program restructuring. For example, when executing a reduction operation, temporary copies can be used to minimize contention. Contention caused by scalars or small vectors can be generally alleviated using array privatization techniques [MAL93]. In contrast, DMM use a sender initiated communication protocol which is effective in reducing memory and network in high traffic situations, making contention less of a factor. The selection of data distribution schemes on NUMA multiprocessors must take memory contention into consideration.

4. **False Sharing**. False sharing can occur at the cache line level on NUMA multiprocessors with hardware support for cache coherence, and at larger granularities on NUMA multiprocessors which use software to manage caches [LS94]. False sharing causes extensive invalidation traffic, or can result

in uncachable data. Although false sharing is not an issue that arises in DMM, it must be addressed when deciding data distributions on a NUMA multiprocessor. A boundary-conscious approach to data distribution is required to reduce or eliminate false sharing.

3 NUMA Conscious Data Distributions

In this section, we use a number of applications to demonstrate the significance of the factors that affect performance on NUMA multiprocessors. We use the Hector multiprocessor as a platform to empirically show the impact of these factors. The KSR1 system is used to show the impact of cache affinity and false sharing because of its larger cache size which allowed us to experiment with larger data sets. Data distributions were implemented using the techniques described in [LS93, AW94].

3.1 Cache Affinity

We use the Multigrid (*mg*) application from the NAS suite of benchmarks to illustrate that the data distributions must be cache conscious. *Mg* is a three dimensional solver calculating the potential field on a cubical grid. We focus on the subroutine $psinv$ which uses two three dimensional arrays U and R. The subroutine mainly performs the following computation inside a triply nested loop: $U(i,\, j,\, k) += \; h(R(f(i),\, g(j),\, h(k)))$, where $f(i)$ = $i-1$, i or $i+1$, as are the functions g and h. The loop nest is fully parallel.

The application has nearest neighbor communications along all three dimensions, which is typical of many scientific applications. Block distributions are used only along the first two dimensions of the arrays since it results in abundant parallelism. The most appropriate distribution scheme on a distributed memory multiprocessor assigns an equal number of processors to each distributed dimension of the arrays, which is suggested in the Syracuse High Performance Fortran applications suite [4].

[4] http://www.npac.syr.edu/

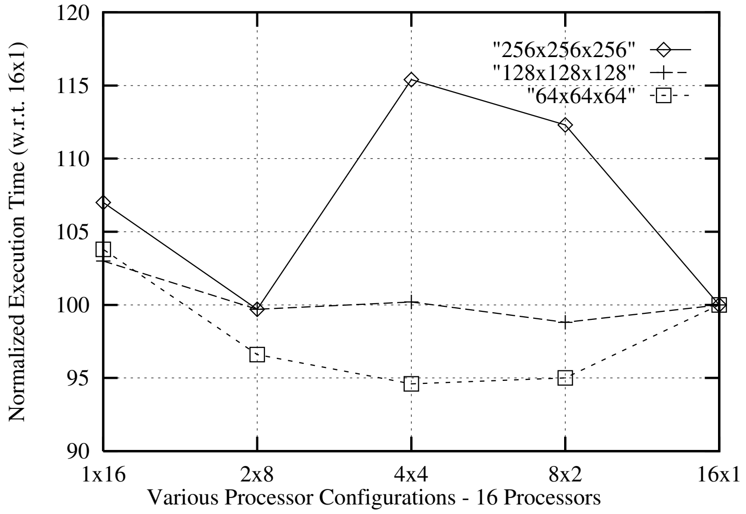

Figure 4: Effect of Cache Interference on Data Distribution

The impact of cache affinity can be easily seen from the Figure 4, which shows the execution time of the application for a number of data distributions and a number of array sizes. A line graph is used to highlight the performance trends caused by changing data distributions. When the size of the arrays is relatively small (64x64x64), data used by a processor fits in the local cache of the processor. Consequently, the predominant factor affecting performance is remote memory accesses. The best distribution scheme is 4x4, which is the same as on a distributed memory multiprocessor. When the size of the arrays is relatively large (256x256x256), data used by a processor no longer fits in the cache of the processor. A large number of cache misses is generated, which must be satisfied from remote memory. The 2x8 distribution scheme minimizes communication while maximizing cache resue, and results in the best overall performance. The 4x4 scheme results in the worst performance. These results are conservative since on the KSR1 COMA memory system real-only accesses replicate data in the local memory of a processor. Subsequent cache misses are satisfied from local memory. Hence, we see a reduced impact of cache misses.

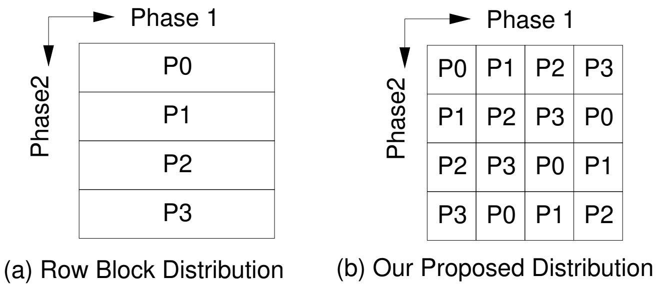

Figure 5: Data Distribution used to alleviate Memory Contention

3.2 Memory Contention

The Altering Direction Integration(ADI) program has two phases with parallelism along orthogonal dimensions in each phase. It operates on three 2-dimensional arrays A, B and X. A single iteration of an outer sequentially iterated loop consists of a forward and a backward sweep phase along the rows of the X and B arrays, followed by another forward and backward sweep phase along the columns of the same two arrays [Kre93]. This application is typical of other programs such as 2D-FFT and Erlebacher which have parallelism in orthogonal directions in different phases of the program.

The best data distribution scheme for ADI remains an issue of debate [Kre93, AALL93]. The two proposed schemes partition arrays along a single dimension, either in blocks or cyclically. On a NUMA multiprocessor, this type of partition leads to remote-access-free parallelism in the first phase of the program, but results in heavy contention in the second phase. This is illustrated using Figure 5(a), which shows a block distribution across the rows of the arrays. In the first phase of the program, there are no remote memory accesses since all data needed by a processor is local to the processor. However, in the second phase of the program, all processors start to sweep across the columns, and hence, all access data in the local memory of processor 0, and then data in the local memory of processor 1, and so on. This access pattern creates contention. To reduce this contention, we propose the following data distribution scheme:

```
!HPF$ PROCESSORS PROCS(N)
```

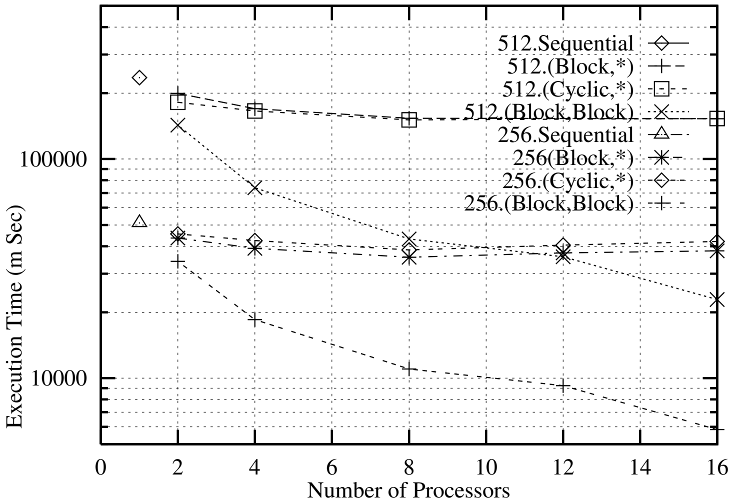

Figure 6: Effect of Data Distribution on Memory Contention

```
!HPF$ DISTRIBUTE B(BLOCK, BLOCK), X(BLOCK, BLOCK) ON PROCS.
```

This scheme is depicted in Figure 5(b) for $N = 4$. In the first phase, each processor sweeps across rows of the arrays, accessing data in local as well as in remote memory modules. In the second phase, processors sweep across the columns of the arrays, each accessing data in a separate memory module, effectively eliminating contention. This is achieved at the expense of an increase in the number of remote memory accesses in the first phase of the application.

The performance of these data distribution schemes is shown in Figure 6. Block and cyclic distributions across one dimension of the arrays clearly suffer in their performance. Our proposed distribution schemes results in significantly better performance because of its effective tradeoff between remote memory accesses and contention.

3.3 Cache line False Sharing

The programs *tred2* (which is part of Eispack), *mdg*, and *trfd* (which are both part of the Perfect Club Benchmark Suite) exhibit parallelism which results in considerable false sharing. These programs have triangular

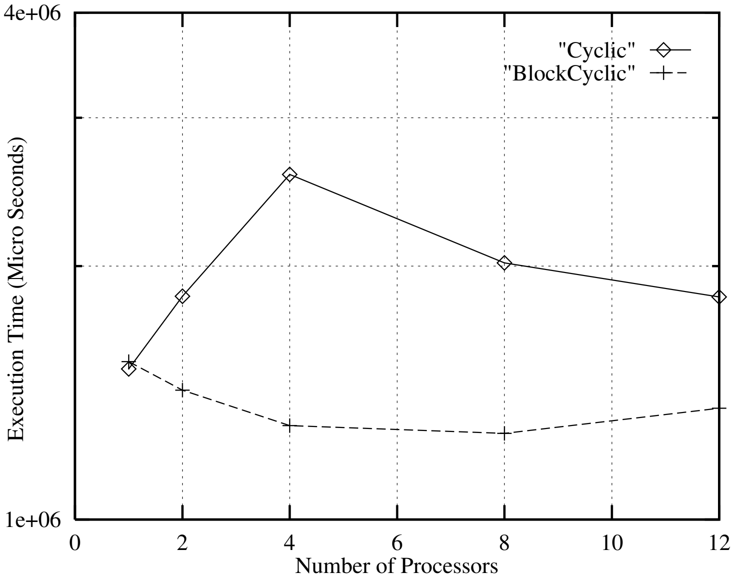

Figure 7: Effect of Data Distribution on False Sharing

iteration spaces which suggest cyclical distribution for load balancing. The choice of this distribution and the storage order of the arrays cause more than one processor to share the same cache line. The impact of this false sharing is shown in Figure 7 for the *tred2* application on the KSR1 multiprocessor, using cyclic and block-cyclic distributions. The use of the cyclic distribution results in heavy invalidation overhead affecting the performance, causing execute time to increase as the number of processors increase. To minimize the effect of false sharing, the arrays are distributed using a block-cyclic distribution, where the size of the block is equal to the size of the cache line. This effectively eliminates false sharing and maintains load balance. This can be seen in Figure 7; the execution time of the application is reduced. However, when the size of the cache line becomes large relative to the size of the arrays, a tradeoff between the cost of load imbalance and the cost false sharing becomes apparent. This can also be seen from the figure; when the number of processors increases, load imbalance becomes significant and execution time increases.

3.4 Remote Memory Access

Many applications such as Matrix Multiply, LU factorization, Jacobi SOR, $flo52$(which is part of the Perfect Club Benchmarks) and programs which have sufficient parallelism along a single dimension, would use the same distribution for a NUMA and DMM system, because the cost of remote memory accesses dominates performance.

4 Concluding Remarks

Although large scale shared memory NUMA multiprocessors are built based on an architecture with distributed memory, the shared memory paradigm introduces performance issues which are different from those encountered in DMM. The high cost of interprocessor communication in distributed memory multiprocessors makes the minimization of communication the predominant issue in selecting data distributions. In the case of NUMA multiprocessors issues such as contention, false sharing and cache affinity cannot be ignored. A methodology for selecting data distributions that considers only the cost of interprocessor communication will not result in adequate performance on NUMA multiprocessors. Such a methodology should also consider the additional issues, as well as interaction and the tradeoffs among them. In this paper, we have presented empirical measurements to show how these issues significantly affect performance, and show that using data distributions that simply minimizes interprocessor communication result in poor performance. Table 2 summarizes the results of the study.

Tools and compilers that attempt to automatically determine data distributions on NUMA multiprocessors must consider all the above issues. We are currently developing a framework for deriving automatic data distributions on NUMA systems. We are also working on deriving a cost estimator to estimate the effectiveness of one data distribution over the other in the presence of the issues we identified in this paper.

Application	Dominant Factor	DMM Distribution	NUMA Distribution
Multigrid	Cache Affinity	(Block(4), Block(4))	(Block(2), (Block(8))
ADI	Memory Contention	(Block, *)	(Block, Block)
		(Cyclic, *)	
Tred2	False Sharing	(Cyclic)	(BlockCyclic)
LU Decomp.	Remote Access	(Cyclic, *)	(Cyclic, *)
Jacobi SOR	Remote Access	(Block, *)	(Block, *)

Table 2: Applications and the dominant factor the affect performance

References

[AALL93] Saman P. Amarasinghe, Jennifer M. Anderson, Monica S. Lam, and Amy W. Lim. An overview of a compiler for scalable parallel machines. In *Languages and Compilers for Parallel Computing*. Springer-Verlag, 1993.

[AL93] Jennifer M. Anderson and Monica S. Lam. Global Optimization for Parallelism and Locality on Scalable Parallel Machines. In *Proceedings of the SIGPLAN'93 Conference on Programming Language Design and Implementation*, pages 112–125, June 1993.

[AW94] Tarek S. Abdelrahman and Thomas N. Wong. Distributed array data management on NUMA multiprocessors. In *Proceedings of SHPCC*, pages 551–559, May 1994.

[BFKK91] V. Balasundaram, G. Fox, K. Kennedy, and U. Kremer. A static performance estimator to guide data partitioning decisions. In *Proc. of 3rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Williamsburg, Virginia, April 21-24 1991.

[GB92] Manish Gupta and Prithviraj Banerjee. Demonstration of automatic data partitioning techiniques for parallelizing compilers on multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):179–193, March 1992.

[Gmb93] Cray Research GmbH. *The Cray Research Massively Parallel Processor System - Cray T3D*. 80922 Munchen, Germany, 1993.

[JK+94] M. Heinrich J. Kuskin, D. Ofelt et al. The Stanford FLASH Multiprocessor. In *Proc. of the 21st Annual International Symposium on Computer Architecture*, pages 302–313, 1994.

[Kre93] Ulrich Kremer. Automatic data layout for distributed-memory multiprocessors. Technical report, Center for Research on Parallel Computation, February 1993.

[KSR91] Kendall Square Research KSR. *KSR1 Principles of Operation*. Waltham, MA, 1991.

[KTR93] Thomas T. Kwan, Brian K. Totty, and Daniel A. Reed. Communication and computation performance of the CM5. In *Supercomputing'93*, pages 192–201. ACM, November 1993.

[LLG+92] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica S. Lam. The Stanford DASH multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[LS93] Hui Li and Kenneth C. Sevcik. Numacros: Data parallel programming on NUMA multiprocessors. In *Proceedings of Fourth Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV)*, pages 247–263, September 1993.

[LS94] Hui Li and Kenneth C. Sevcik. Exploiting Cache Affinity in Software Cache Coherence. In *Proceedings of the 8th ACM International Conference on Supercomputing*, pages 264–273, July 1994.

[LWE91] Richard P. LaRowe Jr., James T. Wilkes, and Carla Schlatter Ellis. Exploiting Operating System Support for Dynamic Page Placement on a NUMA Shared Memory Multiprocessor. In *Proc. of 3rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Williamsburg, Virginia, April 21-24 1991.

[MAL93] D. E. Maydan, S. P. Amarasinghe, and M. S. Lam. Array data flow analysis and its use in array privatization. In *Proc. of 20th ACM Symposium on Principles of Programming Languages*, January 1993.

[V^+] Zvanco G. Vranesic et al. The NUMAchine Multiprocessor. Submitted for publication.

[VSWL91] Zvonko G. Vranesic, Michael Stumm, Ron White, and David Lewis. The Hector Multiprocessor. *Computer*, 24(1), January 1991.