A New Bus Allocation Algorithm for Interconnection Networks of Multiprocessor Systems

Kuochen Wang and Tony Wu
Department of Computer and Information Science
National Chiao Tung University
Email: kwang@cis.nctu.edu.tw

Abstract

In shared multiprocessor systems, the interconnection network is usually the bottleneck of system performance. Memory references usually have two kinds of locality: temporal locality and spatial locality. If a processor references a memory module, it tends to reference the same memory module again. If we do not release a bus that connect the processor and the memory module immediately, we can use the same bus directly without reconfiguration when the processor references the same memory module again. Thus, we propose a new bus allocation algorithm for interconnection networks of multiprocessor systems. A pipelined one-sided crossbar switch, which is essentially a multiple bus, is used to illustrate our design approach. Experimental results show that the new algorithm uses buses more efficiently and reduces the number of reconfigurations. The performance (throughput) using the new bus allocation algorithm is up to 3 times higher than that using the original algorithm. The contribution of this work is designing a high throughput interconnection network to match high performance multiprocessors and to eliminate the performance bottleneck.

Keywords: arbiter, bus allocation, interconnection network, multiprocessor system.

1 Introduction

In order to achieve high performance, modern multiprocessor systems emphasize on increasing their parallel computation power. A multiprocessor system is shown in Figure 1. In a shared memory multiprocessor system, there are several processors and a shared memory. The shared memory is divided into several independent memory modules. The processors are connected to these memory modules by an interconnection network. An interconnection network is a key component in high performance multiprocessor systems [1] [2] [3]. There are several structures for the interconnection network such as single bus, multiple bus, crossbar switch, multi-port memory, and multistage interconnection network [4] [5], which have been proposed for shared memory multiprocessor systems. The crossbar switch provides the highest throughput and interconnection capability among these interconnection networks. It allows all possible simultaneous, non-blocking, one-to-one connections between processors and memory modules. When a bus is allocated for a transaction, it takes at least one clock cycle to reconfigure the interconnections network. Therefore, if we reduce the number of reconfigurations, we can improve the system performance.

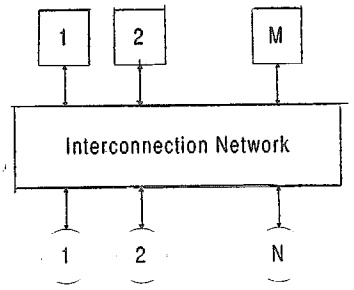

Figure 1: A multiprocessor system.

As we know, memory references have two kinds of locality [4]:

- *Temporal locality* : If a memory address is referenced, it will tend to be referenced again soon.
- *Spatial locality* : If a memory address is referenced, memory addresses that are close by will tend to be referenced soon.

Therefore, if a processor references a memory module, it tends to reference the same memory module again soon. Based on this observation, we propose a new bus allocation scheme for the one-sided crossbar switch. This scheme allows the bus ownership to be retained by the current bus owner even if it currently does not have a pending transaction. Since the one-sided crossbar switch

is a nonblocking switch, it is especially suitable for this scheme. In contrast, in an Intel Pentium Pro processor, the bus ownership can also be retained. However, it increases arbitration latency by two clock cycles if other processors generate new transactions [13].

2 Existing Approach

A pipelined one-sided crossbar switch was proposed in [12]. This pipelined switch, as shown in Figure 2, includes four parts:

- *One-sided crossbar switch*
- *Arbiter*
- *Processor interface*
- *Memory interface*

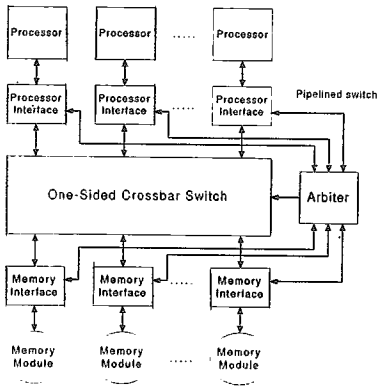

Figure 2: The multiprocessor system architecture.

Note that we add memory interfaces to the original design in [12] to simplify the design of processor interfaces in [12]. Each processor is connected to a processor interface and each memory module is connected to a memory interface. The processor interfaces are connected to these memory interfaces by the one-sided crossbar switch. The arbiter controls the one-sided crossbar switch and arbitrates each transaction. The processor interface handles the data transfer between the processor interfaces and the memory interfaces. The switch is a non-blocking switch, and each bus-line includes an address bus and a data bus. The widths of the address and data buses are both 32 bits. The arbiter arbitrates the connections between processor interfaces and memory interfaces. It controls whether crosspoints are activated or not for connections. The arbiter determines which request will be allocated a bus to establish a connection. The processor interface receives an address (READ transaction) or an address and data (WRITE transaction) from a processor, and sends them to a memory interface through the one-sided crossbar switch. It also receives data (READ transaction) from a memory interface, and sends them back to the corresponding processor. A processor interface controls the sending and receiving of addresses and data in a pipelined way. It keeps track of a transaction in the arbitration and request phases. The memory interface receives an address (READ transaction) or an address and data (WRITE transaction) from a processor interface through the one-sided crossbar switch, and send them to the memory module. It also receives data (READ transaction) from the memory module, and sends them back to the corresponding processor interface. Pipelined (split) transactions of the one-sided crossbar switch was designed [12]. The state diagram of the proposed pipelined protocol is shown in Figure 3. A pipelined interface controls the sending and receiving of addresses and data in a pipelined way. It tracks the transactions in the snoop, response, and data phases.

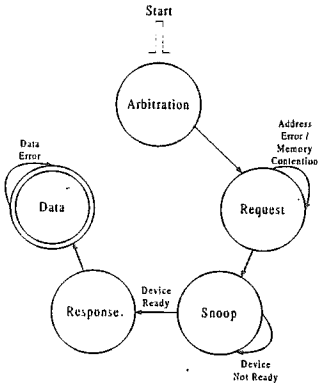

Figure 3: The state diagram of the proposed pipelined protocol.

In [12], a transaction is divided into five phases:

- *Arbitration*: A transaction in a processor is selected to become the owner of a bus on a round-robin basis. After that, the processor will send transaction information (R/W, address, data for write) to the pipelined interface.
- *Request*: The processor interface sends the transaction information to the memory interface through the one-sided crossbar switch. In this phase, the bus that is selected for the transaction is busy for data transfer.
- *Snoop*: The transaction in this phase waits for the memory module ready.

- *Response*: The target device (memory module) sends a response message (R/W, acknowledgment) back to the pipelined interface.
- *Data*: If there is no data error, the read data are transferred to the processor via the processor interface.

3 Design Approach

We use *connection tables* to record the states of each processor, each memory module, and each bus, respectively. For each processor or each memory module, the entries in the connection table, as shown in Figure 4, consists of two fields: C and *bus*. The length of C is one bit and it is used to indicate if a processor/memory module is connected to a bus or not. If C is 0, the processor/memory module is not connected to a bus. Otherwise, the processor/memory module is connected to a bus. If the processor/memory module is connected to a bus, the *bus* field is used to indicate which bus is connected to the processor/memory module, and its length is $logB$ bits, where B is the number of buses.

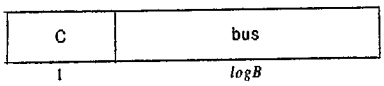

Figure 4: The entries in the connection table for a processor/memory module.

For each bus, the entries in the connection table, as shown in Figure 5, consists of three fields: *state*, *processor*, and *memory module*. The length of *state* is two bits, and it is used to indicate the state of a bus:

- 00: The bus is not connected to any processor or any memory module.
- 01: The bus is assigned to a processor and a memory module, but not connected.
- 10: The address and data information of a transaction are in the bus.
- 11: The bus is connected to a processor and a memory module, but there is no information in it.

The *processor* field is used to indicate which processor to connect to the bus, and its length is $logP$ bits, where P is the number of processors. The *memory module* field is used to indicate which memory module to connect to the bus, and its

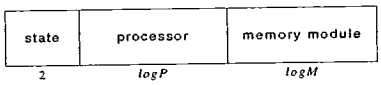

Figure 5: The entries in the connection table for a bus.

length is $logM$ bits, where M is the number of memory modules. The new bus allocation algorithm is described as follows. First, the arbiter selects a transaction, processor p referencing a memory module m, to become the bus owner on a round-robin basis. There are four situations:

- *Processor p is connected to bus i, and memory module m is connected to bus j.* If i is equal to j, the transaction can skip the arbitration phase and enter the request phase immediately. If i is not equal to j, we select bus i for the transaction, and the transaction will enter the arbitration phase.
- *Processor p is connected to bus i, and memory module m is not connected to any bus.* We select bus i for the transaction, and the transaction will enter the arbitration phase.
- *Processor p is not connected to any bus, and memory module m is connected to bus j.* We select bus j for the transaction, and the transaction will enter the arbitration phase.
- *Processor p and memory module m are not connected to any bus.* Because the pipelined one-sided crossbar switch is a non-blocking switch, there is at least one bus available. We select one available bus for the transaction, and the transaction will enter the arbitration phase.

Note that in [12], a bus allocated for a transaction is waiting for being released when the transaction enters the snoop phase. We do not release the bus in the new bus allocation algorithm. If the processor issues a new transaction which references the same memory module later, the transaction can skip the arbitration phase and enter the request phase immediately.

4 Evaluation and Discussion

We implement the new bus allocation algorithm in the pipelined one-sided crossbar switch [12] and compare the throughput of the pipelined one-sided crossbar switch with the new bus allocation algorithm with that with the original bus allocation algorithm. There are five parameters for simulation:

- P: number of processors

- M: number of memory modules
- B: number of buses
- P_r: the probability that a processor issues a transaction in each clock cycle
- P_s: the probability that a processor issues a new transaction and references to the same memory module as the last transaction

And we define throughput as the average number of transactions completed per clock cycle. Figure 6 shows the relationship between throughput and P for $M = B = P$ and $P_r = 1.0$. The results show that the throughput is in proportion to P when $M = B = P$ and $P_r = 1.0$.

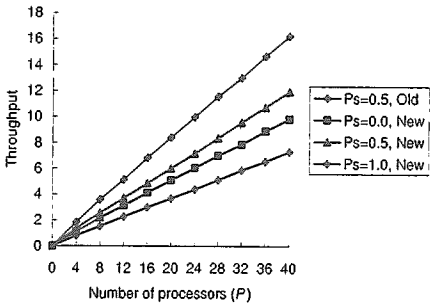

Figure 6: Throughput of each one-sided crossbar switch for P from 4 to 40.

Figure 7 shows relationship between throughput and M for $B = P = 4$, and $P_r = 1.0$. The results show the more memory modules, the higher throughput. The reason is that if we have fewer memory modules, there are more memory conflicts and it results in lower throughput. More memory modules means higher hardware cost and lower P_s. Figure 7 also shows that the throughputs are almost the same in the original design no matter P_s is equal to 0.0 or 0.5. However, in our new design, throughput increases up to 1.3 times when P_s increases from 0.0 to 0.5. Figure 8 shows the relationship between throughput and P_s for $M = B = P$ and $P_r = 1.0$. The results show the higher P_s, the lower throughput in the original design. This is because, if processor p_1 and processor p_2 reference the same memory module at the same time, the higher P_s, the higher probability that processor p_1 and processor p_2 reference the same memory module again later. It results in the more memory conflicts, the lower throughput. In the new design, although there are more memory conflicts when P_s is higher, more transactions can skip the arbitration phase and enter the request phase immediately. Therefore, the higher P_s is, the higher throughput is in our new design.

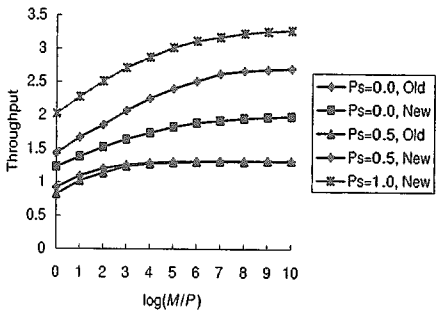

Figure 7: Throughput of each one-sided crossbar switch for M from P to $1024P$.

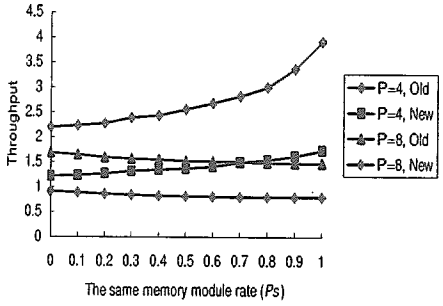

Figure 8: Throughput of each one-sided crossbar switch for Ps from 0.0 to 1.0.

Figure 9 shows the relationship between throughput and P_r for $M = B = P$. The results show that:

- The throughput difference between the two designs is getting smaller as P_r is smaller.
- The throughput will not increase when P_r is greater than 0.33 in the original design.
- The throughput will not increase when P_r is greater than $1 \; / \; [\; 2 \times (\; 1 - P_s \;) + 1 \times P_s \;]$ in the new design.

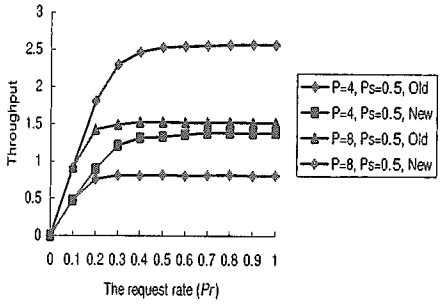

Figure 9: Throughput of each one-sided crossbar switch for Pr from 0.0 to 1.0.

The corresponding reasons are as follows:

- When P_r is small, all transactions can be completed in the two designs, so the throughputs are almost the same.
- In the original design, if a transaction is selected to become the bus owner at time t, it enters the arbitration phase. It takes one clock cycle to reconfigure the one-sided crossbar switch, so the transaction enters the request phase at time t+1. After sending address and data information to the memory interface through the one-sided crossbar switch, the transaction enters the snoop phase at time t+2. Since the bus allocated for the transaction is waiting for release, it can not be allocated for other transactions until time t+3. Therefore, for two simultaneous transactions, it needs to wait three clock cycles to allocate a bus to the second transaction when the first transaction is allocated the same bus. So, if P_r is greater than $1/3 = 0.33$, the bandwidth of the one-sided crossbar switch is not enough to complete all transactions. Therefore, the throughput will not increase when P_r is greater than 0.33.
- In the new design, if a transaction is selected to become the bus owner at time t, it enters the arbitration phase. The transaction enters the request phase at time t+1. At the same time, a new transaction is also selected to become the owner of the same bus. The first transaction enters the snoop phase at time t+2. If the second transaction is for the same processor and the same memory module as the first transaction, it can enter the request phase immediately at time t+2. So the distance of the two transactions is one clock cycle. If the second transaction is not for the same processor and the same memory module, it enters the request phase at time t+3. The distance of the two transactions is two clock cycles. Therefore, the average distance of two transactions is $2 \times (1 - P_s) + 1 \times P_s$ clock cycles. So the throughput will not increase when P_r is greater than $1 / [2 \times (1 - P_s) + 1 \times P_s]$.

5 Conclusions

We have presented an efficient bus allocation algorithm for interconnection networks of multiprocessor systems. The main difference between our new bus allocation algorithm and the original bus allocation algorithm is that we fully utilize the locality of memory references. By not releasing a bus that connects a processor and a memory module immediately, we can use the same bus directly without reconfiguration when the processor references the same memory module again. It helps us to reduce the number of reconfigurations and to improve the throughput of the interconnection network. Experimental results have shown that the throughput of the pipelined one-sided crossbar switch with the new bus allocation algorithm is up to 3 times higher than that with the original bus allocation algorithm. Therefore, our new bus allocation algorithm indeed can improve the throughput of the pipelined one-sided crossbar switch. Our new bus allocation algorithm not only can be implemented in the pipelined one-sided crossbar switch, but also can be extended to other interconnection networks of multiprocessor systems.

6 Acknowledgement

This research was supported in part by the National Science Council, ROC under Grant NSC88-2213-E-009-039.

References

[1] K. Hwang, "Advance Computer Architecture: Parallelism, Scalability, Programmability," *McGraw-Hill*, 1993.

[2] L. Hammond, B. A. Nayfeb, and K. Olukotun, "A Single-Chip Multiprocessor," *IEEE Computer*, pp. 79-85, Sep. 1997.

[3] F. Pong, M. Browne, A. Nowatzyk, and M. Dubois, "Design Verification of the S3.mp Cache-Coherent Shared-Memory System," *IEEE Trans. on Computers*, pp. 135-140, Jan. 1998.

[4] J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach, Second Ed.," *Morgan Kaufmann Publishing Company*, 1996.

[5] W.-J. Hahn, K.-W. Rim, and S.-W. Kim, "SPAX: a New Parallel Processing System for Commercial Applications," in *Proceedings 11th International Parallel Processing Symp.*, Apr. 1997, pp. 744-749.

[6] T. Lang, M. Veloro, and M. A. Fiol, "Bandwidth of Crossbar and Multi Bus Connections for Multiprocessors," *IEEE Trans. on Computers*, vol. 31, no. 12, pp. 1227-1234, Dec. 1982.

[7] K. Hwang and F. A. Briggs, "Computer Architecture and Parallel Processing," *McGraw-Hill*, 1984.

[8] A. Varma, C. J. Ceorgious, and J. Ghosh, "Rearrangeable Operation of Large Crosspoint Networks," *IEEE Trans. on Communications*, vol. 38, no. 9, pp. 1616-1624, Sep. 1990.

[9] C. J. Georgiou, "Fault-Tolerant Crosspoint Switching Network," in *Proceedings of the 14th Int. Fault-Tolerant Computing*, July 1984, pp. 240-245.

[10] A. Varma and S. Chalasani, "Fault-Tolerance Analysis of One-Sided Crossbar Switch Networks," *IEEE Trans. on Computers*, vol. 41, no. 2, pp. 143-158, Feb. 1992.

[11] K. Wang and C. K. Wu, "Design and Simulation of Fault-Tolerant Crossbar Switches for Multiprocessor Systems," *IEE Proceedings - Computers and Digital Techniques*, pp. 50-56, Jan. 1999.

[12] K. Wang and A. Y. Liu, "HDL Design and FPGA Implementation of a Pipelined One-Sided Crossbar Switch for Multiprocessor Systems," in *Proceedings of the 9th VLSI/CAD Symposium*, Aug. 1998, pp. 419-422.

[13] Intel Corp., "Pentium Pro Family Developer's Manual, Volume 1: Specifications," 1997.

[14] K. Wang and Y. H. Hsiao, "A High Performance Pipelined One-sided Crossbar Switch for Multiprocessor Systems," in *Proceedings of the 1998 International Conference on Chip Technology*, Apr. 1998, pp. 264-269.

[15] H. C. Hsiao and C. T. King, "Performance Evaluation of Cache Depot on CC-NUMA Multiprocessors," in *Proceedings of the 1998 International Conference on Parallel and Distributed Systems*, Dec. 1998, pp. 519-526.

[16] J. Carter, C. C. Kuo, R. Kuramkote, and M. Swanson, "Design Alternatives for Shared Memory Multiprocessors," in *Proceedings of the International Conference on High Performance Computing*, Dec. 1998, pp. 41-50.