Design and Implementation of the NUMAchine Multiprocessor*

A. Grbic, S. Brown, S. Caranci, R. Grindley, M. Gusat, G. Lemieux,
K. Loveless, N. Manjikian†, S. Srbljic‡, M. Stumm, Z. Vranesic and Z. Zilic§

Dept. of Electrical and Computer Engineering, University of Toronto, Canada

Abstract

This paper describes the design and implementation of the *NUMAchine* multiprocessor. As the market for CC-NUMA multiprocessors expands, this research project provides a timely architectural design and cost-effective prototype. The key to the successful implementation of our 48-processor prototype is the use of off-the-shelf components and programmable logic devices. Since this machine will serve as a research vehicle for parallel software development, a number of hardware features to enhance experimentation have been included in the design.

1 Introduction

This paper describes the hardware implementation of a multiprocessor called NUMAchine [7]. In NUMAchine, processors, caches, and memory are physically distributed throughout the system. The memory is shared by all processors, but the access latency depends on location. Hardware automatically maintains coherent copies of data throughout the system. This type of multiprocessor is known as a CC-NUMA (Cache-Coherent Non-Uniform Memory Access) distributed shared-memory multiprocessor. CC-NUMA multiprocessors are quickly gaining commercial acceptance for use as powerful compute, file, network and web servers. The shared-memory model eases the transition from uniprocessor-based software. In addition, multiprocessors that support sequential consistency provide ease-of-programming by extending the already familiar uniprocessor programming model.

There has been considerable research interest into how to design CC-NUMA multiprocessors to achieve maximum performance. The Stanford DASH [5] multiprocessor is a mesh-based system that uses separate request and response communication networks. The follow-on project, FLASH [4], is centered around the MAGIC chip, a customized RISC engine acting as coherence, network, and memory controller. Both DASH and FLASH use MIPS-based computing nodes. In contrast, the MIT Alewife project [1] uses the custom-designed Sparcle processor that tolerates memory latency by quickly switching among multiple threads.

In addition to high performance, we are also concerned with cost, ease-of-programming and providing features for parallel software research. The NUMAchine architecture is hierarchical and modular so that small systems can be affordably extended into larger ones, up to a few hundred processors. A directory-based hardware cache coherence protocol scales efficiently with system size by exploiting the hierarchical architecture. The architecture is also exploited to efficiently implement a sequentially-consistent memory model. As well, NUMAchine includes novel hardware features that enhance experimentation. First, non-coherent operations and directory manipulation features permit direct comparison between hardware and software cache coherence. Second, block data transfers and coherence operations reduce cache pollution, bandwidth use, and overhead. Third, a hardware barrier register supports efficient parallel synchronization. Fourth, flexible and non-intrusive performance monitoring hardware improves the observability and accuracy of software-only measurement schemes.

The 48-processor prototype system is based on 150 MHz MIPS R4400 processors. It has a peak performance of 1.7 GFLOPS and a peak bandwidth of 400 MB/s at any point in the interconnection network. The prototype is constructed entirely with off-the-shelf components and programmable logic devices (PLDs) without compromising our goal of a 50 MHz system clock. The use of PLDs allows for a low cost implementation and rapid resolution of problems. PLDs also provide the flexibility to implement new ideas by modifying some of the novel hardware features described above.

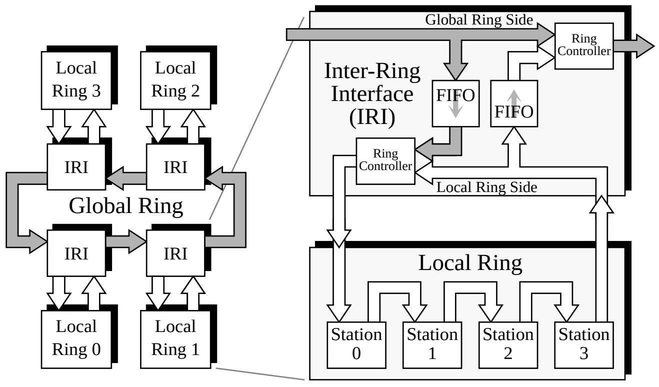

Figure 1: NUMAchine Architecture

2 NUMAchine Architecture

This section gives an overview of the NUMAchine multiprocessor. Computing nodes are clustered into bus-based units called *stations*.

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

†now with Queen's University at Kingston, Canada

‡with University of Zagreb, Croatia

§now with Lucent Technologies, Allentown, PA, USA

A NUMAchine station uses a bus to connect four processor modules, up to two memory modules, a network interface module, and up to two I/O modules. Depending on the address, processors transparently access either local or remote memory. Remote memory requests are handled by the network interface, which routes them to the correct remote station. The network interface also includes a third-level *network cache* for holding data from remote memories.

Stations are interconnected by a hierarchical arrangement of unidirectional bit-parallel rings, as shown in Figure 1. This architecture provides modularity and flexibility because stations may be added to rings as necessary, and the depth of the hierarchy can be extended for more rings. Hierarchical rings, as an alternative to meshes, have been shown to perform well for systems with up to 128 processors and workloads with medium to high memory access locality [6].

A slotted-ring protocol transfers data packets across the network and then reassembles these packets at the remote station. Each ring packet is a portion of a bus transaction augmented with additional routing information. The routing information is contained in a set of *routing masks*, one for each level of the hierarchy [3]. The bits in a routing mask correspond to siblings in that level of the hierarchy. If a particular bit is set, a copy of the packet descends to the children in that group. Packets only travel as high as necessary in the network to reach all the targets. This routing mask mechanism provides a concise way to multicast and broadcast data, and is used by the cache coherence protocol to maintain the directories.

The choice of a ring topology for the interconnection network provides three important benefits. First, the unique path between any two points in the network maintains the relative ordering of packets between the same source and destination. Second, the ring topology provides an efficient method for multicasting and broadcasting. Third, the point-to-point ring connections simplify wiring and allow for a simple routing scheme. The first two benefits allow for an efficient cache coherence implementation, while the third allows fast data transmission rates.

The NUMAchine write-back/invalidate cache coherence protocol is hierarchical, ownership-based, and uses directories to maintain information on cache lines. The directories maintain cache line state information and locations, represented by routing masks. The routing mask scheme permits the directory to scale logarithmically as the system size increases [3]. The protocol supports sequential consistency by exploiting the order-preserving properties of the ring hierarchy.

3 Design Methodology

Our design methodology relies on the use of commodity components to satisfy cost constraints and to reduce design time. These components include two types of PLDs: FPGAs and CPLDs. PLDs permit hardware-software co-design and allow the hardware to be later modified to support new features.

Our design flow consisted of multiprocessor simulations to define the architecture, design partitioning into multiple printed-circuit boards (PCBs), and bottom-up design of each PCB. The simulations guided the selection of architectural parameters, such as the number of ring levels, stations per ring, and processors per station. These parameters then guided the partitioning into multiple PCBs. The bottom-up design methodology for each PCB consisted of the following steps: (1) logic partitioning, (2) assignment of logic to commodity components or PLDs, (3) design entry for PLDs, (4) high-level schematic design entry, (5) board- and system-level simulation, and (6) PCB layout.

The CAD tools used for hardware design were Cadence Logic Workbench (LWB), Cadence Allegro Layout and Altera MAX+plus II. The two Cadence tools provide a state-of-the-art system for the implementation of digital systems. We used LWB for board-level schematic capture and simulation, and Allegro for PCB layout. MAX+plus II supports logic implementation in Altera PLDs using graphical and text-based design entry methods, as well as full timing simulation. The bottom-up PCB design approach resulted from difficulties encountered in integrating LWB and MAX+plus II, as documented in an earlier paper [2].

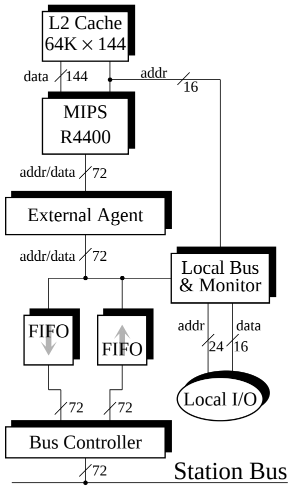

Figure 2: Processor Board

4 Hardware Implementation

Our overriding design goal is high performance within our cost constraints. All boards, busses, and rings operate at 50 MHz, and processors run internally at 150 MHz. Achieving such speeds with current PLDs is a major challenge: clever design methods and careful hand-tuning of circuits is often necessary. Experience guides how and when to apply these manual optimizations [8].

The use of field-programmable devices enabled the design of custom controllers without the higher cost of full-custom chip design. Reprogrammability allowed rapid resolution of problems in the design. Furthermore, the rapid pace of technological improvement in programmable devices enabled, as well as encouraged, prototyping and revisions to integrate multichip designs into a single chip for equivalent cost. The end result is an operational system in less time and at a lower cost than full-custom design.

4.1 Processor Board

The NUMAchine processor board uses a MIPS R4400 64-bit processor running at 150 MHz, with 1 MB of second-level (L2) cache as shown in Figure 2. The 128-bit L2 cache interface runs at 75 MHz and is controlled entirely by the processor, while the 64-bit[1] external (system) interface runs at 50 MHz. To communicate with external memory, the R4400 requires an *External Agent* (EA) circuit. The EA accepts requests from the processor, and forwards external requests for retrieving or invalidating L2 cache lines to the processor. The EA also serves as a bridge to an on-board local bus. The FIFO buffers between the EA and the station bus hold bursts of incoming requests/responses and outgoing write-back data from the processor. A *mailbox* register feature in the FIFOs allows a read request to bypass write-backs waiting in the queue. Read requests are given higher priority and issued to the memory as soon as possible. The bus controller manages data transfers between the FIFOs and the station bus.

[1] 72-bit connections are shown in the figures. These include data and parity/ECC bits

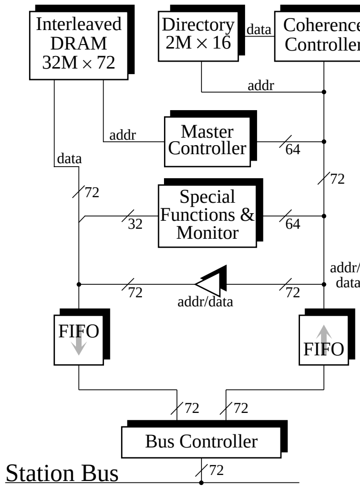

Figure 3: Memory Board

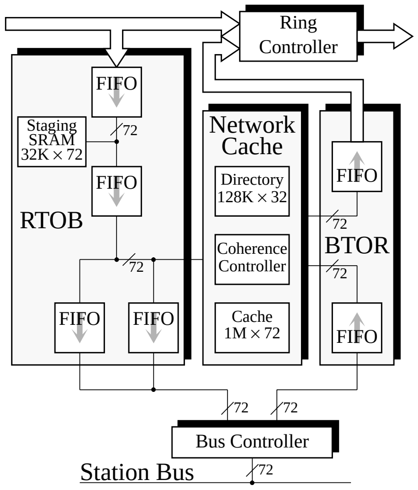

Figure 4: Network Interface Board

The on-board local bus provides a serial port, boot PROM, local hardware monitoring, and a connection to a diagnostic board for bootstrap debugging. The local bus also contains interrupt and barrier registers. The interrupt register provides an efficient mechanism for externally-generated interrupts, while the barrier register supports efficient global synchronization for parallel applications.

4.2 Memory Board

A block diagram of the Memory board is shown in Figure 3. FIFOs receive and send packets to the bus through the *Bus Controller*. The *Master Controller* provides control signals for the FIFOs and coordinates the other functional blocks. A *DRAM controller* manages up to 256 MB of local DRAM. The *Coherence Controller* maintains a directory in SRAM and implements all of the coherence actions and state transitions for the cache coherence protocol. For 128-byte cache lines, 4 MB of directory SRAM are needed for 256 MB of DRAM. The *Special Functions and Monitoring* unit manages special hardware functions on address ranges, described in Section 4.7, and generates interrupts.

For high performance, the DRAM memory is 4-way interleaved. Furthermore, data access is overlapped with cache coherence operations, and successive memory requests are pipelined. For each memory request, the master controller starts the Cache Coherence and the DRAM controllers simultaneously. While the DRAM is accessed, the directory entry is updated and the header packet for the response is placed in the outgoing FIFO. Data packets from DRAM are then placed in the outgoing FIFO (64 bits every clock cycle). There is sufficient buffering to permit processing of the next request as soon as the directory entry has been updated for the previous request. A single memory board can nearly saturate the bus with response data. When two memory boards are used together on a bus, they are interleaved at the cache-line level to increase parallelism and reduce total latency.

4.3 Network Interface Board

The main components of the Network Interface board are the *BTOR* (Bus TO Ring) interface, the *RTOB* (Ring TO Bus) interface, and the *Network Cache*, as shown in Figure 4. The Network Cache consists of the Coherence Controller, 8 MB of SDRAM to cache data from remote memories, and a directory (512 KB of SRAM). The BTOR controller moves data from the incoming bus FIFO to the outgoing ring FIFO or into the network cache input buffers. It also moves data from the network cache output buffers to the outgoing ring FIFO. The *Ring Controller* sends this data as individual 64-bit ring packets mixed in with existing ring traffic. At the destination, packets are transferred from the ring into the incoming ring FIFO. These packets are reassembled into cache lines by the RTOB controller in the *Staging SRAM* and then sent to the network cache input buffers or to one of the outgoing bus FIFOs. The network cache alternates packet processing between the two sides. On each request, the Coherence Controller updates the directory information according to the NUMAchine protocol and may generate responses to the BTOR and/or RTOB sides.

The datapaths on the Network Interface board were initially implemented using discrete buffers. The large number of components required for this implementation complicated the layout of the PCB. In the following revision of the board, we were able to implement this datapath with FPGAs and realize a net savings in cost. This redesign also yielded improved performance by adding more parallelism to the datapaths.

4.4 I/O Board

The I/O board contains a 100 MHz MIPS R4650 processor, up to 32 MB of DRAM storage, 4 DMA channels, and a 33 MHz PCI bus. The PCI bus connects up to 4 SCSI controllers and one PCI expansion slot. The R4650 manages all data transfers and can also execute protocols for parallel I/O and networking. The I/O board is designed to facilitate parallel I/O research through a flexible bus interface and efficient polling/interrupts.

4.5 Station Bus

We chose the Futurebus+ physical backplane for NUMAchine stations because it provides a wide bus with up to 14 card slots. We did not, however, use the Futurebus+ protocol because it is overly complex for our purposes. In addition, the Futurebus+ protocol chips available during our early design work did not meet our 50 MHz speed requirement. We designed a custom, synchronous, split-transaction protocol with a centralized, pipelined arbiter. The core datapath width is 64 bits; a 128-bit bus was considered, but architectural simulations indicated that the marginal improvement in performance did not justify the cost and complexity.

The synchronous protocols for NUMAchine necessitate low-skew clock distribution. For each station, we generate a central clock signal which is then replicated and distributed to all boards using differential ECL for low-skew fanout and noise immunity. Additionally, phase-locked loops are employed to reduce the clock skew.

4.6 Rings

A local ring in NUMAchine is assembled by connecting together the Network Interface boards of up to four stations with high-density, controlled-impedance ribbon cables. The Ring Controller on the Network Interface board adds only a single cycle delay to traffic traveling to the next station. When a packet reaches its final destination, it is removed from the ring and a new packet can be injected by the Ring Controller. In addition, it can simultaneously source and sink data at the maximum link transmission rate of 400 MB/s.

The Ring Controller is implemented as a single CPLD and a number of discrete tri-state registers. The ring clock rate of 50 MHz is limited by the speed at which our design can multiplex the data between the ring traffic and new traffic from the station. A full-custom implementation would allow for a much faster ring clock rate.

To assemble a larger NUMAchine system, the ring hierarchy must be expanded to another level. Rather than constructing four Inter-Ring Interface boards as shown in Figure 1, which would be limited to the same speed as the local rings, we decided to construct it in a more centralized fashion. A Global Ring (GR) backplane was designed with two types of daughter cards. A Local Ring Interface (LRI) daughtercard provides the local ring controller. Signals from four LRIs are redistributed on the GR backplane to six Data-Path (DP) daughtercards and a central controller. Each DP implements an 18-bit slice of the global ring using FIFOs and an FPGA. This organization clusters the global ring logic so that a clock rate of 80 MHz can be achieved for a maximum data rate of 640 MB/s.

4.7 Novel Hardware Features

Dedicated monitoring circuitry is included in all parts of NUMAchine, including the processor, memory, and interconnect. Applications can reconfigure the FPGA-based monitoring hardware for data collection without resetting the machine. Most of the monitoring circuits comprise a large FPGA, fast 64K$\times$32 SRAMs, and supporting CPLDs. A novel concept that is included throughout the monitor is that of a *phaseID*, which is derived from a writable 4-bit register on each processor board. The phaseID, which is attached to *all* transactions leaving the processor, can be used by an application to separate performance data into fine-grained phases. The monitoring hardware on each processor board can separately count monitoring events based on transaction type, phaseID, or address range. The monitoring hardware on the memory board can use these filters as well as the originating processor and cache line state.

Additional specialized hardware also provides uncached operations, non-coherent operations, and special functions. Uncached operations bypass the secondary and network caches. Non-coherent operations still use caches, but bypass the cache coherence protocol when transferring cache lines. Special functions provide low-level control including directory read/write operations, returning modified cache lines to memory, invalidating copies of cache lines system-wide, multicasting of cache lines, block data transfers, prefetching into network caches, and forced write-backs to home memory. In addition, the prototype supports 64-byte or 128-byte cache lines.

4.8 Performance

Table 1 gives the measured latency of read requests on the prototype hardware in 150 MHz processor cycles and 50 MHz system cycles. Note that for remote requests, 1 hop across the ring was used for the data in the table because the numbers were measured on a two-station system. Although not measured, the additional latency to traverse the global ring is estimated to be 24 system cycles.

Level of hierarchy	Procr. cycles (6.67ns)	Sys. cycles (20ns)
L1 cache	1	
L2 cache	6	
Local memory	135	45
Local network cache	165	55
Other L2 cache	255	85
Rem. mem. (same ring)	594	198

Table 1: Measured memory access latencies

5 Current Status

Currently, a 4-station (16-processor) NUMAchine system is operational. Further information, including photographs, can be found on the WWW.[2] A custom parallel operating system with a UNIX-like interface, called Tornado, has been developed. The operating system boots, allows logins, and runs various applications. All of the boards for the NUMAchine prototype have been fabricated. The hardware components are being integrated together to form the complete system.

References

[1] A. Agarwal et al. The MIT Alewife Machine: Architecture and Performance. In *Proc. of the 22nd Annual ISCA*, pages 2–13, June 1995.

[2] S. Brown et al. Experience in Designing a Large-Scale Multiprocessor usign Field-Programmable Devces and Advanced CAD Tools. In *Proc. of the 33rd DAC*, pages 427–432, Las Vegas, NV, June 1996.

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

[4] J. Kuskin et al. The Stanford FLASH Multiprocessor. In *Proc. of the 21st Annual ISCA*, pages 302–313, Chicago, IL, May 1994.

[5] D. Lenoski et al. The DASH prototype: Implementation and performance. In *Proc. of the 19th Annual ISCA*, pages 92–103, Gold Coast, Australia, May 1992.

[6] G. Ravindran and M. Stumm. A Performance Comparison of Hierarchical Ring- and Mesh-Connected Multiprocessor Networks. In *Proc. of the Third International Symposium on HPCA*, pages 58–69, San Antonio, Texas, February 1997.

[7] Z. Vranesic et al. The NUMAchine Multiprocessor. Technical Report CSRI-324, Computer Systems Research Institute, University of Toronto, 1995.

[8] Z. Zilic et al. Designing for High Speed-Performance in CPLDs and FPGAs. In *The 3rd Canadian Workshop on Field-Programmable Devices (FPD'95)*, pages 108–113, May 1995.

[2] `http://www.eecg.toronto.edu/parallel`