RAPID HARDWARE PROTOTYPING ON RPM-2: METHODOLOGY AND EXPERIENCE

Michel Dubois, Alain Gefflaut*, Jaeheon Jeong,
Adrian Moga, and Koray Öner

CENG 97-28

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
(213)740-4475
Fax: (213) 740-7290
{dubois, gefflaut, jaeheonj, moga, oner}@paris.usc.edu

December 1997

RAPID HARDWARE PROTOTYPING ON RPM-2: METHODOLOGY AND EXPERIENCE

Michel Dubois, Alain Gefflaut*, Jaeheon Jeong, Adrian Moga, and Koray Öner

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
(213)740-4475

*Siemens AG
Public Communication Networks Group
Munich, Germany

{dubois, gefflaut, jaeheonj, moga, oner}@paris.usc.edu

Abstract

Field-Programmable Gate Arrays is an emerging technology which promises easy hardware reconfigurability by software at low cost. Entire systems can be built in which some parts are programmable. Such systems are flexible hardware platforms, which can then be tailored to implement various architectures. Each architecture prototype is a detailed hardware implementation of the architecture --including I/O-- on which complex software systems can be ported.

We have built a multiprocessor emulator called RPM --Rapid Prototyping engine for Multiprocessor systems. The second version of the hardware called RPM-2 is up and running. In this paper, we present the modeling methodology, the performance collection mechanism, the calibration of the emulator as well as emulation results obtained for the emulator of a cache-coherent non uniform memory access multiprocessor (CC-NUMA)

Keywords: Field Programmable Gate Arrays, Rapid Hardware Prototyping, Multiprocessor, Shared-Memory, System Design.

1. INTRODUCTION

There are currently many competing ideas to implement multiprocessor systems. Among shared-memory systems, many variants, ranging from software or hybrid hardware/software DSM (Distributed Shared Memory) systems [13] to cache-based multiprocessors with hardware-enforced coherence [21] are under intense scrutiny in the research community. Several projects both in industry and academia evaluate new ideas in multiprocessor systems by building hardware prototypes [1] [4][11][12][14]. In other projects, abstracted machine prototypes are developed on software simulation platforms [5][17][18].

The advantages of hardware prototyping are well known. In an hardware prototype detailed implementations are worked out and thus new problems are discovered and new ideas are found in the process. Hardware prototypes are the ultimate proof of concept of an architectural idea and lead to complete system design in which hardware/software complexity can be evaluated. Moreover, they can run interesting workloads such as operating systems and commercial workloads. In an academic environment, hardware prototyping is an extremely valuable experience for graduate students.

However, hardware prototypes suffer from two major problems: they take too long to build and they are very expensive. Thus only a few ideas are validated in hardware and many good ideas are never implemented. By the time a prototype really works (i.e. when it runs for days without crashing), it is often obsolete. This problem of hardware obsolescence is particularly bad since prototyping projects must explore *future* architectures. The three aspects of hardware obsolescence are:

- first, the *absolute* speed of the prototype is no longer up to par with current hardware;
- second, the *relative* speeds of components have changed so that experiments run on the prototype become meaningless (for example, processor speed is improving much faster than DRAM speed);
- third, the new architecture ideas embodied in the prototype have become uninteresting.

Besides the high cost and long development time, hardware prototypes are often hard to observe. On the other hand, software simulations are very flexible, observable, and relatively inexpensive to develop. They may range from detailed cycle-by-cycle simulations of a target, which are very slow, to trace-driven simulations, in which no machine is simulated but instructions of different processors are simply interleaved and some events such as cache misses are counted. There is often a trade-off in software simulations between realistic simulation speeds and realism [18].

Hardware emulation, the approach adopted in RPM (Rapid Prototyping engine for Multiprocessors) is an intermediate approach between software simulation and hardware prototyping. RPM facilitates the rapid and economical development of complete hardware systems for various configurations of shared-memory multiprocessors with a NUMA (Non-Uniform Memory Access) architecture. RPM reaps most of the benefits of hardware prototypes at a much reduced cost and design effort. Because of its flexibility, the hardware can adapt during its lifetime to the rapid evolution of technology trade-offs and of new ideas in architecture. RPM is also much more observable than typical hardware prototypes. The cost of this flexibility in RPM is its reduced speed.

In the class of architectures which can be prototyped on RPM, each processing element is made of a processor, some cache, and a share of the main memory. The prototyping methodology is based on FPGA (Field-Programmable Gate Arrays) technology [22]. Processors and memories on each board are off-the-shelf but controllers are implemented with FPGAs. To modify the architecture or the parameters of the architecture, the FPGAs are reprogrammed in VHDL and the VHDL programs are downloaded in the FPGAs. From the software point of view, an RPM prototype is indistinguishable from a (slow) hardware implementation of a SPARC multiprocessor.

The overall architecture of RPM and the details of the hardware design can be found in [3] and [15][1]. This paper describes the methodology, tools and environment that we have developed to exploit the emulator, as well as our experience with hardware prototyping using FPGAs. In Section 2 of this paper we briefly describe RPM, as well as the first prototype, a Cache-Coherent Non-Uniform Memory Access (CC-NUMA) architecture similar to the Stanford DASH [12]. Then, in Sections 3 and 4, we expand on the various aspects of the prototyping methodology: multi-cycle Pclock emulation, time scaling, count memory, architecture verification, and emulator programming. Section 5 shows emulation results. We show the results obtained on systems of different sizes and with different application data set sizes. In Section 6, we comment on several aspects of the prototyping methodology. Finally, we conclude in Section 7.

2. OVERVIEW OF RPM-2 AND THE CC-NUMA EMULATION

2.1. Hardware Substrate

RPM is made of nine SPARC processors connected to a Futurebus+ backplane and is currently clocked at 5 MHz. Normally, one processor acts as an I/O node. The single chip processors have both an integer and a floating-point pipelines and the execution of floating operations are overlapped with the execution of integer instructions. They have no on-chip cache and therefore all instruction fetches and data accesses are observable. The block diagram of each processor board is shown in Figure 1. Processors and memories on each board are off-the-shelf but controllers are implemented with FPGAs. In all each boards contains 8 FPGAs, each with the equivalent of more than 10,000 logic gates.

In a typical emulation, a fraction of every on-board memories emulates the target system's caches and memories and the rest is used for performance collection and for the emulation of special registers and buffers (such as those needed to support virtual memory). The first-level memory controller drives the emulation of each pclock. Typically, at the start of each pclock emulation, it resumes the processor execution for one cycle, receive the next processor access (if any), blocks the processor, and then emulates the memory access (in one or several pclocks). The emulation of each pclock is implemented by a combination of control in the FPGAs and memory space in the RAMs attached to the memory controllers. The number of clocks per pclock can be easily changed and it depends on the complexity of the pclock emulation.The Delay Unit (DU) is a programmable chip which emulates variable interconnection delays. Messages that are sent out of the processor node are stored in the Delay Unit for an amount of time that is programmable. If L is the length of a packet, then the delay is equal to $\alpha + \beta \times L$, where α is a fixed delay per packet and β is the time per 32-bit word transfer. The FutureBus+ interconnection is 32-bit wide, and transfers one 32-bit word every 100nsec, for a total bandwidth of 40 Mbytes per second. The main memory speed can be modified by suspending memory requests using interleaving registers as described in [3].

1. However, a new version of RPM (RPM-2) has been built and we will highlight some of the differences in Section 2.

The SCSI interface and serial port allow every board to be configured as an I/O node.

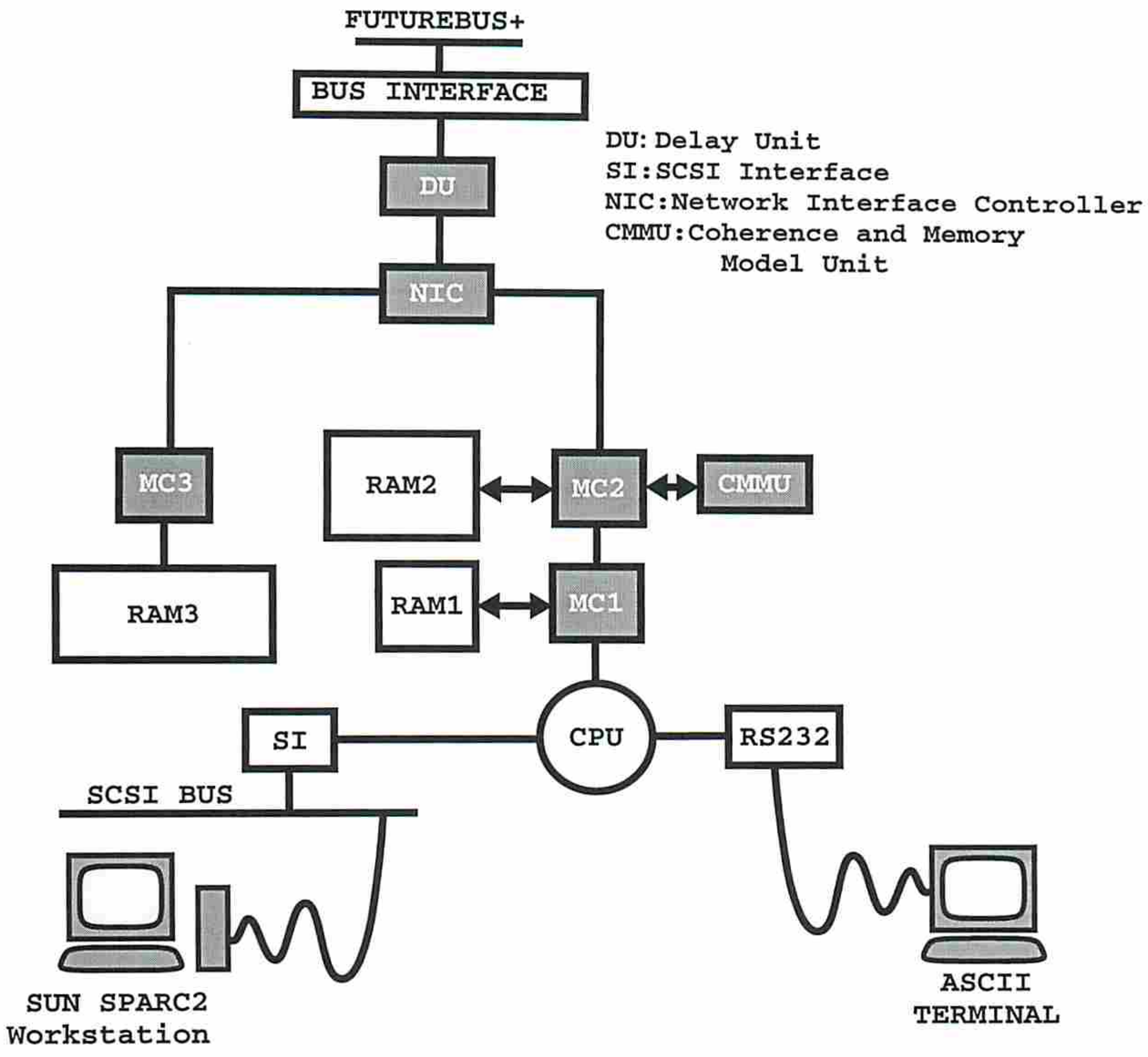

Figure 1. Block Diagram of RPM-2's Processor Node [3]

The CPU is an LSI Logic L64831 SPARC IU/FPU. The first-level cache (RAM1) and the second-level cache (RAM2) are built with SRAM SIMMs and have a maximum capacity of 2 Mbytes and 8 Mbytes respectively. Both cache controllers (MC1 and MC2) are made of 2 Xilinx 4013 FPGAs. The CMMU is made of one Xilinx 4013. The main memory (RAM3) is made of 96Mbytes of DRAM connected to 2 Xilinx 4013 FPGAs plus an off-the-shelf DRAM controller (Cypress CYM7232). The delay unit is built with a FIFO controlled by one AMD MACH 210 chip. The FIFO (8 kbytes) contains blocks and messages which are sent to the bus interface after a programmable delay depending on the target machine's interconnect latencies and packet size. The NIC is built with 1 Xilinx 4013. The FutureBus+ chip set comes from Newbridge and National Semiconductors and it includes bus transceivers plus the Newbridge LIFE chip. The SCSI interface of the I/O board is attached to the SCSI bus of a SPARC station II which currently serves as an I/O server and console for RPM. The RS232 serial interface can connect to a terminal and is used mostly for debugging purposes. The board size is 22"x16" but only three-fourth of the board is populated.

The current hardware (RPM Version 2 or RPM-2) is slightly different from the hardware described in [3], which was RPM Version 1. First of all the boards have been designed to clock at

20MHz instead of the 10MHz of RPM-1. Second, the internal bus, which caused reliability problems in the first hardware version, has been replaced by a Network Interface Controller (NIC). The added advantage of the NIC is that it can be programmed to control the flow of packets in and out of the board. Another modification is the addition of on FPGA to the second-level cache controller to implement the Coherence and Memory Model Unit (CMMU). This chip is dedicated mostly to latency tolerance hardware [9]. Finally, I/O and virtual memory support have been upgraded to facilitate the port of an operating system and of commercial workloads. Pictures of RPM-2 and of one of its boards are shown in Figure 2 and in Figure 3..

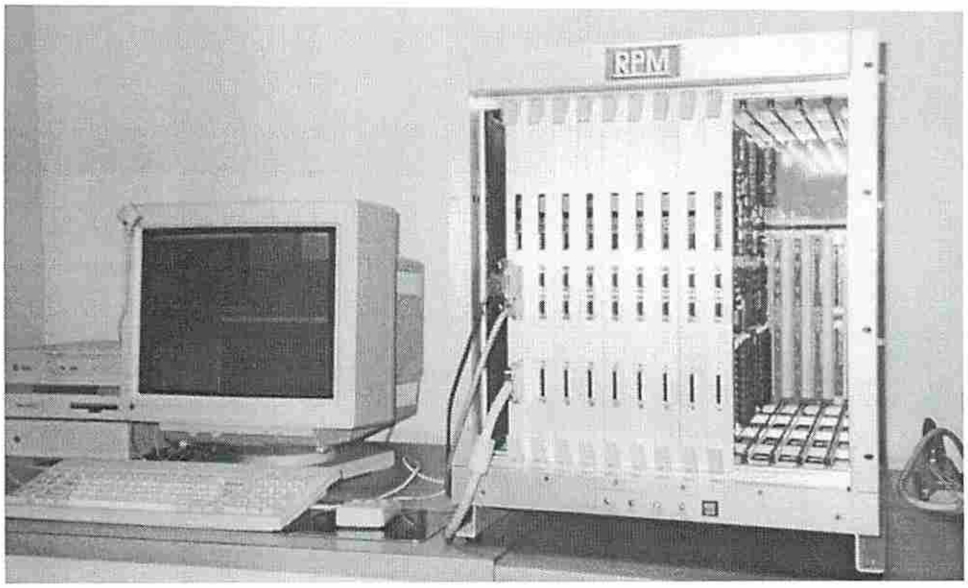

Figure 2. RPM-2 with 9 boards and its Sun host

The memories of every board can be accessed in *Test mode* or in *Emulation mode*. In Test mode (mostly used for booting and debugging), the memories on all boards are all part of a contiguous address space and every location can be accessed randomly. To configure RPM into a target multiprocessor, the FPGA programs of all three controllers are written in VHDL, compiled and then downloaded through the I/O node as part of the booting procedure. Once the FGPAs are programmed, the machine is in test mode. Memories are initialized, the code and data of the application are downloaded into RPM's main memory and the machine then switches to emulation mode where memory accesses are restricted by the addressing mechanisms of the target machine. From this time on, RPM behaves like the target machine

2.2. An Emulator for Small-Scale CC-NUMAs

Today's successful multiprocessors are mostly small-scale systems (2 to 16 processors) with shared memory and cache coherence enforced by a snooping protocol on one or multiple buses. As an alternative to snooping, industry and academia are exploring other, point-to-point interconnect such as rings [2] or crossbars to connect ultra-fast processors in a small number (1 to 32).

Our first emulator is a CC-NUMA under strong ordering of shared-memory accesses [9].

2.2.1. Protocol

The protocol is write-invalidate with a directory organization based on Censier and Feautrier's design [7]. Besides the presence bits and the modify bit per block, we have added some transient state bits to support concurrent transactions on different directory entries. A miss request first goes to the home memory. Before returning a copy of the block, the home memory must sometimes invalidate copies (write miss in the presence of multiple remote copies) or obtain the dirty copy from the owner (miss in the presence of a dirty remote copy). This latter transaction takes four traversals of the interconnect (requester-to-home, home-to-dirty, dirty-to-home and home-to-requester). After invalidations are sent by the memory controller the block state is set to transient and the presence bits indicate which copies have pending invalidations. While invalidations are propagated, the memory controller is released and any other request to the block is *nacked* by the home node. As invalidations are acknowledged, the memory controller resets the presence bits; once all acknowledgments have been received the controller completes the transaction and sets the block to a stable state. More details on this protocol can be found in [16].

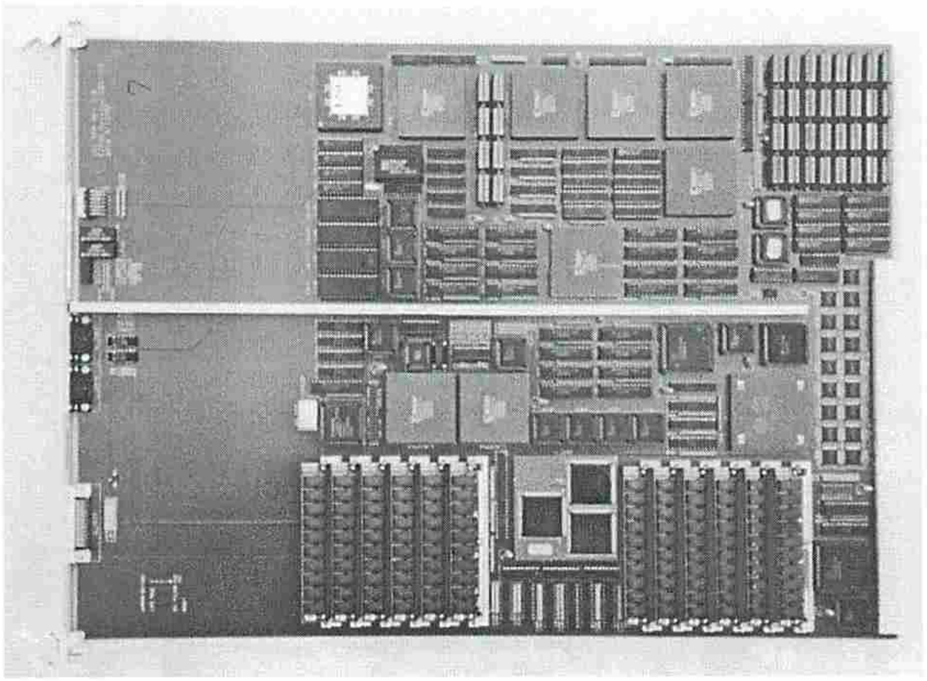

Figure 3. One board of RPM-2. Each board contains 8 Xilinx 4013 -3 FPGAs.

2.2.2. Architectural Parameters

The first-level cache is a 64kbyte direct-mapped write-through cache with a block size of 16 bytes. The second-level cache is a 1Mbyte direct-mapped write-back cache with 16 byte blocks.

The emulated memory size on each board is 32Mbytes. The basic hardware latencies listed in Table 1 (second column) are the number of cycles needed for the emulation. They were obtained by counting cycles in the RPM hardware. They are the lowest possible latencies for the CC-NUMA emulation.

Resource	Event	Basic Hard-ware (clocks)	100MHz processors (pclocks)
FLC	Data Read (single) or Instruction Hit	8	1
	Data Write Hit (single)	24	3
	Data Read Hit (double)	16	2
	Data Write Hit (double)	32	4
	Fill from SLC	16	2
SLC	Hit from FLC	40	5
	Hit from Bus Interface	24	3
	Miss Detection	16	2
	SLC fill	24	3
	SLC restart	24	3
Internal Bus	Request Packet	4	1/2
	Data Packet	8	1
DU and FutureBus+	Request Packet	24	8
	Data Packet	24	20
Memory	Miss (T_{miss})	40	15
	Send k Invalidations (T_{inv})	(10+4k)	(1+2k)
	Ack Invalidation (T_{ack})	24	5
	Receive Write Back (T_{wb})	40	15
	Get Block from Dirty (T_{md})	56	8
	Send Dirty Block to Requester (T_{mr})	72	9
	Nack (T_{nack})	24	5

Table 1. Hardware Latencies (no conflict) in the CC-NUMA Emulator on RPM-2
Miss is the time to receive a packet, fetch the block from DRAM, update the directory and send a packet back to the requester. *Send k Invalidations* is the time to send k invalidation packets. *Ack Invalidations* is the time to receive an acknowledgment for one invalidation and update the directory. *Receive Write Back* is the time to receive a block packet, store the block in DRAM, and possibly forward the block to the requester. *Get Block from Dirty* is the time to receive a packet from the requester, check the directory, and send a packet to the dirty node. *Send Dirty Block to Requester* is the time to receive the block copy from the dirty node and send it to the requester with ownership (memory is not updated). *Nack* is the time to receive a packet, and negatively acknowledge it.

The latencies of second level cache read misses comprise three terms: on-board delays, memory delay, and interconnect delays. From Table 1 the following equations are applicable for the latencies (in clocks) of read misses in the absence of conflicts.

- SLC read miss from Local Home Memory: $36 + T_{miss}$
- SLC read miss from Remote Home Memory (2 hops): $48 + T_{miss} + 2 \times (24 + \alpha) + \beta \times 4$
- SLC read miss from Dirty Node (4 hops): $96 + T_{md} + T_{wb} + 4 \times (24 + \alpha) + \beta \times 8$

Figure 4 shows the decomposition of the latencies of a read miss on a dirty copy. The basic hardware latencies for read misses (α and $\beta = 0$) are 76 clocks (local), 136 clocks (home), and 288 clocks (dirty). There are small fluctuations in these latencies because some timings (such as accesses to the DRAM, FutureBus+, and the second level cache) may vary slightly. However, these simple estimates are surprisingly accurate as we will see in section 4.

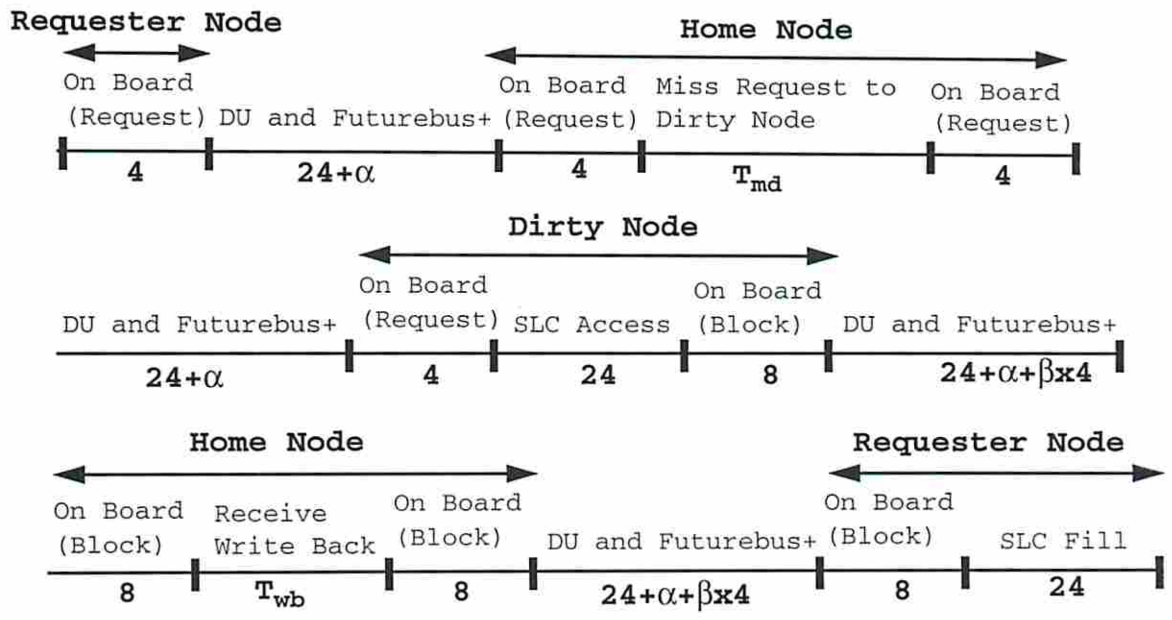

Figure 4. Timings of a Read Miss with Fill from a Dirty Node (no conflict)

2.2.3. Software Environment

We have developed a software environment for RPM to run applications using the ANL programming macros to express and manage parallelism such as the SPLASH-2 benchmarks [23]. Most of the library functions are statically linked directly from SUN's standard C and math libraries. A special operating system trap handler (tr0.o) written to support I/O functions intercepts the system call parameters and passes them over a communication line to a server running on the SUN workstation acting as the emulator's console. The server performs the request and cooperates with the system call stub running on the emulator's I/O board for possible data transfers. Upon completion, the server passes a return code to the stub, which is delivered to the application as a result of the system call. The whole mechanism is transparent to the application and the appearance of a UNIX kernel is entire.

Applications for the emulator are compiled using the standard procedures and UNIX tools for a SUN-4 workstation. Once compiled, applications can be downloaded into the shared-memory of the emulator at a fixed address. To start them, control is passed from the emulator's monitor to the application entry point, after switching from test mode to emulation mode.

3. PERFORMANCE METHODOLOGY

3.1. Multi-cycle Pclock Emulation

Each pclock of the CC-NUMA emulator is executed in eight clocks. The corresponding reduction in emulation speed is compensated by three advantages. First we can emulate complex mechanisms without sacrificing flexibility. Second, the latencies of the target are not limited by the latencies of the basic hardware: even if a large number of clocks are needed to move packets on the board and to emulate memories and complex directories, the target may still have very low latencies, expressed in pclocks. Finally, if processors were executing at the rate of one clock per pclock, the FutureBus+ would be a serious bottleneck.

3.2. Time Scaling

3.2.1. Latencies

To emulate systems with various processor, memory and interconnect technologies, the latencies measured in pclocks must be the same in the target and in the emulator. For example, if, in the target system, processors are clocked at 100MHz (pclock=10nsec), interconnect delay is 80nsec and DRAM access time is 100nsec, then the interconnect delay must be 8 pclocks (or 64 clocks) and each DRAM access must take 10 pclocks (or 80 clocks) in RPM.

The third column of Table 1 shows the latencies for a target systems with 100MHz processors and 100nsec DRAM access times. The values for memory/directory accesses assume that the directory is built in fast SRAMs, that the DRAMs can fetch an entire block in one cycle, and that the memory controller can be clocked at the speed of the processor and works in parallel with the DRAM access. The values of 8 pclocks and 20 pclocks (obtained by setting $\alpha = 40$ clocks and $\beta = 24$ clocks) for the latencies of the target interconnect were chosen from the hypernode of the Convex Examplar, which is an 8 100MHz processor cluster connected by a crossbar switch [8].

From the numbers in Table 1, and the expressions for the read miss latencies in the target, the number of pclocks to service a SLC read miss in the CC-NUMA emulation is 19.5 pclocks if serviced by the home locally, 49 pclocks if serviced by the home remotely, and 91 pclocks if serviced by a dirty node.

3.2.2. Bandwidths

Besides latency, bandwidth is also a critical performance parameter for any hardware resource. The bandwidth of the memories and directories in our CC-NUMA emulator is directly related to their access latencies since main memory on each board is neither interleaved nor pipelined.

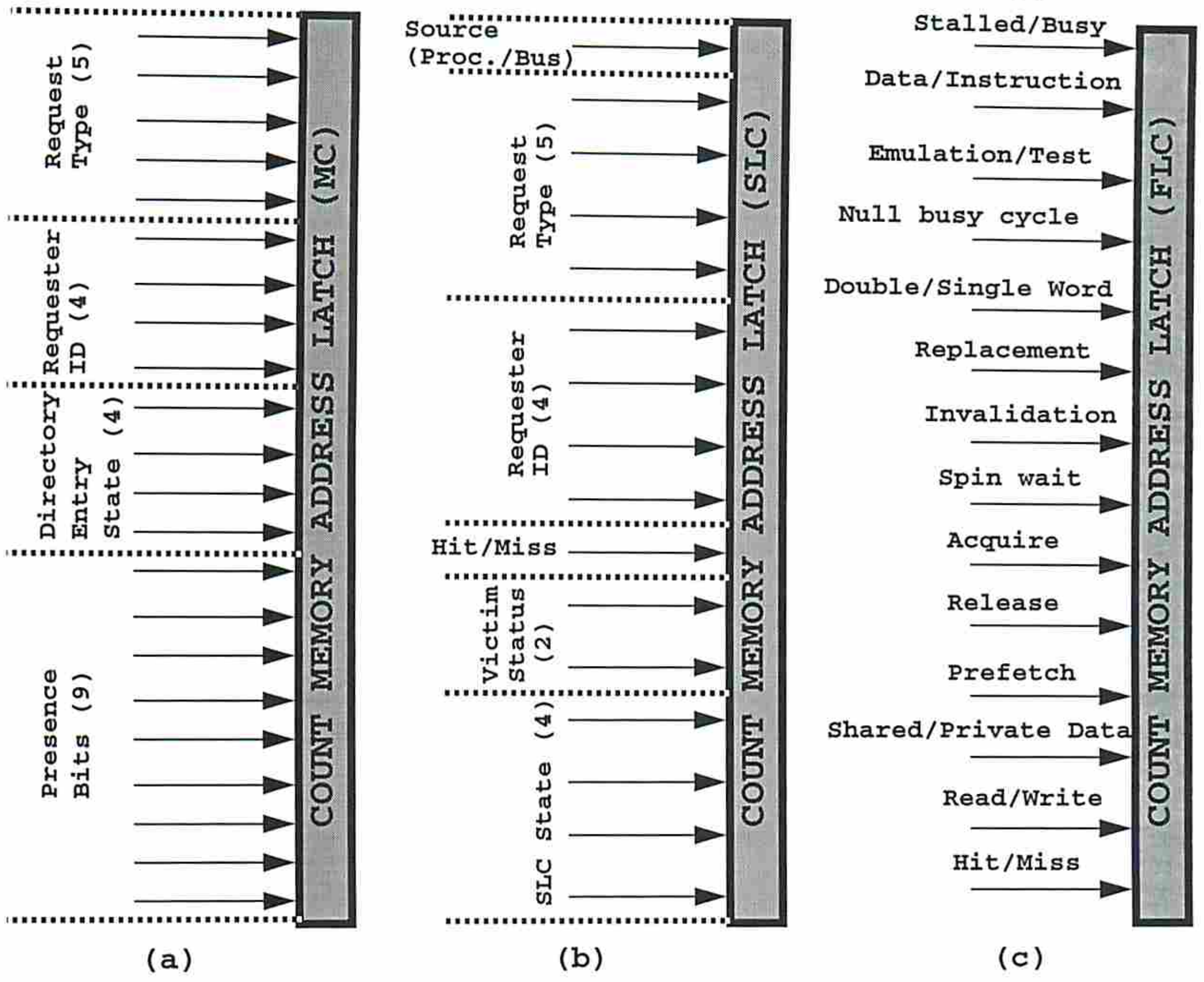

Figure 5. Addresses Used to Access Count Memory in MC, SLC and FLC
The main memory contains 4 M 32-bit counters (a); for each request type, each request source, and each global state of the block, there are 512 counters, one for every possible distribution of the copies among the processors. The second-level cache count memory contains 64K 32-bit counters (b); the location of the event counter depends on the request type, the source of the request (bus or local processor plus the ID of the processor), the state of the block in the cache, the state of the replaced block and the value of the hit/miss line. Finally the count memory of the first-level cache contains 16 K 32-bit counters (c), detecting various data or instruction cycles in the processor.

The current width of the FutureBus+ is 32-bits (extensible to 64 bits) and its bandwidth is 8 bytes per clock or 64 bytes per pclock. With a target pclock rate of 100MHz, this corresponds to an interconnect bandwidth of about 6 Gbytes per second in the target system. This is a huge, practically infinite bandwidth which increases proportionally to the target's pclock rate and this explains why we have observed very low interconnect traffic in our experiments (see Section 5). The FutureBus+ bandwidth can be reduced by changing the bus clock rate or the bus width and by padding packets with extra bytes. This feature allows us to run experiments under limited interconnection bandwidth.

3.3. Count Memory

The primary two mechanisms to collect performance statistics in RPM are memory-mapped counters (implemented in the FPGAs) and memory-mapped event-count memory (implemented in memory). Memory-mapped counters are simple counters that can be read/written as any memory location. Such counters can measure the total execution time, the utilization of the processors and the controllers, and can count events. So far, the only counter we have implemented in the FPGAs is a 40-bit pclock counter in the first-level cache controller of every processor to record the total execution time of each processor.

Event-count memories consist of a set of counters implemented in the SRAMs of the caches and the DRAM of main memory. Events are mapped to memory addresses, and, at the occurrence of a given event, the value stored at the corresponding memory location is incremented. Therefore, to implement event-count memories, it is necessary to have some extra hardware in each controller to read a counter from count memory, increment its value and write the updated value back to memory. In the first-level cache, an event counter is incremented at each pclock, as part of the pclock emulation, whereas, in the controllers of the second-level cache and of the main memory, an event counter is incremented on each controller transaction, which may take multiple pclocks. Figure 5 displays the composition of addresses to the event counters in the three memories, obtained by merging hardware signals corresponding to various events and resource states. Each processor board has three sets of event counters, one for each memory. At the end of an emulation run, all processors store their counters in their main memory in parallel; then the I/O processor uploads all counters to the SUN host, where the events are summed up and combined to obtain the counts of meaningful aggregate events.

3.4. Dealing with I/O

RPM has been configured with one I/O processor and eight execution processors, but since any board can act as an I/O processor, we can prototype systems with more I/O processors and less execution processors. To avoid distortions in the performance of complex commercial workloads and multiprogrammed workloads where I/O and processing are overlapped, we need to scale I/O, just as we scale memory and interconnect delays.

Currently disk I/O is performed at full speed. However, if the prototype is, say, 80 times slower than the target architecture then we need to increase the latency of I/O requests accordingly. This problem can be solved in software, by scaling the latency of I/O operations in a similar manner as was done for memory and interconnect latencies. When an I/O board receives an I/O request, it executes it as fast as possible, at full disk speed, and then it inserts an entry in an event queue with a timestamp indicating the time at which it is supposed to complete in the prototyped system. Periodically, the I/O processor retrieves entries from the queue and interrupts a processor to inform it of the completion of I/O. Besides emulating disk DMA transfers, the I/O board also maintains the time-of-day and tick clock and the console.

4. VERIFICATION, CALIBRATION AND PERFORMACE DEBUGGING

Once an architecture emulator is built, it is very important to verify that the characteristics of the architecture, such as cache sizes, cache organizations, and access times, are correct according to the original specifications defined for the target system. For an emulator built on top of RPM, it is also necessary to verify that the performance counters recorded during executions are accurate. In this section we present approaches used to calibrate and verify the correctness of the CC-NUMA emulator. We first verify the architecture characteristics of the emulator using a method derived from [19]. Then, we verify the emulator by comparing results obtained with a simulator and the results obtained with the emulator running some of the Splash-2 applications.

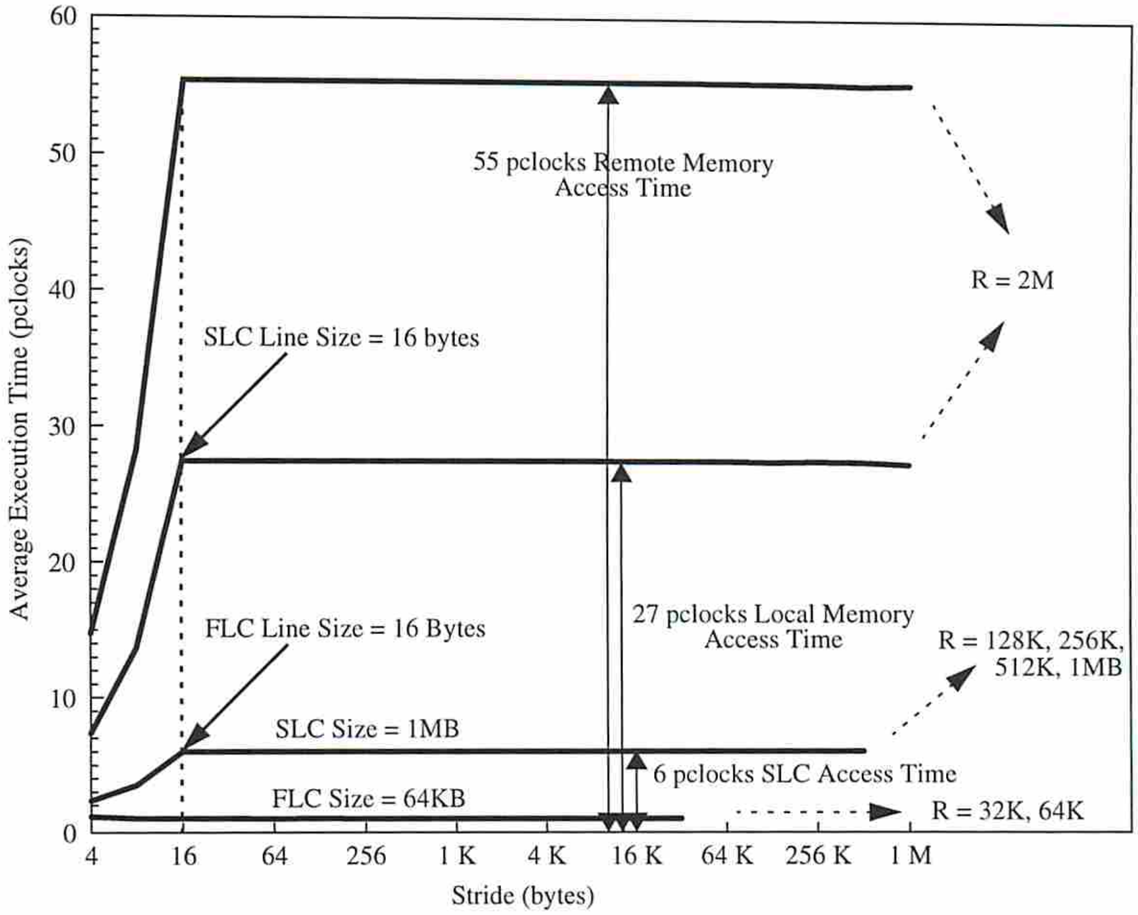

Figure 6. Performance profile of the CC-NUMA emulator

4.1. Verifying the Architecture Parameters

To verify that the emulator respects the specifications of the target architecture, Saavedra's technique described in [19] and based on micro-benchmarks has been applied to the emulator.

The micro-benchmark is made of multiple loops. In each loop, the processor reads every S-th (Stride) element of an array of size R. After each iteration, the stride is multiplied by 2 until it reaches R/2. When S reaches R/2, R is increased and S is reset to its initial value. For each value of R and S, we first run the micro-benchmark with the read access, and then we run it again with the read access replaced by a noop. The times are measured using the pclock counter provided by the RPM hardware in the first-level cache controller. The difference between the execution times of these two loops divided by the number of iterations yields the average read access time. A diagram representing the average time to read a single element for different values of R and S, is obtained. On the diagram, each curve corresponds to a value of R. The same benchmark with write accesses could be used to measure the timings of write accesses.

Different regimes are observed on a graph resulting from the execution of a micro-benchmark. For example, when the size of R is inferior to the size of the first-level cache, all elements accessed in the loop fit in the first-level cache and the average read time as a function of S is a constant, as shown by the bottom curve of Figure 6. By varying the values of S and R, other regimes where the read access hits or misses in the second-level cache are observed. These different regimes allow us to observe experimentally the cache size, organization, and latency times through various features of the diagrams, as pointed out in the figure. A more detail description of these regimes and of the way to interpret the graphs can be found in [19].

Figure 6 shows the graph obtained for the CC-NUMA emulator. From this figure, we can infer that the first-level cache is direct-mapped and that its size is 64Kbytes; we can also verify that the second-level cache is direct mapped with a size of 1MByte and that the lines in both caches are 16 bytes. Access times are confirmed as follows. A read hit to first level cache takes 1 pclock. A read miss in the first-level cache that hits in the second level cache takes 6 pclocks. A read miss in the first- and second-evel caches takes 27 pclocks if it is serviced from the local home memory and 55 pclocks if it is serviced from a remote home memory. These experimental timings are more accurate than the timings estimated by counting cycles, as we did in Section 2. However, the two sets of estimates are in close agreement. Adding the first-level cache access time, the second-level cache miss detection time, the second-level cache restart time and the first-level cache fill time (a total of 8 pclocks), to the second-level cache miss times computed in section 3.2, we find a total of 27.5 pclocks (vs. 27) for a read access with a local second-level cache miss and 57 pclocks (vs. 55.5) for a read access with a remote second-level cache miss.

4.2. RPMsim

RPMsim is an approximate simulator of the CC-NUMA emulator. This simulator was written in C using the CSIM package [20] and the Cache-Mire SPARC interpreter [5]. It consists of several modules implementing the processor, first- and second-level caches, memory controller, main memory, and the shared bus.

The processor module is a simulator for the instruction set of the SPARC architecture. It has no detailed simulation of execution pipelines and it issues a memory cycle upon every invocation, including instruction fetches. The fact that we do not simulate the pipeline introduces some error in the simulation, because, in the real machine, (1) instructions do not always complete in one processor cycle, (2) some cycles are null cycles (null cycles are mostly due to annuled instructions in branch delay slots), and (3) none of the extended precision instructions and a few double precision instructions are implemented in the simulator[2].

5. EMULATION RESULTS

In this section, we present some of the performance data collected on the CC-NUMA emulator. In order to observe the effects of instruction misses and local data accesses --which are often not included in simulation studies-- the code and private data of processes are allocated at the top of the address space, followed by the shared data segment. Addresses are not interleaved so that consecutive addresses are located in the same board. The allocation of memory affects the speedup. For programs with very small data set sizes, most if not all data are in board 0, which then reaches high utilization values. Additionally, for program having high miss rates for local data and instructions, the instruction and data (read/write) stall times are large.

All programs except MP3D are taken from the SPLASH-2 benchmark suite [23]. The selected data sets are larger than the default values for Cholesky and FFT and smaller for Barnes-Hut, FMM, Radiosity, Radix, Raytrace, and Volrend.

5.1. Emulation Speed

RPM currently emulates the execution of 625,000 target pclocks per second at the 5 MHz clock rate. The clock rate is limited by the speed of the FPGAs, which depends mostly on the quality of the synthesis tools. We expect to raise this clock rate to 10MHz at least in the near future. Figure 7 (a) shows the emulation times of all benchmarks. The emulation times vary from 10 to 140 minutes on one RPM processor and from 2 to 20 minutes on eight processors. FMM takes the most time and Barnes-Hut comes second. This graph gives a feel for the speed of RPM. Figure 7 (b) shows the emulation rate of the CC-NUMA emulator on RPM, for one set of experiments. This rate (in terms of million of instructions emulated per second) is directly proportional to the speedup of the target system, given the time-scaling methodology. The best emulation rate is achieved for Volrend in which RPM reaches nearly 3 million emulated instructions per second, which is 60% of RPM's peak emulation rate and the worst case is for MP3D with 833,000 emulated instructions per second.

Figure 7 (c) displays the utilizations of the memory controller on board 0. To find the utilization of each memory controller we sum up the times the memory controller takes to process specific request types including the secondary messages and their replies. The counters in count memory yield the number of time a request of a given type has been executed and the time taken by each request is given by Table 1. MP3D has the worst-case memory utilization. In MP3D the memory controller utilization of board 0 reaches 94.5% on 8 processors. Due to the large number of messages generated by MP3D the memory controller becomes a hot spot and the read and write stall times increase with each additional processor.

Figure 7 (d) shows the FutureBus+ utilization. This utilization is computed from the event counts using a similar procedure as for the memory controllers. The FutureBus+ is very much underutilized. In order to emulate limited bandwidth systems we can artificially increase the size

2. These functions do not appear in the code generated by the GCC compiler. However, some of them may occur in math library functions linked together with the application. To handle the situation, these functions have been implemented as application traps to the simulator. Because they make heavy use of registers and have few memory accesses, the caching behavior of the application is not severely affected, but the application execution time may be affected

of each packet sent to the FutureBus+.

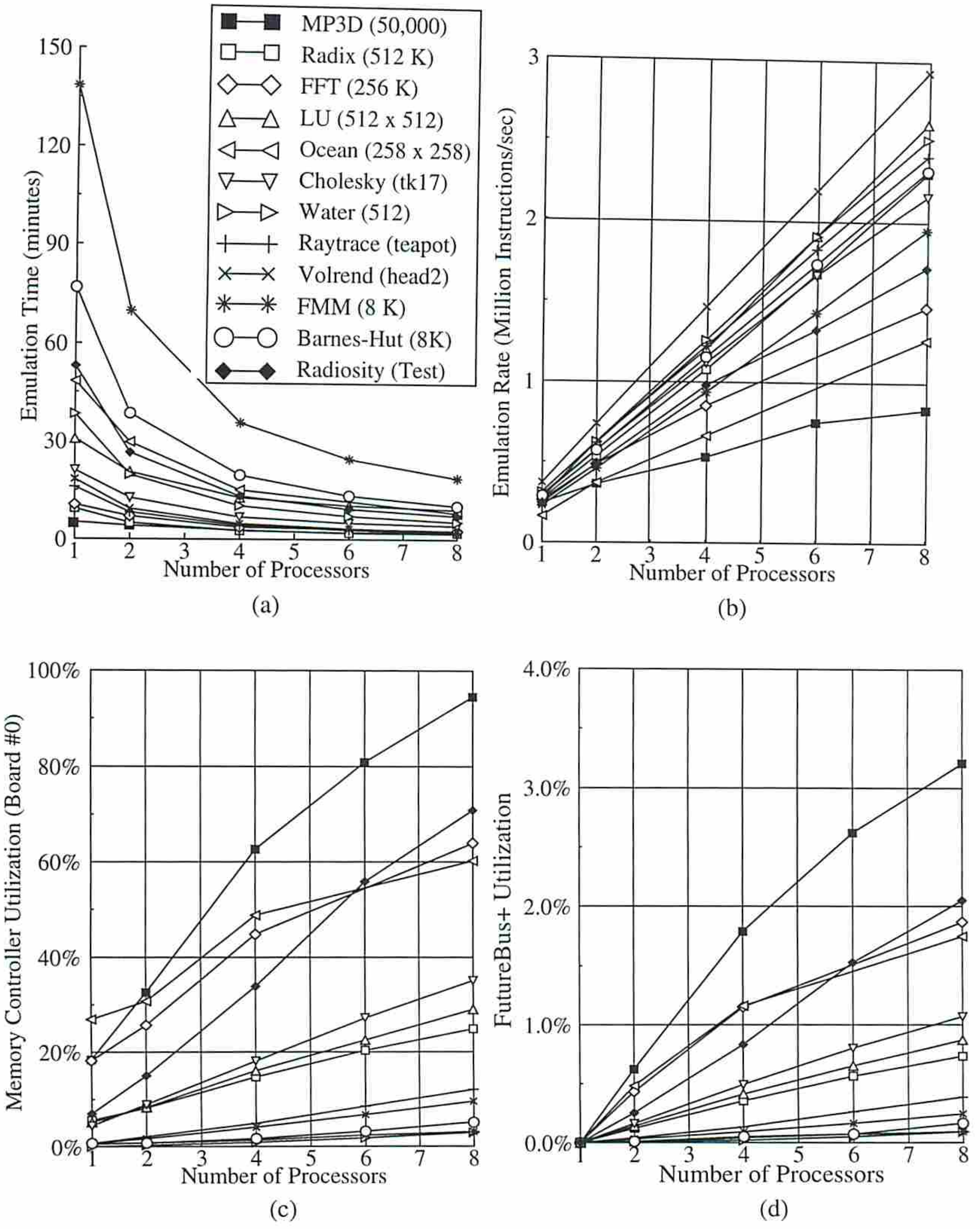

Figure 7. CC-NUMA Emulation Performance
(a) Emulation Times; (b) Emulation rates; (c) Worst-case Memory Controller Utilization (board 0); (d) FutureBus+ Utilization.

Programs such as Volrend, Water-nsquared, Raytrace, Barnes-Hut, and Radix, which are known to have good speedup run at high emulation rates on RPM, in contrast with MP3D and FFT. Radiosity, which usually shows good speedup, has a very large data set size and its instruction miss rate is higher than in all other benchmarks, which in turn results in up to 18% instruction stall time on an 8-processor system. This high instruction miss rate may be partially explained by the nondetermistic behaviour of Radiosity. There may also be some trashing between data and instruction accesses. Although LU has normally worse speedup than FFT, it achieves a very good emulation rate. The main reason for this is that the work is not uniformly distributed among processors in LU. When a processor is waiting for other processors (busy waiting), it stays in a tight loop with a single data access, which hits most of the time. During this time, RPM is close to its peak emulation rate.Ocean has normally better speedup than FFT, but it exhibits a large local data traffic [23]. We can also see from Figure 7 (c) that the utilization of the memory controller on board 0 reaches 60% in Ocean. By contrast, Cholesky has usually lower speedup, but still achieves better emulation rate in these experiments. This is mostly because of Cholesky's smaller data set size and lower data traffic.

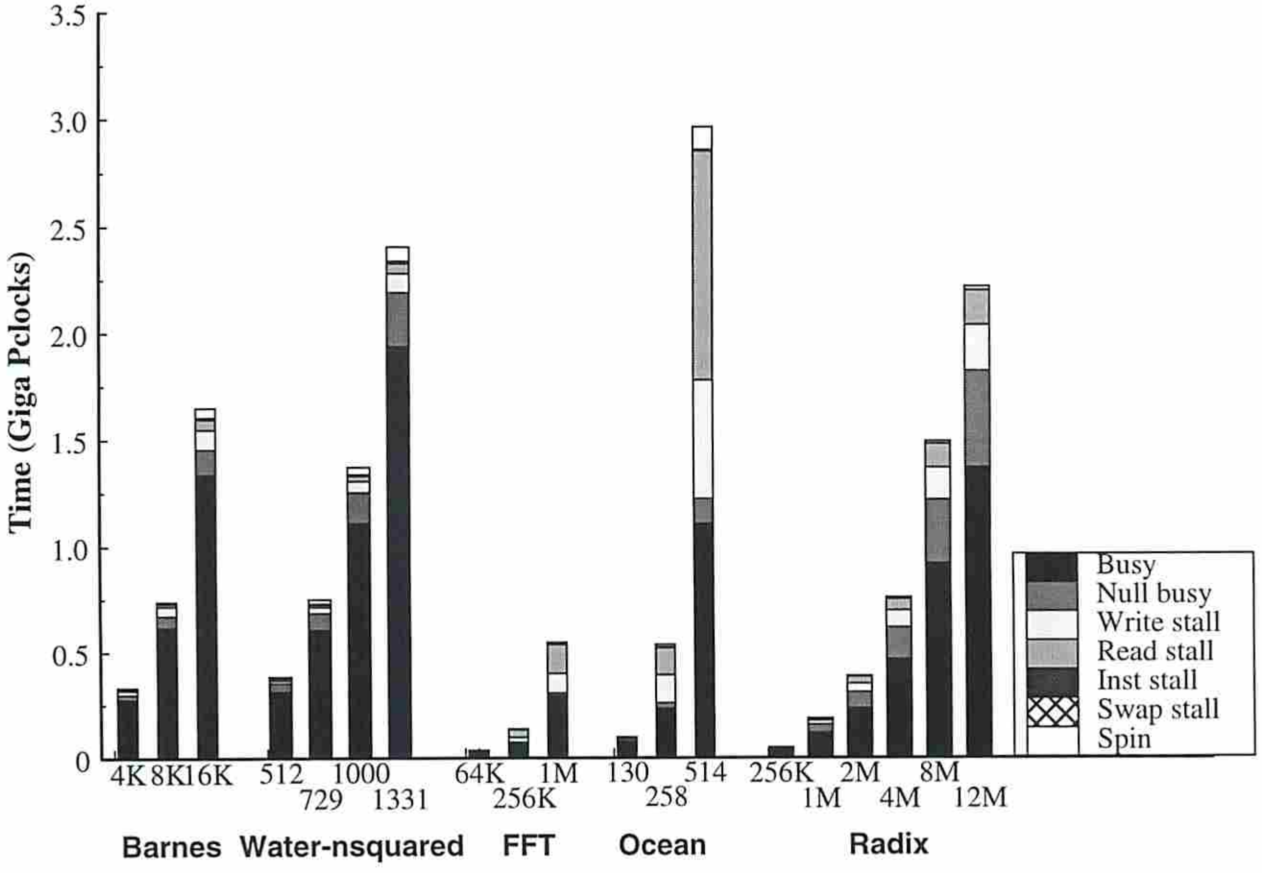

Figure 8. Total Execution Times of the Five Benchmarks on a 4-processor System

5.2. Varying Data Set Sizes

We have also examined the effects of increased data set sizes for Barnes-Hut, Water-nsquared, *Ocean,* Radix, and FFT on a 4-processor system. For Barnes-Hut we have used 4K, 8K, and 16K particles. For Water-nsquared the number of molecules were changed from 512 up to 1331. We ran Radix with 256K, 1M, 2M, 4M, 8M and 12M integers with a radix of 1024. For FFT 64K, 256K, and 1M points are used. We simulated Ocean with 130x130, 258x258 and 514x514 grid sizes. The memory used by these benchmarks are 8, 16, and 32 MB for Barnes-Hut, 550KB, 775KB, 1MB, and 1.4MB for Water-nsquared, 2, 8, 16, 32, 64, and 96 MB for Radix, 3, 12, and 48 MB for FFT, and 3.7, 14, and 56 MB for Ocean.

When increasing the data set sizes we were limited by three factors: (1) Execution time, (2) Memory size, and (3) Completion of the program execution. Among the selected benchmarks Radix, Ocean, and FFT are memory-bound. On a 4-processor RPM we have 128 MB memory and for these three benchmarks we could not run bigger problems because of our memory limitations. Water-nsquared and Barnes-Hut are compute-bound programs and we are currently limited by their execution time. In the current RPM configuration we are limited to about two hours of execution time because of the 32-bit resolution of the event counters in count memory. In the near future we plan to interrupt the IO processor periodically every two hours to freeze the whole system, upload all the counters to the host Sun SPARC Workstation, and resume the program. We expect this mechanism to add very little distortion to the collected performance statistics (it takes a couple of pclocks to stop all processors) while increasing the tolerance to crashes.

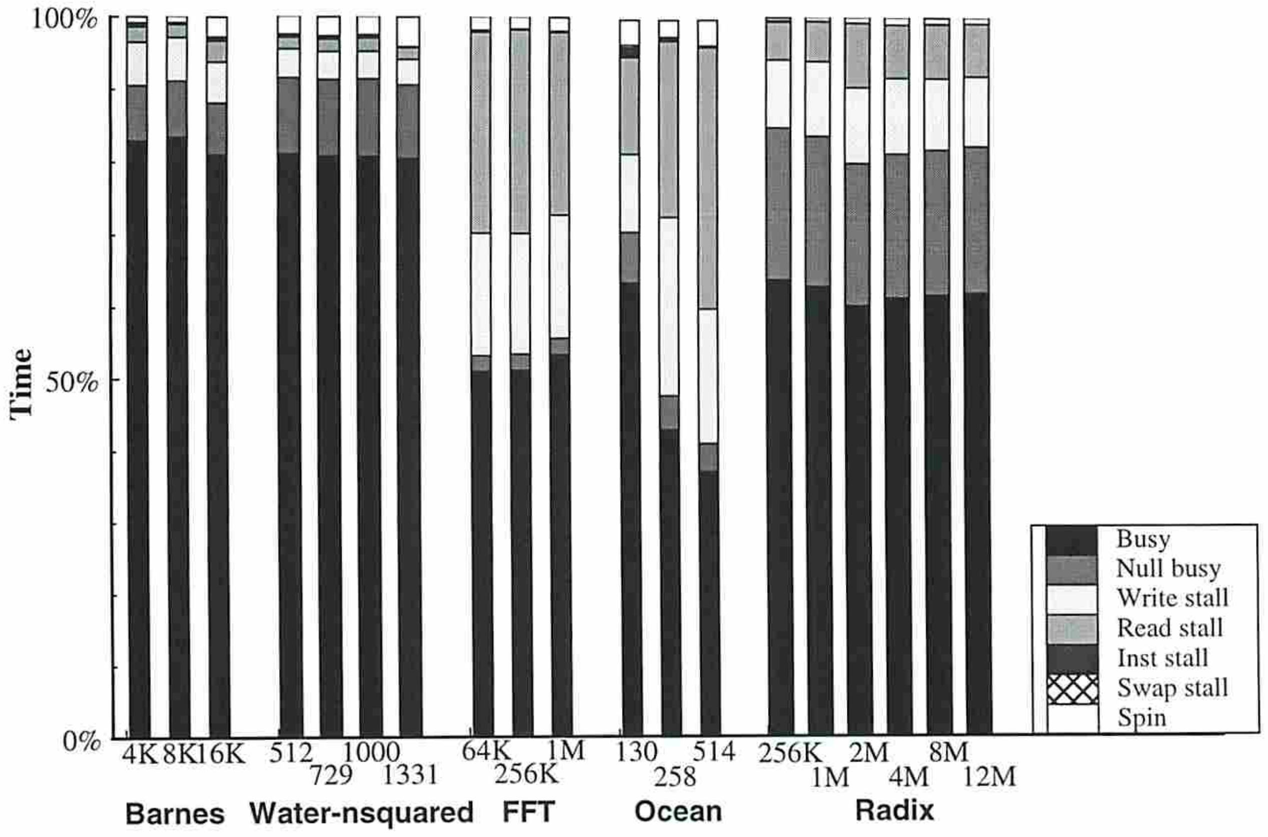

Figure 9. Normalized Execution Time of the Five Benchmarks on a 4-processor System

Figure 8 and Figure 9 show the total and the normalized execution times of the five benchmarks on four processors. The run for Ocean 514 took the longest time, about one hour and 20 minutes on RPM. The busy time is the total execution time including instruction executions when all data read/write accesses hit in the First Level Cache. The Null Busy time is the sum of all the times when the processor can not execute a useful instruction because of an pipeline interlock or a unused branch delay slot. The write stall, read stall, swap stall and instruction stall times are the times during which a processor is blocked pending the completion of a write, a read, a swap and an instruction fetch in the memory system. The spin time is the total synchronization time excluding the swap stall time.

Barnes-Hut and Water-nsquared have very high processor utilizations and their memory behavior is not affected by the increased data set sizes, because their working set sizes are much smaller than 1 MB and therefore they fit in our second-level cache easily. Pipeline overheads dominate. The memory behavior of Radix is also unaffected data set size increases. This result can be explained mainly by the communication-to-computation ratio of Radix. This ratio is constant as long as the number of processors is fixed. One interesting point about Radix is the large amount of time it spends in Null Busy cycles. A large number of branch delay slots are not utilized. In this case, processor overheads dominate memory overheads.

FFT is another benchmark whose memory behavior is unaffected by increasing the data set sizes. In FFT the most important data set size is one row of the matrix [23], which fits our second level cache. On top of that the communication-to-computation ratio decreases logarithmically with increasing data set sizes. We have run FFT with three data set sizes: 64K points (3 MB), 256K points (12 MB), and 1M points (48 MB). The changes in communication-to-computation ratio from 64K points to 256K points and from 64K points to 1M points are 0.9152 and 0.8437, respectively. We see the slight effect of the reduced communication-to-computation rate in terms of increasing busy times for larger data sets.

Ocean is the only benchmark which shows a dramatic difference in its memory behavior as the data set size is increased. Ocean has a large working set size which does not fit into our 1 MB second-level cache for the 514 x 514 grid size. (It is possible that the working set size for the 258 x 258 grid does not fit in our second-level cache as well.) As the data set size increases the miss rates in the second-level cache increases from 0.4% to 1.3% and then to 2.3%. As reported in [23], the local data traffic increases drastically with the data set size for 4-processor configurations and a 16 bytes block size. Because of the data allocation, the service time of the read/write misses to local data as well as the data traffic affect execution speed.

5.3. Simulation Experiments

Table 2 shows some comparisons between RPM and RPMsim. All the experiments reported in Table 2 have been conducted for four processors clocked at 5MHz. The simulations were performed on a SUN SparcStation-10 with 128 Mbytes of memory (a 50 MIPS machine). The time values correspond to the parallel section of the benchmark only. The average emulation speedup is close to 170 and it should more than double when eight processors are used instead of four. Also, this speedup should again quadruple if we reach the 20MHz clock for which the boards of RPM-2 were designed. Applications with high busy times (Barnes-Hut, Volrend, Raytrace, Water) achieve large emulation speedups.

In actuality, comparisons with simulations are quite meaningless, since the outcome depends on the level of details in the simulations, on the efficiency of the simulation and on the speed of the machine on which the simulation runs. The emulation speedups would be much higher if the emulation was compared to cycle-by-cycle simulation.

Benchmark	Parameters	Emulation time(sec.)	Simulation time (sec.)	Emulation speedup	Parallel run time (10^6 clk.)	Simulated parallel run time (10^6 clk.)	Simulation accuracy (%)
barnes	4k	533	94483	177.3	333.1	343.1	+3.0
cholesky	bcsstk14	37	6546	176.9	23.5	24.5	+4.3
fft	-m16 -n4096	50	6444	128.9	31.3	31.7	+1.3
lu	-n128	10.4	2163	207.9	6.5	6.9	+6.2
radiosity	test	848	105083	123.9	529.9	324.5	-38.8
radix	262144 int.	74	11253	152.1	46.4	40.8	-12.1
raytrace	teapot	252	49973	198.3	157.3	166.2	+5.7
volrend	scaleddown4	70	11588	165.5	43.8	35.2	-19.6
water-nsq	343	303	56498	186.5	189.4	192.3	+1.5

Table 2. Comparison of Emulation with Simulation

The simulation accuracy ranges from surprisingly good (Barnes-Hut, Cholesky, FFT, LU, Raytrace, Water) to moderate (Radix, Volrend) and bad (Radiosity). In the emulation, Radix spends 15 to 20% of its time in null busy cycles, which are unaccounted for in the simulation. Volrend makes heavy use of mathematical library functions, which are not simulated. Hence the simulated execution times are too optimistic. Finally, Radiosity seems to have followed different convergence paths to the solution in the simulation and in the emulation. Clearly, we will have to include pipeline effects in the simulation and simulate more of the mathematical library functions if we want to improve the accuracy of the simulator.

6. ABOUT THE METHODOLOGY

There are many aspects of the project that could have made the methodology more effective. We could have built a system with a large number of processors and a reconfigurable network to increase the generality of the prototypes. We could also have clocked RPM faster. However the project would have been much more expensive and risky. In this section we discuss three aspects of the methodology: flexibility, performance and generality.

6.1. Flexibility of RPM

There is considerable flexibility in configuring RPM. Nonetheless, the hardware prototypes which can be built on RPM have some basic limitations. Most of these limitations are also present in traditional hardware prototypes. First, the overall hardware configuration must be the same in the prototype and in RPM. For example, we cannot prototype systems with three levels of caches or systems with shared caches. Second, we cannot prototype systems with more than eight execution processors. Third, we cannot implement specific interconnections. We can only change the latencies as well as the bandwidth, which are the most important factors of interconnect performance.

Fourth, the processor architecture is fixed. Thus RPM prototypes are not appropriate to evaluate the effects of processor architecture on multiprocessor performance. An issue might be non-blocking loads (as opposed to prefetches, which we implement.) The depth of the processor pipeline or the issue width of superscalar processors may also have a serious effect on multiprocessor performance, as our own results show. Finally, RPM cannot emulate multiprocessor systems with non-SPARC processors.

To conclude, whereas RPM cannot prototype *any* multiprocessor architecture, it is much more flexible than traditional hardware prototypes and can be used to explore a large number of practical and useful multiprocessor architectures.

6.2. Performance

The comparison of RPM's speed with simulation speed is not easy to make. RPM emulates an architecture in hardware and in all its details. One major advantage of a simulation is that it can be as detailed or as abstracted as desirable. Depending on how abstracted the simulation is, RPM is slower or faster than the simulation. For example, simulations of system code on SimOS run with a slowdown between 10 times and 50,000+ times, depending on the level of details [18]. The detailed simulation of the SGI Challenge to verify the hardware and its performance on small operating system kernels reportedly ran at about one cycle of the target per second [10]. On the other hand, trace-driven simulations can be extremely fast.

RPM could emulate the execution of 2,500,000 target pclocks per second when the 20 MHz clock rate is reached, independent of the number of processors. The peak emulation rate is 20 Million target instructions per second, with 8 boards at 20 MHz. However, the emulation rate is strongly affected by the characteristics of the application. RPM approaches its peak emulation rate for programs with little memory activity and with very high computation-to-communication ratio.

The speed bottleneck in RPM prototypes is the FPGAs. The clock rate is limited by the speed of the FPGAs, which depends mostly on the quality of the synthesis tools.

Like any piece of hardware, RPM performance degrades with time with respect to target machines due to the fast pace of technology improvements. However, if the boards were designed to run at a higher clock rate, RPM could follow the technology curve and its speed could be improved over the years by upgrading the FPGAs, since FPGA suppliers maintain pin-to-pin compatibility. FPGAs have improved dramatically over the years and all indications are that this trend will continue unabated.

6.3. Generality

There is nothing in the methodology of RPM that would prevent us from building better hardware prototyping platforms in the future. A different processor will have to be chosen. One drawback will be that more and more of the memory hierarchy will migrate on chip, thus reducing the flexibility of RPM but simplifying the design of prototypes. Observability will also be affected (as in any hardware prototype). However, we note that the current trend is to include more and more performance evaluation hooks in processor chips.

6.4. A "Dream" Machine

In the RPM project, we have been quite conservative in the design of the hardware, because of the novelty of the approach, the limited budget, and the risks associated with aggressive hardware designs in an academic environment. Of course, larger, faster emulation platforms could be built.

If we used modern processors with on-chip first-level caches, the prototypes would be simplified and would run faster[3]. The reason is that, in the current machine, the first-level cache controller must execute in one pclock all the details of every processor access, including stopping/starting the processor, translating addresses in the TLB, accessing the cache, and updating count memory. Thus the first-level cache controller is by far the most complex and dictates the number of clocks per pclock. On the other hand each second-level cache access takes several pclocks.

Instead of a bus interconnection, we could build a bit-serial, hypercube interconnection (implemented for example with S3.mp's TIC chip [14]), which could emulate complex, point-to-point interconnections more faithfully.

Finally, we would design the boards to run at much higher frequency (100MHz) in order to be able to upgrade the boards as faster FPGAs become available.

With these considerations in mind a 128-processor machine with two clocks per pclock and clocked at 20MHz (upgradable to 100MHz) could be built today. Such a machine would have a peak emulation rate of 1.28 billion emulated instructions per second (counting one instruction per cycle per processor), and even more, if processors execute more than one instruction per cycle.

7. CONCLUSION AND FUTURE WORK

Multiprocessor emulation is an alternative to prototyping and software simulation, both in flexibility and observability. In this paper we have related our initial experience with emulation technology based on field-programmable gate arrays.

RPM is a configurable hardware platform or substrate on top of which various multiprocessor architectures can be implemented in hardware. Because the controllers are made of FPGAs, architecture parameters as well as performance measurement hardware can easily be changed. We have described the methodologies and illustrated them with our first CC-NUMA emulation on RPM.

In the future, we plan to continue the prototyping work, which will result in three broad contributions:

• Demonstration and evaluation of hardware prototypes of shared-memory architectures with various hardware/software mechanisms by porting an entire system with multiprogrammed and com-

3. As a trade-off, some flexibility and some observability would be lost

mercial workloads

• Comparison of these machines and of various DSM systems for scientific workloads with large data set sizes as well as system and commercial workloads, on the same hardware substrate.

• Development and demonstration of a viable methodology for the rapid and cost-effective hardware prototyping of multiprocessor systems using FPGA technology.

Recently we have completed the port of Solaris 2.4 on RPM-2 and we expect to run non-scientific workloads soon.

Acknowledgments. Funding for this work has been provided by the National Science Foundation under Grants MIP-9223812 and MIP-9633542. Alain Gefflaut is funded by INRIA/IRISA (France). Besides the authors, several individuals have contributed to the project. In particular, we want to thank Per Stenström from Lund University (Sweden) and Massoud Pedram from EE-Systems, U.S.C. Luiz Barroso, Jacqueline Chame, Sasan Iman, and Krishnan Ramamurthy participated in the design of the RPM emulator. Through their University Program several companies helped reduce the cost of the hardware and software needed for the project. These companies are Advanced Micro Devices, Synopsys, Viewlogic, Axil Workstations and Xilinx. Finally, John Granacki from ISI offered the services of EZFAB, which is part of the ARPA-sponsored Systems Assembly Project.

8. REFERENCES

[1] Agarwal, A., Bianchini, R., Chaiken, D., Johnson, K.L., Kranz, D., Kubiatowicz, J., Lim, B.-H., Mackenzie, K., Yeung, D., "The MIT Alewife Machine: Architecture and Performance," *Proceedings of the 22nd International Symposium on Computer Architecture*, June 1995.

[2] Barroso, L.A., and Dubois, M.,"Performance Evaluation of the Slotted-Ring Multiprocessor," *IEEE Transactions on Computers*, Vol. 44, No. 7, pp.878-890, July 1995.

[3] Barroso, L.A., Iman, S., Jeong, J., Öner, K., Ramamurthy, K., and Dubois, M., "RPM: A Rapid Prototyping Engine for Multiprocessor Systems," *IEEE Computer*, pp. 26-34, February 1995.

[4] Blumrich, M.A., Li, K., Alpert, R., Dubnicki, C., Felten E., and Sandberg, J.,"Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer," *Proceedings of the 21st International Symposium on Computer Architecture*, April 1994.

[5] Brorsson, M., Dahlgren F., Nilsson, H., and Stenström, P. "The CacheMire Test Bench: A Flexible and Effective Approach for Simulation of Multiprocessors", In *Proceedings of the 26th Annual Simulation Symposium*, pp. 41-49, March 1993.

[6] Catanzaro, B., *Multiprocessor System Architectures*. Prentice-Hall, 1994.

[7] Censier, L., and Feautrier, P.,"A New Solution to Coherence Problems in Multicache Systems," *IEEE Transactions on Computers*, Vol. 27, No. 12, pp.112-118, December 1978.

[8] Convex, "Examplar Architecture". Convex Press, 1993.

[9] Dubois, M., Scheurich, C., and Briggs, F.A., "Synchronization, Coherence, and Event Ordering in Multiprocessors," *IEEE Computer*, Vol. 21, No. 2, pp. 9-21, February 1988.

[10] Galles, M. and Williams, E.,"Performance Optimization, Implementation, and Verification of the SGI Challenge Multiprocessor," *Proc. of the 27th Hawaii Int. Conference on System Sciences*,

Vol. 1, 1994, pp. 134-143.

[11] Kuskin, J., et al. "The Stanford FLASH Multiprocessor," *Proceedings of the 21st International Symposium on Computer Architecture*, April 1994.

[12] Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W.-D., Gupta, A., Hennessy, J., Horowitz, M., and Lam, M.S., "The Stanford DASH Multiprocessor," *IEEE Computer,* pp. 63-79, Vol. 25, No. 3, March 1992.

[13] Li, K. and Hudak, P., "Memory Coherence in Shared Virtual Memory Systems," *ACM Transactions on Computer Systems*, Vol. 7, No. 4, pp. 321-359, November 1989.

[14] Nowatzyk, A., Aybay, G., Browne, M., Kelly, E., Parkin, M., Radke, B., and Vishin, S.,"The S3.mp Scalable Shared-Memory Multiprocessor," *Proceedings of the 1995 International Conference on Parallel Processing*, pp. I.1-10, August 1995.

[15] Öner, K., Barroso, L.A., Iman, S., Jeong, J., Ramamurthy, K., and Dubois, M., "The Design of RPM: An FPGA-based Multiprocessor Emulator," *Proceedings of the 3rd ACM International Symposium on Field-Programmable Gate Arrays*, June 1995.

[16] Pong, F., Stenström, P., and Dubois, M.,"An Integrated Methodology for the Verification of Directory-based Cache Protocols," *Proceedings of the 1995 International Conference on Parallel Processing*, pp. I.158-166, August 1994

[17] Reinhardt, S., Hill, M.D., Larus, J.R., Lebeck, A.R., Lewis, J.C., and Wood, D.A., "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers," *Proceedings of the ACM Sigmetrics Conf. on Measurements and Modeling of Computer Systems,* pp. 48-60, May 1993.

[18] Rosenblum, M., Herrod, S.A., Witchel, E., and Gupta, A., "Complete Computer System Simulation: The SimOS Approach," *IEEE Parallel and Distributed Technology*, Winter 1995, pp. 34-43.

[19] Saavedra, R.H., Gaines, R.S., and Carlton, M.J., "Micro Benchmark Analysis of the KSR1", *Proceedings of Supercomputing'93,* Portland, November 1993.

[20] Schwetman, "CSIM: A C-based, Process-oriented simulation Language," *Proceedings of 1986 Winter Simulation Conference*, pp. 387-396, 1986.

[21] Stenström, P.,"A Survey of Cache Coherence Schemes for Multiprocessors," *IEEE Computer*, Vol. 23, No. 6, pp. 12-24, June 1990.

[22] Trimberger, S.,"A Reprogrammable Gate Array and Applications," *Proceedings of the IEEE*, Vol. 81, No. 7, pp. 1030-1041, July 1993.

[23] Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., and Gupta, A.,"The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proceedings of the 22nd International Symposium on Computer Architecture*, June 1995.