Simulation of Multiprocessor IP Router Using Augmint

Xi Chen and Kun Yan
Department of Computer Science
University of California
Riverside, CA 92521
{ xichen, kyan }@cs.ucr.edu

ABSTRACT

The popularity of the Internet has resulted in the Internet traffic growing rapidly during the past several years. The increasing importance of telecommunication motivates the development of fast IP routers, and several improvements have been previously introduced [1]. However, only a few of them have been proposed with detailed study on multiprocessor performance evaluation. In this project, we study the multiprocessor architectures of IP routers based on cache-coherence shared memory technologies. We use Augmint, a fast execution driven multiprocessor simulator, to simulate a multiprocessor IP router application, run it on two different multiprocessor architectures, SMP and CC-NUMA, and evaluate their performance with different architecture parameters. Thus, we design the multiprocessor IP router applications, modify the cache-memory backend of Augmint to simulate CC-NUMA and SMP architectures, compile and simulate the router applications with Augmint and give a detailed study on performance analysis.

1. INTRODUCTION

A router is a node that is connected to two or more networks. The key function of a router is to forward messages or packets toward the destination node based on its address. The architecture of a generic router [5], shown in Figure 1, consists of three components:

- Line cards that have the router physically attached to various networks employing different data-link technologies.
- A routing processor that participates in routing protocols and creates a forwarding table that is used in packet forwarding.
- A backplane that interconnects the input line cards, the output line cards and the routing processor.

High-speed routers have been a bottleneck for high-speed networks. With the exponential growth of IP-based networking and emerging data-link technologies, router designers are working on the bottleneck imposed by lengthy-routing table lookups, shared bus architectures and software-based forwarding. A number of advances [5] have been done on new router designs.

- With the advance in silicon chips, implementation of application-specific integrated circuits designed for IP routing is not difficult.
- The third generation of routers is designed with the shared bus replaced by a switch fabric, which increases internal bandwidth and allows packets to move in parallel through the router, finally increasing overall throughput.
- The number of memory accesses a router requires to find the matching route entry and the speed of the memory together determine the speed of a route lookup algorithm. The performance of route lookup algorithms has been improved in several ways, such as hardware-oriented techniques, table compaction techniques and hashing techniques.

Although much work has been done on IP routers, the simulation approaches can not evaluate the impact of architectural features, such as speculative execution, multiprocessing, instruction level parallelism, and cache architectures on the performance of routers. In [2], Bhuyan, et.al. proposed an approach of execution-driven simulation for IP router architectures. They incorporated a model of an IP router into an execution-driven simulator RSIM and evaluated the application performance. The goal of this project is

- Modifying the backend of the simulator Augmint and simulating the cache-coherence shared memory protocols based on CC-NUMA and SMP,
- Designing a simplified multiprocessor IP router, which will be incorporated into Augmint as a simulated application,
- Feeding the simulation with real internet packet traces from NLANR [3] and study its property in simulation process, and
- Analyzing the performance of the simulated multiprocessor IP router with the experimental results.

The rest of the paper is organized as follows. First, we present an overview of the simulator Augmint in section 2. In section 3, we describe the multiprocessor router application. The simulated architectures for the router application, CC-NUMA and SMP, are intruduced in section 4. The experimental result and its analysis are given in senction 5. The last section presents the conclusion and future work.

2. OVERVIEW OF AUGMINT

Trace-driven and execution-driven simulations are two main approaches to multiprocessor simulation. Trace-driven simulation uses traces of instructions or memory reference of real applications to drive memory simulation, while execution-driven simulation uses the execution of the applications with the simulation of the target systems, which allow the target systems to influence the behavior of the applications, thus yielding more realistic results than trace-driven simulation. Most publicly available execution-driven simulation tools only simulate RISC architectures. These RISC-based execution-driven simulations do not capture the instruction mix and memory reference patterns of CISC architectures. To fill this gap, Augmint [6] provides a public simulation toolkit for Intel x86 architectures and runs on x86-based uniprocessor systems under different operating systems, like UNIX/Linux, Windows NT or Solaris.

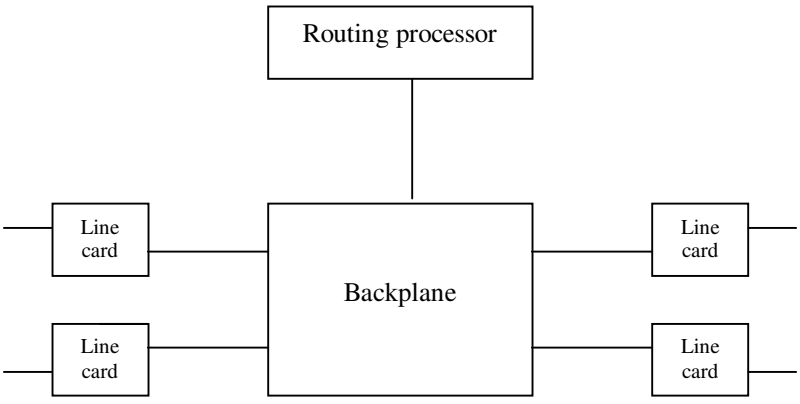

Figure 1: The Basic Architecture of a Router

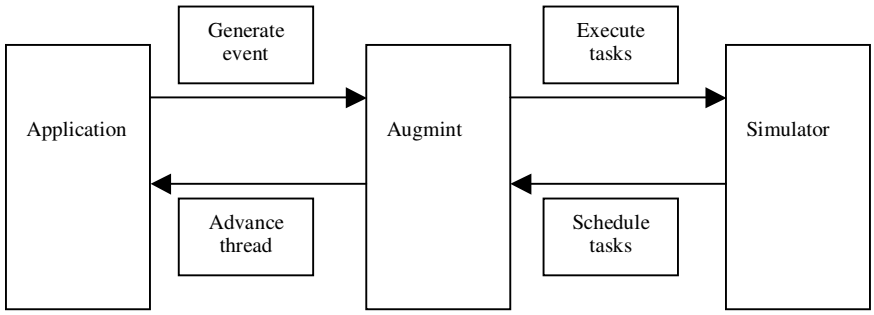

Figure 2: Augmint Simulation Environment

2.1 Structure of Augmint Simulation

As shown in Figure 2, the Augmint simulation environment consists of three modules: an application, Augmint and an architecture simulator. The code of the three modules, an augmented application, Augmint and an architecture simulator, are linked into one executable. The augmented application is composed of the application code and augmented code used to interact with the Augmint at specific events such as lock obtain and reclaim, memory references and so on. It is Augmint's job to handle these events, and normally it creates and schedules tasks on a centralized task queue for these events, which in turn can be simulated by the simulator. The architecture simulator includes a bunch of functions that define the actions and timing for various tasks created and called by Augmint such as loads, stores and bus requests.

First, the Augmint thread allocates application thread structure, creates a number of resources, like shared and private address spaces, initializes resources in the simulator, and sets up a centralized task queue. Thus all simulation tasks are scheduled on this task queue as well as executed in an order of simulated time. Augmint also schedules a task that switches context to the main application thread, as the first task in the task queue. Once this task is extracted and executed, control is passed to the main application thread. When the main application thread executes the application code until an event is generated, it switches context back to the Augmint thread that handles the generated event. Therefore, at a specific simulated cycle, a task is created and inserted into the task queue. When this task becomes the earliest one on the task queue, Augmint extracts it and executes a function provided by the architecture simulator to process this task, which also can generate and schedule more tasks on the task queue, associated with the task. Once a task completes, to allow the corresponding application thread to advance, it sends a signal to Augmint to switch context to the corresponding application thread. Generally speaking, to handle an event from the application, Augmint creates a new thread, schedule a task on the task queue, and call functions provided by the architecture simulator to process the associated tasks.

From Figure 2, because of the relationship between the application, Augmint and the architecture simulator, we often call Augmint as the frontend and the architecture simulator as the backend. Augmint runs applications written in C. Memory allocation functions and parallel constructs such as barriers, locks, thread creation, and semaphores are expressed in m4 macros in a fashion similar to applications in the SPLASH and SPLASH-2 suites [6]. We can use C and predefined macros to write any parallel applications. Also we can modify the backend, i.e. the architecture simulator, to implement some specific architecture features such as cache coherence or bus behavior and add some statistics data collection mechanism. Finally, as illustrated in Figure 3, we compile and link them together to get one executable.

Although Augmint is an execution-driven simulation tool, it also supports trace-driven simulation for uniprocessors and multiprocessors, with minor effect on the part of simulator developers. The simulator developer can implement a simple application to read in instructions or memory reference from a trace file, decode the instructions, and generate events for Augmint. To allow complete flexibility for supporting any trace file format, Augmint shifts the trace capability to an application that reads and decodes traces. It is useful for users to run Augmint in a trace-driven mode, since they would like to use their simulators with both applications and traces collected from other systems.

2.2 How to Use Augmint

Building an Augmint based executable involves a number of steps, as shown in Figure 3. In the initial phase, the application source code is handled by m4 macro processor to translate parallel constructs as m4 macros into events and context switches defined in Augmint. The resulting source code is compiled into x86 assembly code by GNU C compiler. At the end of each basic block, the assembly code is augmented with instrumentation code, which updates the simulated clock. The augmenter also processes each memory reference with instructions to calculate the data address, value and size, to create an event for Augmint, and to switch context to the Augmint thread. Then, to generate a single executable, the augmented code is assembled into object code and linked with the object code of the simulator and Augmint.

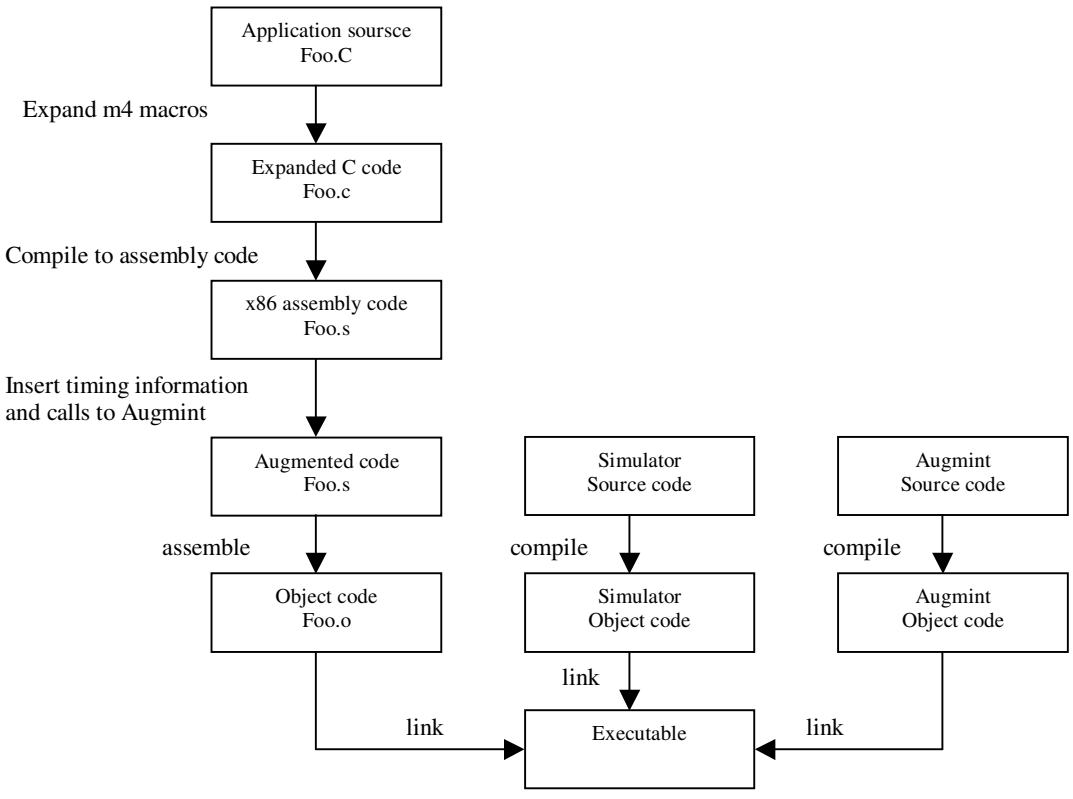

Figure 3: The Procedure for Generating a Simulation Executable

3. THE MUITIPROCESSOR IP ROUTER APPLICATION

In this project, we study multiprocessor router architectures with cache coherence protocols. According to the interconnection network technologies, multiprocessor routers can be classified into two categories, bus-based and switch-based. In shared-bus router architecture, packets are transmitted only once over the shared bus. This reduces the number of bus transactions and speeds up packet forwarding by using a local routing table cache. But the limitation is the scalability due to the nature of bus. To overcome the limited scalability of bus-based routers, a switch fabric interconnection is used to replace the shared bus in alternative designs of multiprocessor routers. This provides enough bandwidth for transmitting packets between interface cards and processors and allows the number of processors more scalable. However, the new bottleneck becomes the speed of packet processing. Therefore, in our project, we simulate two multiprocessor architectures, SMP and CC-MUMA, to incorporate these two kinds of routers respectively. Figure 4 and Figure 5 illustrate the general architectures of these two routers.

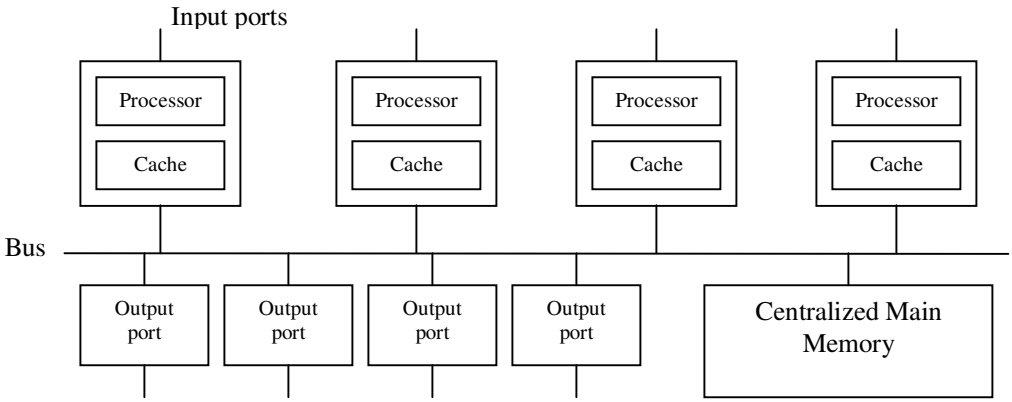

Figure 4: Architecture of the SMP-Based Router

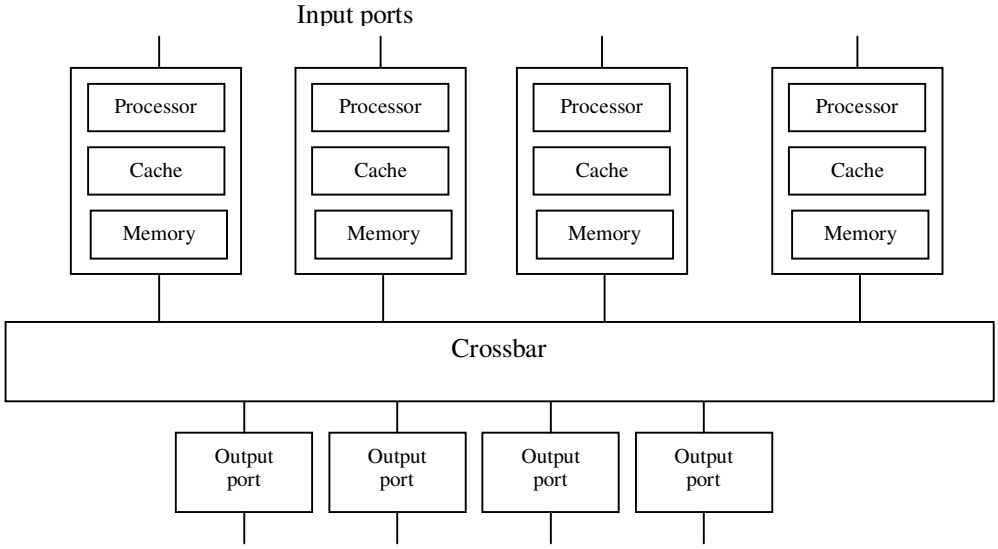

Figure 5: Architecture of the CC-NUMA Based Router

Our approach is to write a multiprocessor IP router application with C language and m4 macros, combine it with SMP and CC-MUNA architecture backends of Augmint simulator respectively, and then simulate them with NLANR traces and get the performance evaluation results.

3.1. General Structure

In our implementation, the simulated router has equal number of input ports and output ports. Each input port is associated with a processor and all the processors share the same memory space, which is simulated large enough to hold the whole routing table and buffered packets. In SMP bus-based architecture, the physical memory is centralized; in CC-NUMA architecture, the memory is distributed among processors and each processor owns a portion of the memory and a crossbar interconnection is simulated. In both architectures, each processor has a local fixed-size cache, whose size and cache line can be changed as simulation parameter.

The router application reads real internet traces [3] as its input and the packets are distributed to different input ports/processors in round-robin. Using the same approach as in [1], we establish a routing table from the extracted packet traffic itself and use red-black tree, which is illustrated below, as the routing table structure. For these two architectures, we use the same algorithm and data structures to establish the routing table and search the routing table. Therefore, in the SMP bus-based router, the routing table data structure resides in the main memory; in the CC-NUMA router, the routing table is distributed evenly to all the processors since the round-robin memory allocation algorithm is used. In addition, we simulate forwarding packets from input ports to output ports by memory copy operations, which can introduce similar delays as real ports, and we also assume the packet size as a constant of 32 bytes. To focus on the underlying architecture simulations, we do little for optimizing the parallel routing algorithm and routing table division among processors.

In our design, the router application has two running stages. Stage 1 is the routing table establishment stage and only one processor is involved in this stage, which reads the trace file, assigns an outport number randomly to each incoming packet and constructs a routing table in shared memory space accordingly. Stage 2 is the routing table lookup stage and all the processors participate in this stage. In parallel, each processor reads a portion of the trace file, searches the shared routing table with the IP addresses from the trace file and forwards the packets to corresponding output ports. Obviously, stage 1 is not suitable for performance evaluation because only one processor is involved during the routing table establishment procedure. Therefore, in our performance analysis, only the execution of stage 2 is counted.

3.2. Routing Table Structure

In the simulation, we use 400,000 HTTP packets from National Lab for Applied Networking Research (NLANR) and store them into a packet traffic trace file. During execution stage 1, from the trace file, we create a routing table in the shared memory space. A routing table entry, which has two components, IP destination address and outports, is created for each different IP address from the trace file. And an outport number is assigned to each entry in random. At this step, we use a red-black tree, which is a balanced tree, as the underlying routing table structure and lookup algorithm. A red-black tree is a binary search tree with one extra bit of storage per node: its color, which can be either RED or BLACK. By constraining the way nodes can be colored on any path from the root to a leaf, red-black trees ensure that every simple path from a node to a descendant leaf contains the same number of black nodes, so that the tree is balanced. After the balance tree is created, the processor reads the IP destination addresses, searches the balanced tree and forwards the packets to corresponding outports. The structure of a red-black tree incorporated in the routing table is shown in Figure 6.

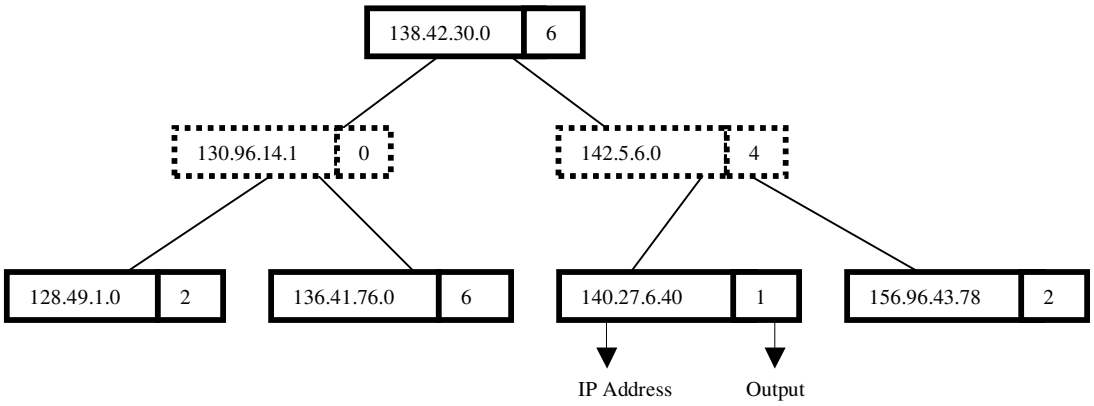

Figure 6: A Red-Black Tree
A Box with Solid Border Represents Black Node;
A Box with Dashed Border Represents Red Node

4. CUSTOMIZED BACKENDS OF AUGMINT

In our design, the SMP router architecture uses uniform/centralized shared memory and broadcast/bus-based snooping cache coherence protocol; the CC-NUMA router architecture uses non-uniform/distributed shared memory and non-broadcast directory-based cache coherence protocol. We simulate these two architectures by writing our own backends of Augmint, which means that we can use the same application described above to run with two different architectures and compare them in several aspects of performance.

4.1 Customized backends of Augmint

To model the SMP architecture (figure 4), we need a cache module (cache.c) and a bus module (bus.c) in the backend. The cache module includes a simulated write-back 2-way associative cache with varying size for each processor and the implementation of the cache coherence protocol. In addition, a simple LRU cache replacement policy applies. A read miss in the cache for a processor causes a read bus request to be issued. A write miss to an invalid block for a processor causes an exclusive read bus request to be issued, and a write miss to a valid shared block causes an invalidation bus request to be issued. The bus module arbitrates the order of these requests and introduces a constant delay (50 cycles) for each of these bus transactions. Additionally, the delay for memory write-back is skipped because we can use a write-back buffer to hide the delay. In SMP, we simulate a snooping cache-coherence protocol with write invalidate strategy, which is illustrated in figure 7.

For the CC-NUMA architecture (figure 5), we also need two modules in the backend, a cache module and an interconnection network (IN) module. Different from the bus-based architecture, the cache module in the CC-NUMA architecture simulates a directory structure to keep the sharing information of each memory block. We use a hash table to simulate a full-mapping directory structure. We use a 32-bit vector as the processor pointers, so up to 32 processors are supported. However, we believe that it is easy to support more processors by increasing the length of the pointer vector or replacing the full-mapping strategy by limited mapping. Similar to the SMP architecture, in each processor node, a 2-way associative cache with a simple LRU replacement policy is supported.

In IN module, we simulate a crossbar interconnection, which connects all the processor nodes and whenever both a source node and a destination node are free, the communication between them can be initialized. The IN module handles all the memory access or cache coherence requests from the processor nodes and introduces different delays according the referenced memory address. So each local memory access has a 30-cycle delay and each remote memory access or cache coherence transaction has 80-cycle delay. In addition, we implement a directory-based non-broadcast cache coherence protocol in this architecture, which is similar to the protocol illustrated in figure 7 except following several aspects:

(1)For each memory access, the access type is checked by the IN module. If it is a local memory access, the request is handled by local memory rather than through the crossbar.

(2)The pending read and write states should wait for crossbar to become free between the source and destination nodes.

(3)For remote memory access transactions, if the source and destination nodes are disjoined, the requests can be handle concurrently by the crossbar.

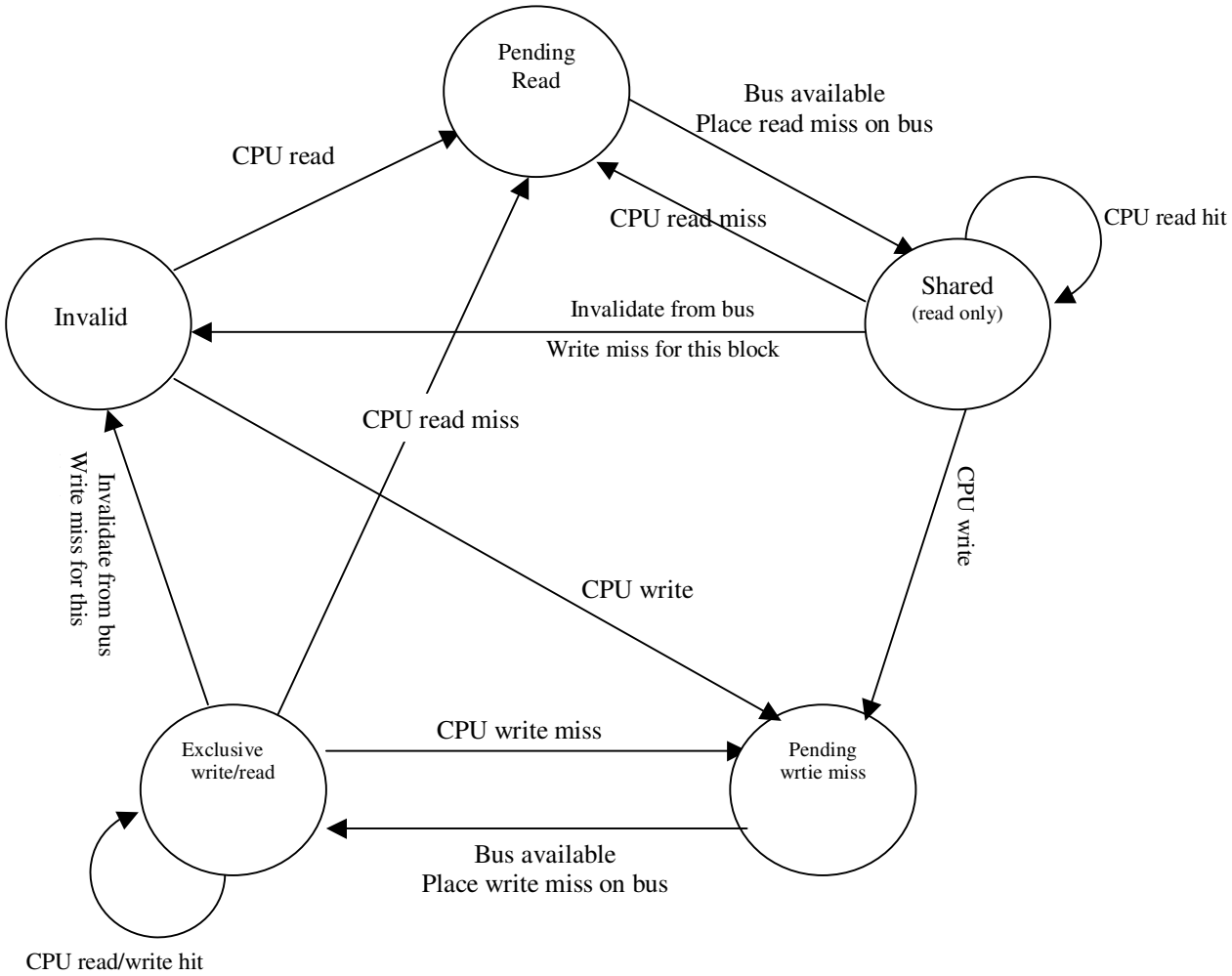

Figure 7: Bus-Based Cache-Coherence Protocol with Write Invalidate

4.2 Building Simulation Executables

After the backends of Augmint are customized, they should be incorporated into the Augmint simulation tool. As illustrated in section 2, first of all, we recompile the modified backend source files such as cache.c and bus.s, which includes the cache and interconnection modules. The object files are created and linked into Augmint backend libraries. Then we need to compile the router application source file, router.C, which is written by C and m4 macros, and link its object file with the Augmint simulation tool and backend libraries. Finally, we get an executable files which binds the router application, customized cache-memory backends and Augmint simulation mechanism together. Therefore, in the next section we present the experimental simulation results from running these executable files.

5. EXPERIMENTAL SIMULATION RESULTS

In the experimental simulation, we use a NLANR trace, which contains about 400,000 packets from real internet traffic, to feed the router applications. We evaluate the execution time and cache miss rate for these two architectures, SMP and CC-NUMA, with different numbers of processors (2 – 32), different sizes of caches (8KB – 128KB) and different cache block sizes (8 bytes – 128 bytes). Therefore, we can study on how the number of processors, cache size and cache block size affect the cache behavior and overall performance of these two different router architectures. In addition, we can compare the two router architectures and analyze their pros and cons.

5.1 Simulation of SMP-based Router Application

Table 1 lists the specification of the simulated SMP-based router application. First of all, we study the impact of number of processors on the total execution time and cache miss rate in the SMP architecture. Therefore, we fix the cache size as 64KB and cache block size as 32 bytes for each processor. Figure 8 shows how the number of processors affects the execution time of our SMP router application, and from it we can see that the execution time decreases first, with the increase of the number of processors, until the number of processors is beyond 16. When the number of processors reaches 32, however, the execution time increases because of the bottleneck effect of limited bandwidth of bus interconnection. Figure 9 demonstrates the effect of varying the number of processors on the cache miss rate. As expected, the cache miss rate increases steadily as the number of processors increases, and the increased cache misses are from coherence misses due to increased cache invalidates and exclusives memory accesses. Therefore, we can conclude that the bus interconnection limits the scalability of multiprocessor architectures.

Number of Processors	2 - 32
Memory Structure	Centralized Shared Memory
Memory Transaction Latency	50 clock cycles
Cache Structure	2-way associative with LRU replacement policy
Cache Size(KB)	8 -128
Cache Block Size(bytes)	8 - 128
Cache coherence protocol	Simulated snooping, write invalidate and write back
Interconnection	Bus

Table 1: Specification of the SMP-Based Router Application

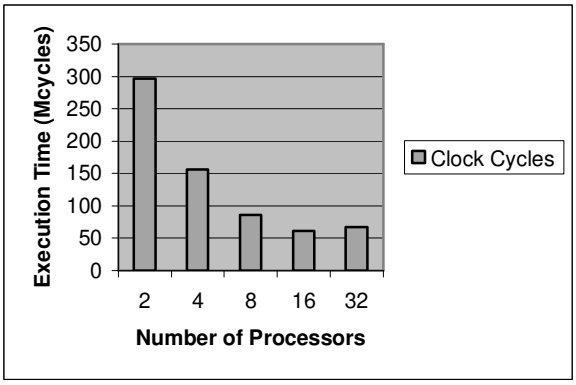

Figure 8: Execution Time versus Number of Processors (SMP)

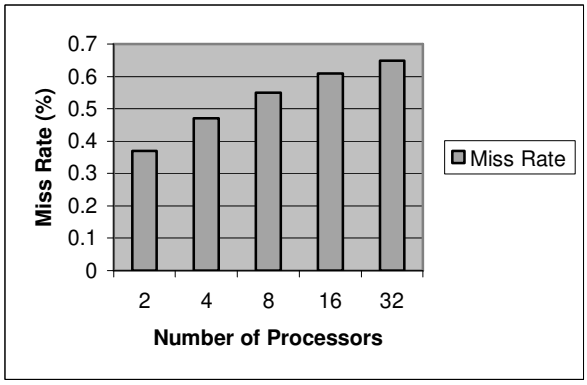

Figure 9: Miss Rate versus Number of Processors (SMP)

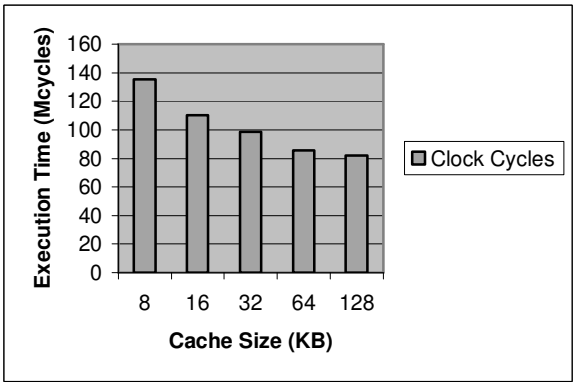

Figure 10: Execution Time versus Cache Size (SMP)

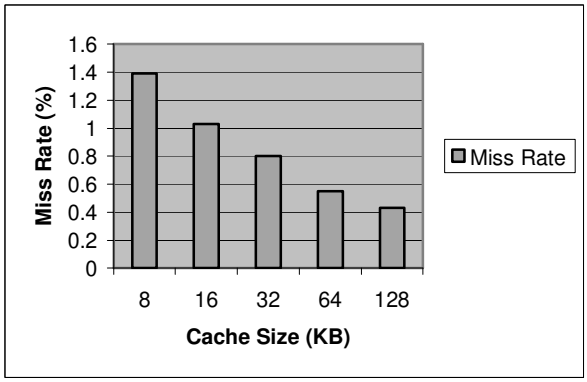

Figure 11: Miss Rate versus Cache Size (SMP)

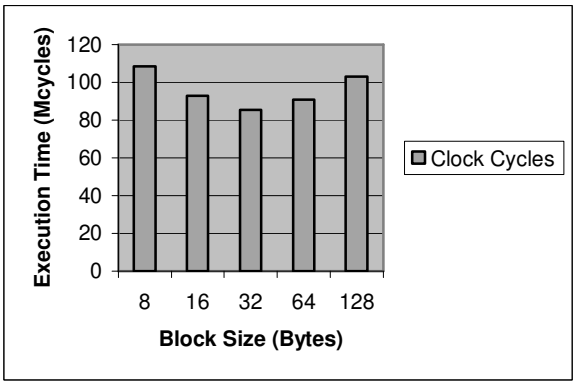

Figure 12: Execution Time versus Block Size (SMP)

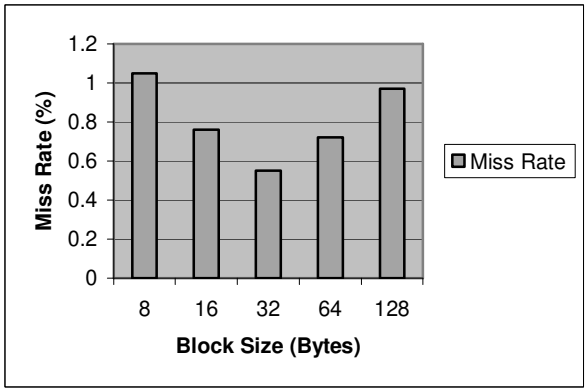

Figure 13: Miss Rate versus Block Size (SMP)

Then we talk about how the cache size affects the simulated running time and cache misses. And we fix the number of processors as 8 and cache block size as 32 bytes again. For each processor node, we vary the cache size from 8KB to 128 KB. As figure 10 shows, the execution time of our router application keeps decreasing with the increase of cache size. However, as the cache size increase to 64KB, the decreasing rate of execution time becomes low, which is because the CPU time, rather than the memory access time, begins to dominate the execution time. Figure 11 demonstrates that the cache miss rate decreases steadily as the cache size increases, which means that the capacity misses still dominate the total cache misses. Due to the characteristics of our router application, we know that the most cache misses are read miss (about than 70%), so increasing cache size effects the total miss rate significantly.

To study the cache behavior in multiprocessor router applications, we also give the experimental results about the effect of varying the cache block size on execution time and cache miss rates. We fix the number of processors as 8 and cache size as 64 KB for each processor. Figure 12 describes how the cache block size affects the execution time. From figure 12, we can see that the execution time increases first and then decreases with the increase of cache block size. From this result, we can explain that the execution time increases first because increasing block size makes use the space locality better, reduces the number of memory accesses and increases the cache hit rate. However, after the cache block size increases to a threshold, the possibility of false sharing and cache pollution arises, and running time increases rather than decreasing. Figure 13 shows how the cache miss rate is affected by the cache block size. The similarity of varying trends in figure 12 and 13 shows that the cache behavior greatly affects the execution time.

5.2 Simulation of CC-NUMA Router Application

Table 2 lists the specification of the simulated SMP-based router application. In CC-NUMA, we analyze a similar set of data as in SMP, which enable us to compare these two systems in the next sub-section. To evaluate the impact of varying number of processors on the execution time and cache miss rate, we use a 64KB 2-way associative cache with block size of 32 bytes and vary the number of processors from 2 to 32. As figure 14 shows, the execution time decreases steadily with the increase of number of processors. This trend is different from that in the SMP system and can be seen as an evidence of scalability of CC-NUMA. From figure 15, we see that the cache miss rate increases as the number of processors increases, which is due to the increased sharing degree among more processors.

Number of Processors	2 - 32
Memory Structure	Distributed Shared Memory
Memory Transaction Latency	Local 30 clock cycles; remote 80 clock cycles
Cache Structure	2-way associative with LRU replacement policy
Cache Size(KB)	8 -128
Cache Block Size(bytes)	8 – 128
Cache coherence protocol	Directory-based, write invalidate and write back
Interconnection	Crossbar

Table 2: Specification of the CC-NUMA Router Application

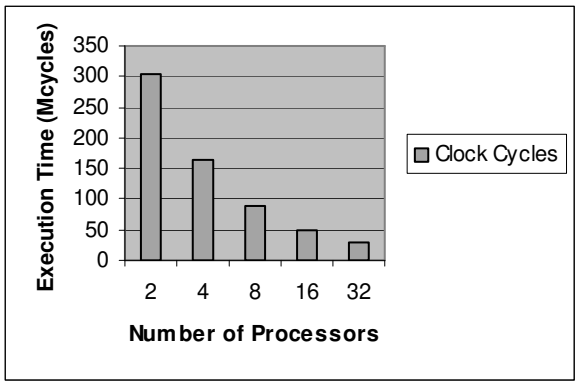

Figure 14: Execution Time versus Number of Processors (CC-NUMA)

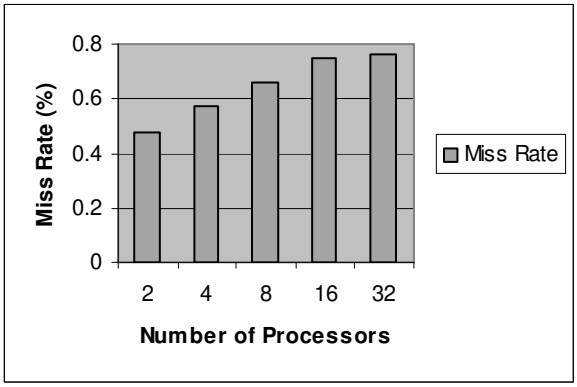

Figure 15: Miss Rate versus Number of Processors (CC-NUMA)

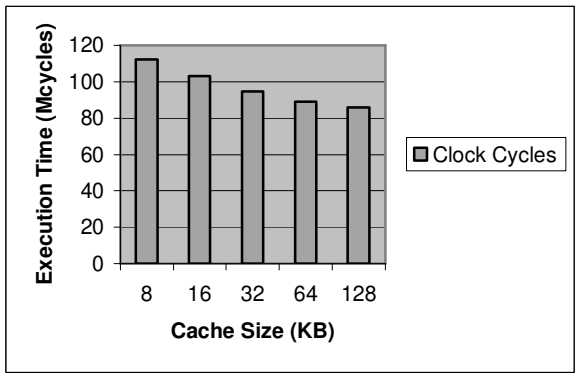

Figure 16: Execution Time versus Number of Processors (CC-NUMA)

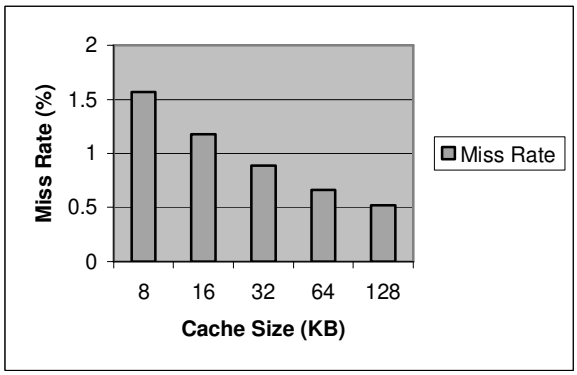

Figure 17: Miss Rate versus Cache Sizes (CC-NUMA)

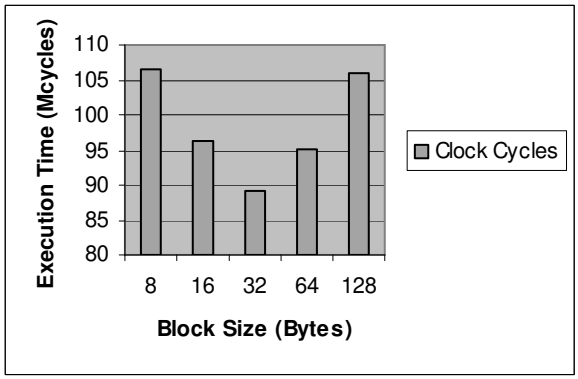

Figure 18: Execution Time versus Block Sizes (CC-NUMA)

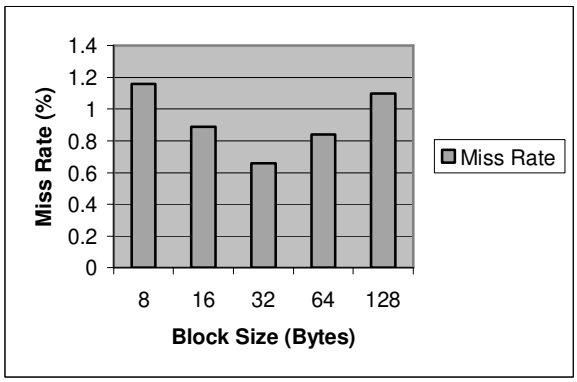

Figure 19: Miss Rate versus Block Sizes (CC-NUMA)

The experimental results concerned with the cache behavior are similar to that in SMP system. As figure 16 and 17 shows, both the execution time and cache miss rate increase with the increase of cache size from 8KB to 128 KB. Again, this gives us an obvious impression that increasing the cache size is an effective way to increase the performance of multiprocessor systems. From figure 18 and 19, we see that both the execution time and cache miss rate decrease first and then increase as the cache block size increases from 8 bytes to 128 bytes. The reason of these varying trends is similar to that in SMP as we explain before.

5.3 SMP versus CC-NUMA

In our experiment, we evaluate the systems with up to 32 processors. So the difference between SMP and CC-NUMA is not very obvious. In addition, we also different arbitrary assumptions about memory access delays in two systems. In the SMP system, we assume that the latency of a bus transaction is uniformly 50 clock cycles; and in the CC-MUNA system, we assume that a local memory access needs 30 clock cycles and a remote memory access or cache invalidation needs 80 clock cycles. But we still can see something from comparing the performance of two systems. In SMP, the execution time first decreases and then increases with the increase of the number of processors, while in CC-NUMA, the execution time decreases steadily. From this fact, we clearly see the limitation of bus-based multiprocessor systems and the obvious scalability of CC-NUMA systems. In addition, we can see that if the number of processors is small, the SMP system performs better than the CC-NUMA, which is due the different features of bus and crossbar interconnections. The former is more efficient when traffic is not so heavy, but has limited bandwidth, while the latter introduces more overhead for each communication through the network, but provides much higher bandwidth and is obviously more scalable. In addition, the cache behaviors in SMP and CC-NUMA are similar. This is because of the characteristics of our router application, where memory reads dominate the memory accesses and the impact of write invalidation is not obvious.

6. CONCLUSIONS

In this project report, we describe the background of our project, introduce a multiprocessor simulator Augmint and analyze the architectures of two kinds of routers, SMP and CC-NUMA. We write a simplified multiprocessor router application and run it with these two multiprocessor architectures by modifying the cache-memory backends of Augmint. Finally we give a detailed performance evaluation report based on our simulation of the router application. It is shown that using multiprocessor architecture can enhance the performance of IP routers. As mentioned before, we do little in parallel routing algorithm design, do not consider the efficiency problem, and use a very straightforward algorithm in our router application. In addition, we skip the write-back latency in the simulation. Therefore, the future work is to design a more sophisticated parallel routing algorithm, refine the simulated cache coherence protocol with write-back latency and compare it with our current experimental results.

REFERENCE

[1] L. Bhuyan and H. Wang, "Execution-Driven Simulation of IP Router Architectures," *IEEE International Symposium on Network Computing and Applications (NCA'01), Boston, October 2001*

[2] S. Keshav and R. Sharma, "Issues and Trends in Router Design," *IEEE Communications Magazine*, vol. 36, no. 5, pp. 144-151, May 1998

[3] National Lab of Applied Network Research, Sanitized access log. ftp://ircache.nlanr.net/Traces/, Jan 2002.

[4] S. Lin and N. McKeown, "A Simulation Study of IP Switching," *Proceedings of ACM Sigcomm*, September 1997

[5] C. Metz, "IP Routers: New Tool for Gigabit Networking," *IEEE Internet Computing*, November/December 1998

[6] A. Nguten et al, "The Augmint Multiprocessor Simulation Tookit for Intel x86 Architectures," *Proceedings of 1996 International Conference on Computer Design, October 1996*

[7] T. Wolf and J. Turner, "Design Issues for High Performance Active Routers," *IEEE Journal of Selected Areas in Communications*, vol. 19, no. 3, pp. 404-409, March 2001

[8] MINT Tutorial and User Manual, J. E. Veenstra, R. J. Fowler, The University of Rochester, Technical Report 452, June 1993.