The NUMAchine Multiprocessor

R. Grindley, T. Abdelrahman, S. Brown, S. Caranci, D. DeVries, B. Gamsa, A. Grbic, M. Gusat, R. Ho, O. Krieger, G. Lemieux, K. Loveless, N. Manjikian, P. McHardy, S. Srbljic, M. Stumm, Z. Vranesic and Z. Zilic
Department of Electrical and Computer Engineering
University of Toronto, Canada
zvonko@eecg.utoronto.ca

Abstract

Small-scale multiprocessors are becoming increasingly economical and common, whereas larger multiprocessors continue to have higher per-node costs. The NUMAchine multiprocessor project seeks to make large-scale multiprocessors more economical while maintaining high performance by exploring architectural and hardware features for low-cost, modular multiprocessors. To demonstrate our approach, we have implemented a prototype system that is scalable to 128 processors. An efficient directory-based cache coherence protocol exploits our hierarchical ring-based interconnect and supports sequential consistency. This paper documents the design choices and the resulting performance of the system using both simulation results and measurements on the prototype hardware.

1. Introduction

For multiprocessors to gain widespread use, economic considerations must play a role in their design. Currently, systems with 8 processors are available as commodity parts, and in the next decade it is possible that the size of commodity multiprocessors may reach 16 or 32 processors. For this to happen, the overall cost of these multiprocessors must be kept low and the architecture must be sufficiently modular so that small systems can be easily and affordably extended into larger ones. While much of the multiprocessor research to date has focussed on performance, we have extended our research to examine architectural and hardware features for low-cost modular multiprocessors while maintaining good performance and scalability.

The NUMAchine multiprocessor[1] has been developed at the University of Toronto as part of a research project on hardware and software for parallel computing. It is a CC-NUMA distributed shared-memory (DSM) multiprocessor designed for cost-effective performance. Processors, caches, and memory are distributed across a number of *stations* interconnected by a hierarchy of unidirectional bit-parallel rings. The simplicity of the interconnection network permits the use of wide communication paths at each node and a novel scheme for routing packets between stations enables high-speed operation of the rings. A directory-based hardware cache coherence protocol is used. The protocol scales efficiently with system size by exploiting the hierarchical architecture and the natural ordering properties of the rings; it implements a sequentially-consistent memory model. A prototype machine has been constructed to serve as a useful test-bed for further hardware and software research. All of the control circuitry is implemented in programmable logic, making it possible to add or change functionality at the hardware level without requiring physical modifications to printed-circuit boards.

The primary research contributions of the NUMAchine hardware implementation are: i) proof-of-concept for a simple, low-cost multiprocessor architecture based on hierarchical rings and ii) an efficient hardware cache coherence protocol that exploits the architecture's multicast capabilities. The paper is organized as follows: Section 1.1 identifies related commercial and experimental multiprocessors. Section 2 describes the NUMAchine architecture and prototype details. Major architectural and design choices are discussed in Section 3. Prototype performance results are presented in Section 4 and the current status of the prototype is given in Section 5. Section 6 concludes the paper.

1.1 Background

A variety of large-scale multiprocessor architectures has been developed, as indicated in Table 1. The relevant features considered here are: type of clustering, type of interconnect, presence of caches for remote data, and the

[1]This work was supported by the Strategic Grant #STR0149404 from the Natural Sciences and Engineering Research Council of Canada

Table 1. Some commercial and experimental multiprocessors

Name	Cluster	Interconnect	Features
DASH [14]	bus	mesh	remote access cache
FLASH [7]	non-clustered	mesh	programmable protocol processor, page replication/migration
Origin2000 [13]	paired-processors	cube	page replication/migration
I-ACOMA [20]	bus	mesh	simultaneous multithreading, cache-only memory architecture
Teracomputer [19]	non-clustered	multistage switch	multithreaded execution, no caching or data replication
Starfire [1]	bus	multiple buses	global snooping, crossbar for responses
V-class [8]	crossbar	toroidal ring	remote data caches
KSR1 [12]	non-clustered	hierarchical rings	cache-only memory architecture
NUMA-Q [16]	bus	ring	remote data caches

choice between non-uniform memory (NUMA) or cache-only memory (COMA) architectures. Clustering processors together is a means of leveraging commodity symmetric multiprocessor (SMP) nodes. There are a number of possibilities for the system-wide interconnect including meshes, multistage switch networks, and rings. Each has advantages and disadvantages in terms of performance, complexity, and cost. Some systems include caches for remote data to mitigate longer memory access latencies as the system size increases. Finally, some systems employ a cache-only architecture (COMA) to automatically replicate and migrate data in hardware, rather than rely on caching with home memory locations as in NUMA systems. The systems listed in Table 1 use NUMA architecture, unless otherwise stated. The variety of architectures in Table 1 suggests that there is no single best approach when engineering such systems.

The NUMAchine multiprocessor employs clustering, uses a hierarchical ring-based interconnect, provides a remote data cache, and uses a NUMA architecture. It is therefore more similar to the last three systems in Table 1. Nonetheless, this is an introduction to NUMAchine, and comparison to other systems will be left to future papers.

2. NUMAchine Architecture and Prototype

The NUMAchine architecture permits modular system construction which can affordably scale from a small to a large number of processors. Architecture and prototype details are described below.

2.1 Architecture Overview

NUMAchine consists of a number of *stations* connected by a two-level hierarchy composed of *Local Rings* and a *Central Ring*, as shown in Figure 1. The ring hierarchy is joined by an *Inter-Ring Interface*. Data transfers across rings are divided into *packets* which are sent according to a slotted ring protocol. Routing packets on the NUMAchine ring is simple and fast: each station checks a single bit to determine if a packet has reached its destination.

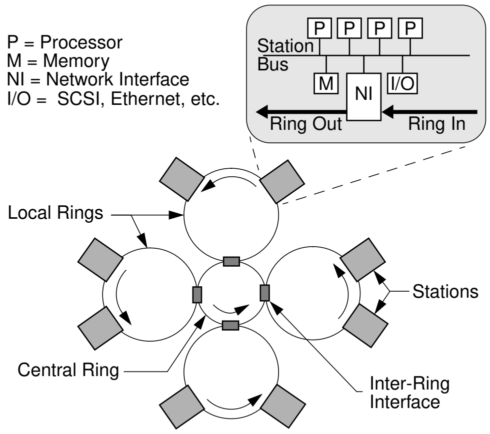

Figure 1. NUMAchine architecture

Unidirectional slotted rings were chosen for a number of reasons. First, they can perform as well as meshes for up to 128 processors [17, 18] when some data locality is present. Second, stations can be added one at a time without significant re-wiring or topology changes, making them highly modular and cost-effective. Third, rings exhibit two features useful for implementing cache coherence and memory consistency: inherent sequencing of requests/responses, and natural broadcast capabilities. For example, a single request invalidates multiple copies of a cache line as it traverses the ring hierarchy, and serves as an acknowledgment upon its return to the source of the request.

Each station in NUMAchine consists of a bus connecting a number of processors, a memory module, an I/O module, and a *network interface* that connects to a Local Ring. The physical memory is distributed among the stations such that each memory address has a fixed *home station*. Local memory accesses within a station incur only bus interconnect latency, whereas *remote* memory accesses incur additional ring interconnect latency to access addresses whose home location is on another station. The network interface also contains a *network cache* (NC) that reduces the average latency for remote data accesses.

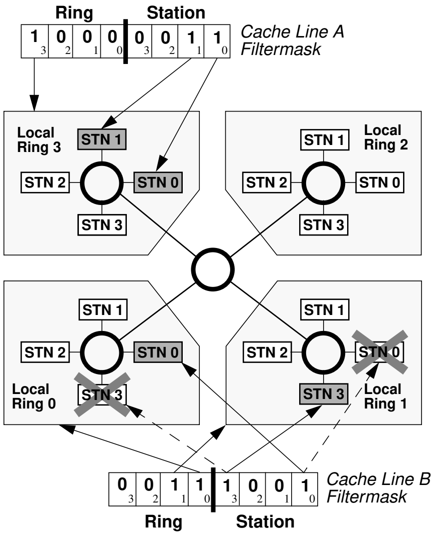

Figure 2. Filtermask examples

2.2 Cache Coherence

NUMAchine uses a directory-based, write-back/invalidate cache coherence protocol. A detailed description of the cache coherence protocol can be found in [4]. Directories are maintained at two levels: the memory level and the network level.

Memory Level: The memory directories maintain the system-wide coherence state of each cache-line-sized memory block, and identify cached locations using two small bitmasks. The *processor mask* identifies processors on the local station with a copy of a cache line. The *filtermask* uses separate ring and station fields to identify remote stations which also contain a copy. The name of this mask reflects its intended role in limiting (filtering) the destinations for invalidation requests. Multiple remote locations are represented by OR-ing their respective filtermasks into a single filtermask to reduce the number of bits in the directory. Although the resulting filtermask may occasionally overspecify the true caching stations, this merely causes some invalidations to be sent where none are needed. The number is typically small because most cache lines are shared by a small number of processors.

Two filtermask examples are given in Figure 2. Copies of cache line A are located on stations 0 and 1 of Local Ring 3, and the resulting filtermask formed by OR-ing this information precisely indicates the locations of copies of line A. Copies of cache line B are located on station 0 of Local Ring 0 and station 3 of Local Ring 1. When a single filtermask is formed for these locations, it *overspecifies* the locations of cache line B. It appears that two additional stations have copies (station 3 on Local Ring 0 and station 0 on Local Ring 1) when in reality they do not. Such overspecification only occurs when copies are on different rings and on different stations on those rings.

Network Level: The network cache also maintains a directory with a processor mask for each cache line present in the cache. This mask specifies the local processors with copies of cache lines whose home memories are on other stations. Typically, only the processors listed in the processor mask receive invalidation messages sent for this cache line. However, a cache line in the network cache may be replaced by another without informing the home memory and without invalidating copies in the processor caches on this station. If an invalidation is received for such a cache line, the worst case must be assumed and all processors on the station must be sent the invalidation. A relatively large network cache makes this an infrequent occurrence.

Coherence States: The memory directories maintain four coherence states for each cache line:

Local Valid (LV)	One or more processors caches within the station have a shared copy. Remote stations do not have valid copies.
Local Invalid (LI)	One local processor cache has a modified copy. There are no other valid copies.
Global Valid (GV)	One or more remote stations have a shared copy, and there may also be local copies in processor caches within the station.
Global Invalid (GI)	One remote station has a modified copy, and there are no other valid copies.

A line in any one of the four states can also be locked. Locking of the line occurs at the beginning of a coherence action that requires multiple stages, and ensures that no other access to the line is possible until the transaction completes.

State information contained in the network cache complements the information in the home memory. The set of states in the network cache (GV, GI, LV, LI) is similar to that in main memory. One difference is that the Local states, LI and LV, indicate the station has a modified copy; hence, the same line is in the GI state at the home memory. Upon receiving a request for the cache line, the home memory must forward it to the station with the modified copy. Data responses are sent both to the home memory and the requester. Another difference is that the GI state only means that no valid copies of the cache line exist on the station.

2.3 Memory Consistency

NUMAchine supports sequential consistency, which is the most intuitive model for writing shared-memory programs. As well, since modern high-performance processors gain little additional performance from relaxed consistency schemes, there is little incentive for implementing these less-intuitive programming models [9]. Although providing sequential consistency may involve additional hardware cost in some architectures , the NUMAchine architecture inherently provides simple and efficient means for supporting it. Fixed sequencing points are defined on both the Local and Central Rings. A multicast (limited broadcast) invalidation does not become active until it passes the sequencing point on the highest ring level that must be traversed to reach all multicast destinations. The simple routing for the rings makes the activation decision trivial. This imposes the necessary ordering for sequential consistency between any two multicasts on a given ring level, although the average traversal length for sequenced packets (i.e. invalidations) is slightly increased.

2.4 Flow Control, Deadlock Avoidance, and Retry

To prevent loss of data due to buffer overflow, NUMAchine uses a flow control scheme that signals nearby senders to stop issuing packets if the receiving buffer is nearing capacity. No timeouts or negative acknowledgments are required in this approach, simplifying the implementation. Buffers with large capacities are relatively inexpensive and provide adequate space for in-transit messages when the flow control is invoked.

NUMAchine avoids deadlock arising from circular dependence amongst resources by distinguishing between *sinkable* packets (those that will not generate new requests such as data response) and *non-sinkable* packets (such as requests). Sinkable responses are permitted to bypass non-sinkable requests in certain cases that would otherwise lead to deadlock. Details of deadlock avoidance and flow control are documented in [15].

NUMAchine provides a retry mechanism for requests that are negatively-acknowledged when they encounter a locked state in a directory. This approach avoids the need to buffer an arbitrary number of such requests at the locked site. Instead, the originator of the request employs a modified binary exponential back-off retry algorithm.

2.5 Prototype Implementation

The cards on a NUMAchine station are shown in Figure 3. Each station can hold four processor cards, two memory cards, two I/O cards, and one network interface card. The station bus is a 64-bit, split-transaction, address/data multiplexed bus running at 50 MHz, for a peak bandwidth of 400 Mbytes/sec. The physical bus is based on the Futurebus+ standard, but NUMAchine uses a novel synchronous bus protocol with decentralized busy detection and centralized arbitration [15].

Each processor card has a 150 MHz MIPS R4400 microprocessor with an external 1-Mbyte secondary cache. The secondary cache line size is boot-time selectable between 64 bytes and 128 bytes.

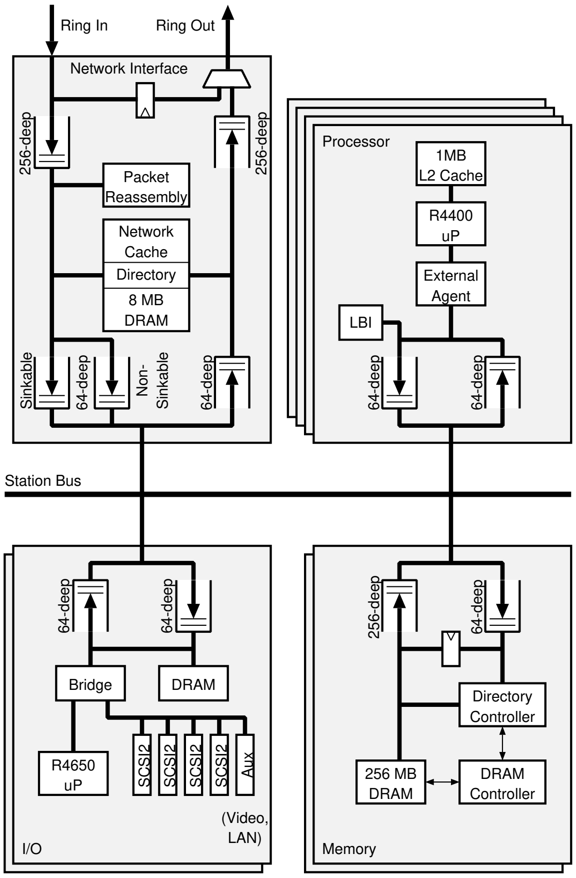

Figure 3. Cards on a NUMAchine station

The memory card contains up to 256 Mbytes of 4-way interleaved DRAM that can feed the bus at the peak bandwidth of 400 Mbytes/sec. The memory card also hosts a directory controller that uses SRAM memory to maintain the state of each cache-line-sized DRAM memory block on the board. Finally, the memory card includes a controller for special functions such as block memory-to-memory transfers and block coherence operations.

The I/O card provides access to local disk storage, as well as a PCI interface for a commodity LAN adapter or video card. The I/O subsystem supports disks distributed among four SCSI-2 controllers.

The network interface card provides the connection between the station bus and the Local Ring. The incoming and outgoing paths are separate and each is 64 bits wide. It also contains an 8-Mbyte network cache for remote data. The controllers on the network interface card are responsible for managing the network cache as well as the packaging and reassembly of cache lines to and from the ring hierarchy.

The connection between each Local Ring and the Central Ring is through an Inter-Ring Interface (IRI) consisting of FIFO buffers for the upward and downward paths, along with the associated controllers. For a 64-processor machine consisting of four Local Rings, four IRIs are necessary. Our implementation places all four IRIs on a single board with daughter cards. Further implementation details of the prototype have been reported in [5].

3. Rationale for Design Choices

Simulations of SPLASH-2 benchmark programs were used to analyze the performance impact of the rings, queue sizes, filtermasks, network caches, and coherence protocol overhead. These results are described below and further details can be found in [6].

3.1 Simulator

The NUMAchine simulator uses MINT [21] as its front-end. The back-end models NUMAchine's memory system, generating appropriate delays when requests are passed through caches, buses, rings, etc. When modeling a complex system such as a multiprocessor, the trade-off between model accuracy and simulation time can be significant. The NUMAchine simulator attempts to efficiently capture all the salient details of data transfers and protocols.

All queues are modeled accurately, providing good indication of occupancy and congestion. Operating system calls take zero time as seen from the virtual processor. Page fault overhead is not modeled and a round-robin page placement policy is used.[2] The default behavior is to model only the parallel section of a program. When skipping over the sequential code, the simulator correctly executes instructions, but allows all loads and stores to succeed immediately, bypassing the cache and without doing any page mapping.

3.2 Ring Performance

This section shows that the design of the ring hierarchy and flow control prevent saturation of the network.

Ring Utilization: In a single ring clock cycle, a ring slot may depart a ring interface in either an empty or utilized state. An empty slot departs a ring interface when there is no new data to inject into the ring and either *i)* an incoming packet terminates its traversal at the interface, or *ii)* an empty slot was received. Utilized ring slots may be categorized into one of the following three states:

Send	A new packet is injected into an outgoing ring slot.
Forward	A received packet is passed on to the next ring link.
Split	A received packet is passed on, and a local copy is passed down.

[2]The results are somewhat pessimistic since all pages, private and shared, are currently allocated using the round-robin policy.

Ring utilization is obtained by averaging the utilized slots over all the ring's interfaces.

Figure 4 shows utilizations of the Central and Local Rings. The large utilizations (10%–40%) for FFT, Ocean and Radix arise from a higher communication-to-computation ratio, as described in [22]. In contrast, the other benchmarks exhibit markedly lower utilizations.

For all benchmarks running on 64 processors, Central Ring utilization is higher than the Local Ring, as expected. For 32 processors (only two Local Rings) the situation is reversed, indicating that plenty of additional bandwidth is available on the Global Ring.

Ring Queue Depths: To obtain greater insight into ring performance, we measured the maximum and average depths of the incoming and outgoing queues in the network interfaces. The same queue sizes as the prototype were used (512 entries for Central, and 256 for Local), with flow control being invoked when a queue reaches 75% of capacity. The results are presented in Figure 5.

Most of the benchmarks have low average queue depths, and the maximum queue depths are usually below the threshold for flow control invocation. Hence, the rings are able to handle the amount of traffic generated by these benchmarks and the flow control mechanism is not invoked. Note that moderately-sized commodity buffers are sufficient to provide this performance.

Only three of the benchmarks actually achieve queue depths that cause flow control to be invoked: Barnes, Ocean, and Radix. For Barnes and Ocean, the impact of the flow control is minimal, since Central Ring queue depths exceed the threshold only a small number of times. Furthermore, this does not significantly impact the Local Ring queues, since their thresholds are not reached.

Only Radix invokes the flow control mechanism a significant number of times. Because of this, average and maximum depths for the Local Ring queues also increase (since they are frequently blocked from injecting packets into the Central Ring) and can potentially saturate. Despite this, the Central Ring utilizations, while higher than for most of the other benchmarks, do not exceed about 40%.

Interestingly, Barnes exhibits a large difference between its maximum and average queue depths and its ring utilizations are low (less than 10%). Thus, this application has a few short but heavy bursts of communication. In contrast, FFT exhibits higher utilizations (5%–40%), but has lower maximum queue depths, indicating higher volume communication that is more evenly distributed in time.

Central Ring Speed: The raw speed of the interconnection network has an effect on performance in any multiprocessor system. As shown in Figure 5, the ring-injection queues into the Central Ring can become long. Increasing

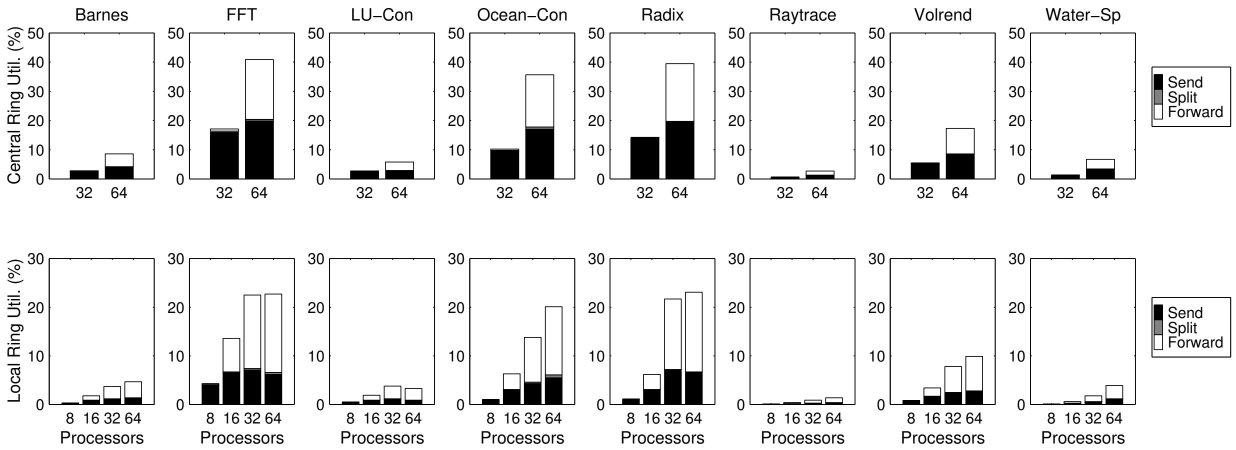

Figure 4. Central and Local Ring utilizations

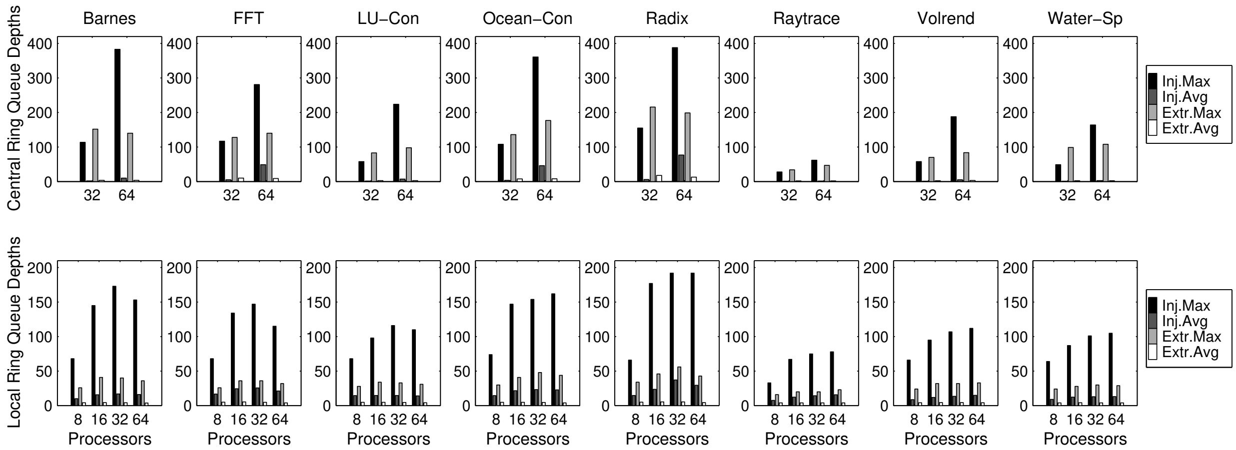

Figure 5. Central and Local Ring queue depths

the Central Ring speed should help empty out the queues, while increasing the rate of requests being injected into the lower levels of the hierarchy. The key point is that the two levels of hierarchy must be well-balanced. We simulated speeds from 50 to 250 MHz, in 50 MHz steps. The performance increased by 11% at 100 MHz, and then flattened out. Higher Central Ring speeds put pressure on the IRI extraction queues, and the limiting factor becomes the rate at which packets can be transferred to the station. For the prototype, we determined that 50 MHz is sufficient, but the Central Ring can run at any speed up to 80 MHz.

3.3 Filtermask Performance

As the number of stations in the system increases, the probability of the filtermask overspecifying stations increases. Since the filtermask can uniquely identify each station, the only concern is in overspecifying targets of multicasts, specifically invalidations. This imprecision in the filtermask can be measured by keeping track of the exact number of sharers in the memory coherence directory in the simulator. When an invalidation is sent out, the actual number of stations targeted is divided by the real number of sharers to obtain the *overinvalidation rate*. For the simple case of two sharers, the overinvalidation rate can be as high as two if the sharers are on different rings and stations. (Note that if the sharers are on the same station or the same Local Ring then the overinvalidation rate is one, i.e., the filtermask is precise.) We expect the rate to be around two if the number of sharers on average is two, as we did not tune the SPLASH-2 programs to take advantage of locality.

Figure 6 shows the overinvalidation rate averaged for all invalidations. The rates can reach as high as 3.3, but most are around 2.5 when using 64 processors. This amount of multicast traffic does not impair the machine performance, because on average they do not generate broadcast inval-

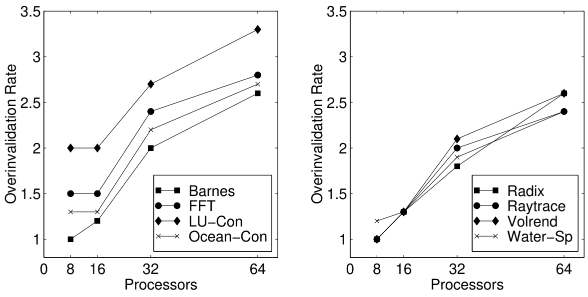

Figure 6. Overinvalidation rates

idations to all stations. Avoiding heavy broadcast traffic is important in maintaining system scalability, as shown in [2].

3.4 Network Cache Performance

The most fundamental metric for measuring Network Cache performance is the *hit rate*. A request for a remote cache line is considered a hit in the NC if it does not generate any network traffic (though it will generate local on-station traffic). The simplest such case occurs when data fetched during a remote shared read from one processor can be used to satisfy a subsequent request by another on-station processor. More complicated scenarios are described in [4].

We classify the NC hits based on the type of access (shared or exclusive) and the current coherence state of the line. There are five possible combinations:

- *SHR_LV* A shared read to an NC-owned line (Local Valid). The NC responds with data.
- *SHR_GV* A shared read to a globally shared line (Global Valid). The NC responds with data.
- *SHR_LI* A shared read to a line that is dirty in one of the local processor caches (Local Invalid). The NC mediates the intervention to obtain a copy of the line.
- *EXC_LV* An exclusive read (or upgrade) to an NC-owned line. The NC responds with data or an upgrade acknowledgment (invalidate).
- *EXC_LI* An exclusive read to a locally dirty line (Local Invalid). The NC mediates the exclusive intervention.

Figure 8 indicates that while the hit rates can be quite good for some applications, there is considerable variability. FFT rarely hits because its data access pattern consists of migratory data which has little spatial or temporal locality. Radix has an all-to-all communication phase which is heavily write-dependent. Since its writing pattern is fairly random, the probability that a line will be shared on the same station decreases as more stations are used. This accounts for the declining hit rate. Except for Radix, where most hits come from sharing locally modified data (EXC_LI), the most prevalent source of hits is from accesses to global read-shared data (SHR_GV). Although the NC certainly does provide a (sometimes substantial) caching effect, the overall effect on performance is hard to determine from the hit rate alone.

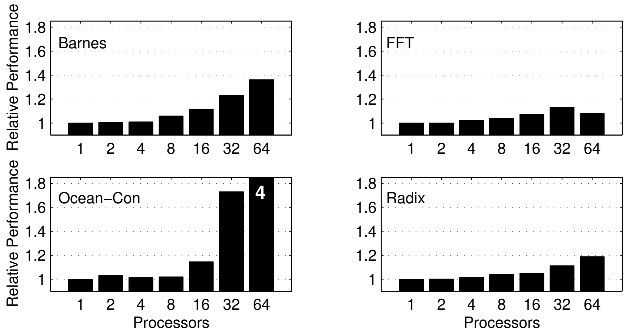

Figure 7. Turning off cache coherence

3.5 Coherence Overhead

NUMAchine's cache coherence scheme efficiently uses the ordering and multicasting properties of rings to achieve low coherence overhead. As a test, we ran the simulator without enforcing coherence. Although the machine is rendered functionally incorrect, it is possible to evaluate the coherence overhead and place an upper bound on any potential performance improvement.

To model a non-coherent system, the memory model was changed. First, stores to a processor cache that find a line present in any state are treated as hits; no coherence information is passed on to the memory or NC. Second, read requests (shared or exclusive) to home memory always return data. The directory is neither checked nor updated. Finally, if a cache line is present in the NC in any state, it is considered a hit and data is returned. Note that NC misses must still fetch the line remotely, although the remote memory is guaranteed to hit. Locked lines still generate NACKs.

By turning off coherence we eliminate three types of overhead: *i)* all NACKs, except for local NACKs from the NC, *ii)* latency to obtain write ownership (invalidation), and *iii)* latency for fetching modified cache lines from the caches of other processors, because the home memory always hits.

The results for Barnes, FFT, Ocean and Radix are shown in Figure 7. Ocean shows dramatic improvement because data is typically delivered from the caches when coherence is off. With cache coherence present, this data is invalidated by remote writes and requires re-reading of the cache line, creating additional ring traffic. Barnes shows a slight improvement, while FFT and Radix show almost no improvement. The conclusion is that the cache coherence protocol (including the effect of the NC) performs well for our test programs.

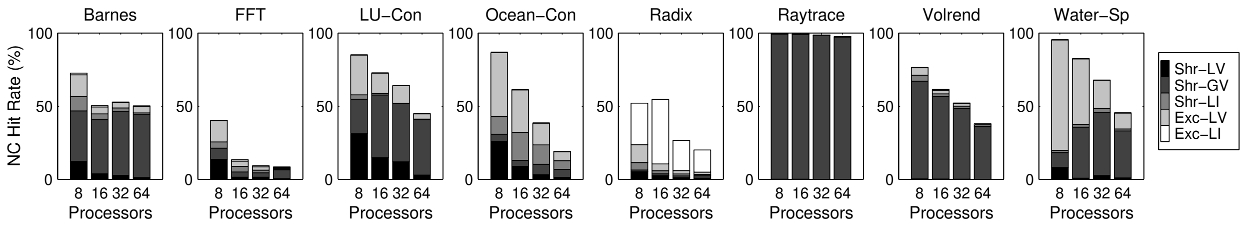

Figure 8. Network Cache hit rates

4. Prototype Performance

Using the simulator discussed in the previous section and measurements on the hardware prototype, we present access latencies and speedups for selected benchmarks.

4.1 Access Latencies

Table 2 gives the measured read latencies to different parts of the memory hierarchy on an unloaded system. The latency is measured from the time the processor issues the read request to the time the processor receives the first data packet. The ratio of remote to local access latencies, the "NUMA-ness" of the system, is about 4:1. This ratio puts NUMAchine mid-range when compared to the implementations mentioned in Section 1.1, which are as low as 2:1 and as high as 10:1. The absolute values of latencies are larger than in the other systems mentioned largely due to the implementation of all control logic in programmable logic devices. Nevertheless, performance of the system is good, as shown in the next section.

4.2 Speedups

We present the measured speedups of 8 unmodified SPLASH-2 benchmarks [22] in Figure 9 with a summary in Table 3. Results are shown for both the simulator and the 16-processor hardware prototype. Uniprocessor runtimes were obtained using the parallel versions of the benchmarks. Although the simulator is sometimes optimistic, general patterns for most benchmarks match hardware results. The exceptions are FFT and LU, which do not show a close correspondence.[3]

For the base problem sizes, 4 out of the 8 benchmarks have a parallel efficiency (speedup divided by the number of processors) greater than 60%. The remaining benchmarks (FFT, LU, Ocean and Radix) are known to speed up poorly for the base problem size, even with 16 processors [11]; our results confirm this finding. Of these, the speedups for Ocean and Radix have a parallel efficiency greater than 60% with larger problem sizes. We expect LU would improve similarly, but the 1024 size shown is the largest we can run on the simulator in a reasonable amount of time. Also, we are investigating the poor performance of FFT.

In general, benchmarks with low communication-to-computation (comm-to-comp) ratio and good locality of reference (Barnes, LU, and Water) are known to have better speedups. Our speedup results match this trend, with the exception of LU which is known to have load imbalance problems. The low ring utilization of these benchmarks observed in Figure 4 confirms their low comm-to-comp ratio. Although Raytrace and Volrend show only moderate locality, their low comm-to-comp ratio still produces low to medium ring utilization and good speedup is maintained. It is interesting to note that these five benchmarks all have high NC hit rates (above 50%), which is important for reducing latency.

[3]We are in the process of examining the simulator to determine the source of these discrepancies.

Table 2. Measured access latencies

level of hierarchy	150-MHz PCLKs	50-MHz SCLKs
L1 cache	1	*n/a*
L2 cache	6	*n/a*
Local memory	135	45
Local network cache	165	55
Other L2 cache	255	85
Rem. mem. (same ring)	594	198

Table 3. Performance summary

Benchmark Characteristics			Observations	
name	c-to-c ratio	locality	ring util.	speedup
Barnes	low	good	low	good
FFT	high	poor	higher	poor
LU-Con	low	good	low	poor
Ocean-Con	high	moderate	higher	good[1]
Radix	high	poor	higher	good[1]
Raytrace	low	moderate	low	good
Volrend	low	moderate	medium	good
Water-Sp	low	good	low	good

[1] *larger problem size required*

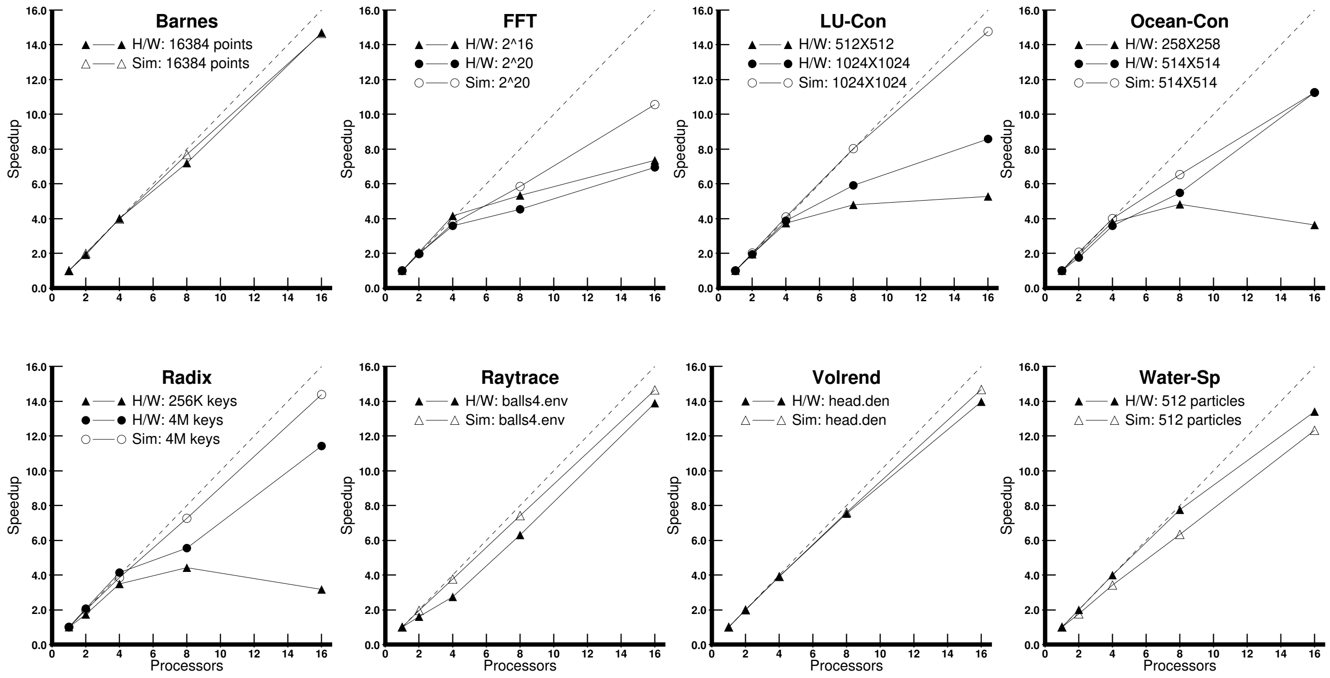

Figure 9. Parallel speedups on the NUMAchine prototype

Of the remaining three benchmarks (FFT, Ocean, and Radix), achieving good speedup is difficult because of the large bandwidth requirements: flow control is invoked for Ocean and Radix and all three show higher ring utilizations. Increasing the problem size for Ocean and Radix does result in good speedup, but the same is not true for FFT. Further increases in problem size for FFT and Radix has not helped much in other studies [11].

Another trend is the change in the slopes of the speedup curves when the number of processors is increased, which is common for DSM machines. The slopes are fairly constant up to 4 processors indicating that station parameters have been chosen well. The slope decreases for larger configurations, since some requests are now for off-station memory. This is particularly noticeable for applications with large remote communication requirements.

4.3 Larger Systems

Results for larger configurations are shown in Figure 10. These speedups were obtained using the NUMAchine simulator and show the expected prototype performance. Larger problem sizes were used for FFT, LU, Ocean and Radix.

The performance of the system is good, achieving a parallel efficiency of greater than 60% for 7 out of 8 benchmarks on 64 processors. It remains to be seen how well the prototype Central Ring performs, although speedups obtained through simulation are encouraging.

5. Current Status

The NUMAchine prototype consists of three Local Rings, each containing four stations (16 processors). The Central Ring is undergoing testing, and upon completion will link together the Local Rings to form a 48-processor system. Updates on the hardware status, with photographs, can be found at `http://www.eecg.toronto.edu/parallel`.

We have developed a custom parallel POSIX-compliant operating system, Tornado [3], to take advantage of NUMAchine's inherent clustering and provide user-customizable support in the kernel for parallel file systems. Tornado boots and runs parallel programs such as the SPLASH-2 suite and numerous X11 programs. The systems group is also working closely with IBM research, and has ported the K42 operating system to NUMAchine [10].

6. Conclusion

Lessons learned from the evolution of the desktop PC are that cost, usability, and modularity are crucial to gain widespread acceptance of an architecture. Today, 4- and 8-way multiprocessors are prevalent in the workstation environment. We believe that medium-scale parallel processing will also become widely used. Hence, the goal of the NUMAchine project is to develop cost-efficient and usable parallel computing. Our experience has shown that it is pos-

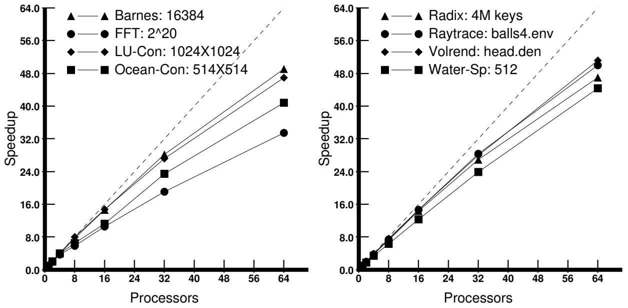

Figure 10. NUMAchine simulator speedups

sible to build an effective medium-scale multiprocessor using a simple architecture with off-the-shelf parts.

The NUMAchine hardware consists of a CC-NUMA 48-processor prototype using a hierarchical ring network. We have identified several beneficial ring properties: inherent broadcast, which provides an effective mechanism for the dissemination of cache coherence information; natural ordering of transactions, which not only simplifies the coherence protocol but also facilitates a simple implementation of sequential consistency; and a simple and fast bitmask routing algorithm. We have also demonstrated a two-level hardware cache coherence scheme which exploits the ring properties described above.

The simplicity of the ring network reduces the design complexity considerably. The incremental cost of inserting an extra node into the system is also small. This paper has shown that this cost-effective architecture does not sacrifice good performance.

NUMAchine's design naturally provides support for sequential consistency – an important aspect of usability. To further extend usability, one goal is to use the operating system to hide the "NUMA-ness" from the user by exploiting the underlying clustered architecture. The iterative tuning of parallel programs must also disappear if this class of machines is to become widely used. For these reasons, work on parallelizing compilers and parallel file systems are two major areas of research that will be carried out on the NUMAchine hardware.

References

[1] A. Charlesworth. Starfire: Extending the SMP envelope. *IEEE Micro*, 18(1):39–49, Jan/Feb 1998.

[2] K. Farkas, Z. Vranesic, and M. Stumm. Scalable cache consistency for hierarchically-structured multiprocessors. *J. of Supercomputing*, pages 345–368, 1995.

[3] B. Gamsa, O. Krieger, J. Appavoo, and M. Stumm. Tornado: Maximizing locality and concurrency in a shared memory multiprocessor operating system. In *Proc. of the 3rd Symposium on Operating Systems Design and Implementation*, pages 87–100, 1999.

[4] A. Grbic. Hierarchical directory controllers in the NUMAchine multiprocessor. Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, 1996.

[5] A. Grbic, S. Brown, S. Caranci, et al. Design and implementation of the NUMAchine multiprocessor. In *Proc. of the 35th Design Automation Conference*, pages 66–69, 1998.

[6] R. Grindley. *The NUMAchine Multiprocessor: Design and Analysis*. PhD thesis, Dept. of Electrical and Computer Engineering, University of Toronto, 1999.

[7] M. Heinrich, J. Kuskin, D. Ofelt, et al. The Stanford FLASH multiprocessor. In *Proc. of the 21st Intl. Symposium on Computer Architecture*, pages 302–313, 1994.

[8] Hewlett Packard. *Architecture Reference Guide—V2500 Server*. Document No. A3725-96004, 1999. Available at http://docs.hp.com/hpux/systems/#vclass.

[9] M. D. Hill. Multiprocessors should support simple memory-consistency models. *Computer*, 31(12):28–34, Aug 1998.

[10] IBM Research. *The K42 Project*. Information is available at http://www.research.ibm.com/K42/.

[11] D. Jiang and J. P. Singh. A methodology and an evaluation of the SGI Origin2000. In *Proc. of the Joint Intl. Conference on Measurement and Modeling of Computer Systems*, pages 171–181, 1998.

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

[13] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA highly scalable server. In *Proc. of the 24th Intl. Symposium on Computer Architecture*, pages 241–251, 1997.

[14] D. Lenoski, J. Laudon, T. Joe, et al. The DASH prototype: Implementation and performance. In *Proc. of the 19th Intl. Symposium on Computer Architecture*, pages 92–103, 1992.

[15] K. Loveless. The implementation of flexible interconnect in the NUMAchine multiprocessor. Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, 1996.

[16] T. Lovett and R. Clapp. STiNG: A CC-NUMA computer system for the commercial marketplace. In *Proc. of the 23rd Intl. Symposium on Computer Architecture*, pages 308–317, 1996.

[17] G. Ravindran and M. Stumm. A performance comparison of hierarchical ring- and mesh-connected multiprocessor networks. In *Proc. of the 3rd Intl. Symposium on High Performance Computer Architecture*, pages 58–69, 1997.

[18] G. Ravindran and M. Stumm. On topology and bisection bandwidth for hierarchical-ring networks for shared-memory multiprocessors. In *Proc. of the Intl. Symposium on High Performance Computing*, 1998.

[19] Tera Computer Company. Information on their architecture is available at http://www.tera.com/tech/index.html.

[20] J. Torrellas and D. Padua. The Illinois aggressive coma multiprocessor project (I-ACOMA). In *Proc. of the 6th Symposium on the Frontiers of Massively Parallel Computing*, 1996.

[21] J. Veenstra. MINT tutorial and user manual. Technical Report 452, Computer Science Department, University of Rochester, May 1993.

[22] S. Woo, M. Ohara, E. Torrie, J. P. Shingh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. of the 22nd Intl. Symposium on Computer Architecture*, pages 24–36, 1995.