Parallel Merge Sort on a Binary Tree On-Chip Network

Stephan Wong, Stamatis Vassiliadis, and Jae Young Hur
Computer Engineering Laboratory
Electrical Engineering Department Delft University of Technology
Mekelweg 4, 2628 CD, Delft, The Netherlands
Email: {J.S.S.M.Wong | S.Vassiliadis | J.Y.Hur }@ewi.tudelft.nl

Abstract— **Recently, advances in the microelectronics technology enable us to integrate increasingly more processor cores in a single chip. As the chip density continues to grow, the communication latency compared to the computation time is becoming a dominant factor in modern single chip multiprocessors systems. Consequently, networks on a chip (NoC) have been introduced as an alternative to shared busses in order to provide higher scalability, higher abstraction levels, and increased modularity. However, many challenges remain in the design of the NoC, e.g., maintain a certain quality of service (QoS) with limited bandwidth, incorporate deadlock-free routing, and implement congestion-free flow control. In order to meet these challenges, network topologies play an important role in multiprocessor systems. In this paper, we present a case study encompassing the implementation of an NoC based on the binary tree in a field-programmable gate array (FPGA) and the parallel merge sort as an application. A hardware design for a packet switched source router with a worm-hole flow control and a round-robin arbitration is described. The implemented platform provides a homogeneous, deadlock-free, congestion-free, cost-efficient communication mechanism and application-specific synchronization. In addition, the distributed shared memory organization allows for easier application programming. The experiments indicate that a single 32-bit sorting-specific processing element consumes less than 1% of the chip area. In addition, each binary tree router requires less than 3% of the area of the FPGA, while providing up to 2.9Gbps at 180 MHz.**

Keywords— **networks on a chip, field-programmable gate array, topology, parallel sorting.**

I. Introduction

As technology advances, the latency of the network interconnect is increasingly becoming the most prominent latency factor in modern on-chip multicore systems [1]. Conventional on-chip communication schemes based on shared buses do not provide sufficient scalability. Recently, networks on a chip (NoC) have appeared as a promising design paradigm providing better scalability, different abstraction levels, and modularity [2]. However, there are still open issues in the design of an NoC system, e.g., maintain congestion-free flow control and a specific quality of service (QoS) with limited bandwidth, incorporate distributed dead-lock free/starvation free/live-lock free routing, and provide fault-tolerance and load balanced routing. These issues must be resolved while minimizing latencies and maximizing the throughput. In order to meet these challenges, network topologies play an important role and greatly influence the behavior of systems. In this paper, we present a case study that entails the implementation of an on-chip network in a field-programmable gate array (FPGA). The parallel merge sorting algorithm is considered as an application since the sorting is one of most well-studied areas and is relatively simple. A merge sorting-specific 32-bit processing element and a specific simple synchronization scheme have been implemented. The on-chip network has been modeled based on the tree topology. The tree topology has been adopted since the topology offers a minimum hop count and congestion-free traffic for merge sort. The network component is generic, i.e., it is independent from the computational components. The routing schemes are tailored for the tree topology. Input queued packets are arbitrated with a round-robin policy. In addition, each network component communicates with neighboring components using a handshaking protocol. A flit-based wormhole flow control is adopted for efficient buffer utilization and pipelined message transmission. The implemented platform provides a homogeneous and deadlock-free routing mechanism. Furthermore, the memory organization entails a distributed shared memory allowing much easier application programming. Experimental results indicate that the binary tree is indeed suitable for the parallel merge sort algorithm.

The organization of this paper is as follows. In Section 2, related works are described. In Section 3, parallel merge sorting algorithm is reviewed. The implementation for the case study and experimental results are described in Sections 4 and 5, respectively. In Section 6, conclusion is made and future work is presented.

II. Related Work

Various NoC-based systems have been publicized and are chronologically listed in the following. aSOC (mesh, 2000) [3], Æthereal (mesh, 2002) [4], SOCIN (mesh, 2002) [5], SPIN (fat-tree, 2003) [6], T-SoC (fat-tree, 2003) [7], HERMES (mesh, 2004) [8] have been previously introduced among others. It can be observed that the mesh topology has been utilized for the majority of the proposed systems as the main underlying topology. In 2005, a MultiNoC system [9] based on the HERMES [8] switch has been introduced. Our system is similar to MultiNoC system [9] in that the handshaking protocol and flow control are managed in a similar way. However, the followings are still differ from the MultiNoC [9] in several ways. First, the network is based on the tree topology. Second, we focus on sorting-specific application and 32-bit processing element has been implemented. Third, the tree routing scheme is based on source routing. Fourth, the synchronization scheme is optimized for merge sort. Fifth, the system consists of completely homogeneous nodes.

III. Parallel Merge Sort

A. Basic Definitions

Some terminologies of the interconnection network are given in the following. *Latency* refers to the average packet delay, measured from header departure to arrival of last packet from the network. *Throughput* refers to the average number of packets routed through the network per clock cycle. The *diameter* of a network is the maximum distance between any two processing nodes, where *distance* refers to the number of links within the shortest path. *Area cost* is in terms of the number of communication links. The definition of the *sorting* can be formally re-stated as follows [10]:

Definition Given a sequence $S = \{x_1, x_2, ..., x_n\}$ of n items on which a linear order is defined, the purpose of *sorting* is to arrange the elements of S into a new sequence $S' = \{x'_1, x'_2, ..., x'_n\}$ such that $x'_i < x'_{i+1}$ for $i = 1, 2, ..., n-1$.

B. Merge Sort

An example of the parallel merge sort is illustrated in Fig. 1. Consider 7 nodes $N0$, $N1$,..., $N6$ are arranged in a binary tree and the address space is divided over the 7 nodes as depicted in Fig. 1 (a). Fig. 1 (b) depicts the initially arranged un-sorted elements in $N3$,$N4$,$N5$,$N6$. Elements $\{4, 3\}$, $\{5, 7\}$, $\{6, 2\}$ and

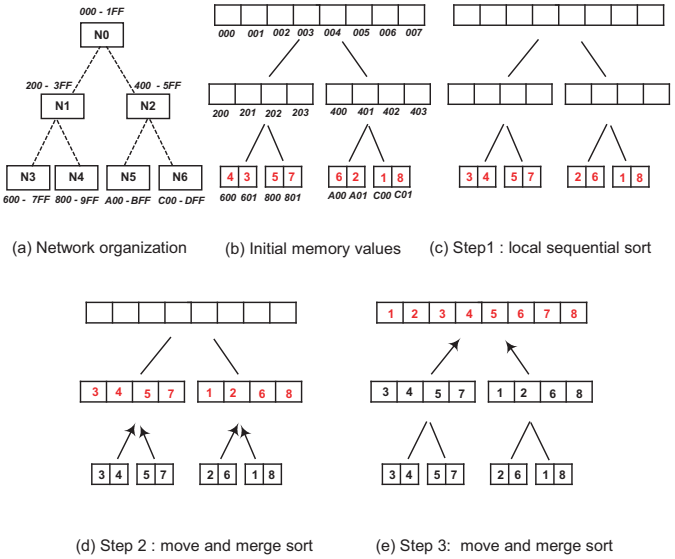

Fig. 1. Merge Sort - an example.

$\{1, 8\}$ are located at the addresses $(600\text{H}, 601\text{H})$ in $N3$, $(800\text{H}, 801\text{H})$ in $N4$, $(A00\text{H}, A01\text{H})$ in $N5$ and $(C00\text{H}, C01\text{H})$ in $N6$, respectively. In step 1 of parallel merge sort, a local sequential sort in $N3$ to $N6$ is performed as depicted in Fig. 1 (c). After that, each of the elements in $\{3, 4\}, \{5, 7\}$ is moved up to $N1$. Similarly, each of the elements in $\{2, 6\}, \{1, 8\}$ is moved up to $N2$. As soon as these remote elements are loaded, the local merge sort in each of $N1$ and $N2$ is parallelly performed and the resulting array is stored in each of the address space in $N1$ and $N2$, respectively as depicted in Fig. 1 (d). Similarly, in step 3, each element is moved up to the topmost node $N0$ from $N1$,$N2$ and merge sort is performed as depicted in Fig. 1 (e), resulting in the final sorted array. As the algorithm intuitively indicates, the tree topology is the most suitable for the implementation.

IV. Implementations

A. System Overview

The following assumptions were made in the implementation and system design. First, the interconnection network is assumed to be static, i.e., the underlying direct network is composed of pairs of processing node and router. Second, adjacent nodes are connected by a pair of unidirectional channels in opposite directions. Third, there is no traffic load other than merge sort. Consider the application program is chosen as depicted in Fig. 1. Based on the abovementioned assumption, the underlying network is organized as depicted in Fig. 2. Note that PN (processing node) refers to the combination of processing

element (PE), network interface (N/I) and memory. Node (N) refers to the combination of the processing node (PN) and router (R).

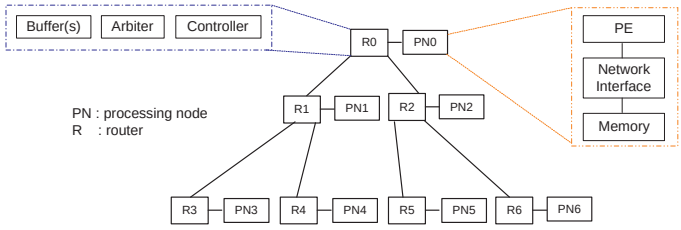

Fig. 2. Underlying Organization of System.

The network is indeed an interconnection of routers and these routers are interconnected within certain topologies. The processing nodes and the network are independent each other. Each processing node and router assumed to be homogeneous in this work. The network component is modular, considered as an IP (intellectual property) and can be replaced by other networks without having to change the processing node. The memory in each node is physically distributed but logically shared. The communication with a neighboring component is implemented using a handshaking protocol. The N/I handles the packetization (or segmentation) and de-packetization (or assembly), together with the distributed shared memory management. The synchronization is achieved using the dedicated 'en' and 'sorted' signal ports.

B. Processing Node

The PE is implemented utilizing a merge sort-specific state machine with 13 states. In this implementation, it is assumed that the number of element and the first address in each array are pre-defined in the state machine. Once the 'en' (enable) port is activated, the state machine starts remote loading to perform merge sorting. Once the merge sort is finished, the 'sorted' pin is activated. The PE performs the sorting of unsigned integers.

The major part of the N/I is a state machine to deal with the packet and memory management. For merge sort, 2 types of packets are required. First, a *request* packet is used to send a load request to a remote node. Second, a *data* packet is used to return with the data to the requesting node. Each packet is composed of flits, 6 flits for the *request* packet and 8 flits for the *data* packets. Each flit is 8 bits in size. The first flit of each packet is a header containing the target node address. When packets from the network are received, the N/I identifies the packet. If the arrived packet is a *data* packet, then the packet is forwarded to the PE (after assembly) and if the arrived packet is a *request* packet, then the packet is forwarded to memory module (after assembly). When a PE accesses the memory, N/I checks the address. If the address is a local address, the PE directly reads (or writes to) the local memory. If the address is a remote address, then the N/I generates a *request* packet to the corresponding node by utilizing an address map.

The memory component is located in each processing node. The memory module is used for following 2 purposes. First, the PE locally stores data after the merge sort is performed. Second, the memory module receives a *request* packet and directly returns the *data* packet to the remote requesting node. The Xilinx Block RAM primitive, $RAMB16\_S36$ is utilized as our memory component, providing a single port and 512 entries with each 32-bit data.

C. Tree Router

As a key component in the network, the router determines the path for packet flows to the destination. Two main operations in the router are input port selection process and output port selection process. The input port selection process is performed in arbiter and the output selection process is performed in the controller. In this work, flit-based wormhole flow control has been adopted for efficient buffer utilization. As soon as the header arrives at the router, the header is stored in the input buffer and waits for the acknowledgement from the controller to flow to the output port. This input port selecting process is performed based on the round-robin arbitration policy. Once the header is granted access to the output port, the controller determines an output port based on the routing scheme and header information. The implemented routers are dead-lock free, since there is no cyclic path.

As depicted in Fig. 3, there are 4 ports in each tree router, namely, PR (parent), LC (left child), RC (right child) and Local port. Each port contains two unidirectional ports for handshaking protocol. The FIFO buffer is located in the input port. Round-robin scheduling is done in the sequence with PR, LC, RC, Local, PR, and so on. As an example, the routing scheme at the router R1 in Fig. 2 is implemented based on the following table.

Destination	N0	N1	N2	N3	N4	N5	N6
Port name	PR	Local	PR	LC	RC	PR	PR

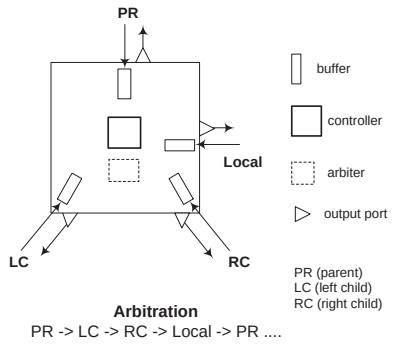

Fig. 3. Tree Router.

V. Experimental Results

The experimental results are presented in Fig. 4. For the experiments, Modelsim has been used for the behavioral (pre-synthesis) and timing (post-synthesis) VHDL simulation. The Xilinx ISE [11] tool has been used for the synthesis, mapping and placement/routing. For the prototyping, the Digilent XUP V2P board with Virtex II Pro FPGA has been used. Xilinx ChipScope Pro tool [11] was used for on-chip verification. The experimental results for the 7-node tree are shown in Fig. 4 (a). 66824 cycles are required for 512 elements to be sorted. Fig. 4 (b) shows the synthesis result for single router. 2.7% of slices are required for the single router. The clock frequency is 180.4MHz, number of port per router is 4, flit size is 8, hence, the peak bandwidth for tree can be derived as $(180.4 \text{ MHz} / 2) \cdot (4 \text{ ports}) \cdot (8 \text{ bits}) = 2.9\text{Gbps}$. Note that it takes 2 cycles for buffer to store and forward each flit. The PE is area efficient requiring less than 1% of the logic in FPGA. In addition, the N/I in the processing node and buffer in the router component are most area-consuming. Fig. 4 (d) shows the #cycle (latency) with respect to the number of elements (traffic load). As the number of elements increases, the latency grows linearly since the routing latency grows linearly to #router.

VI. Conclusion

In this work, a binary tree based on-chip network for the merge sort is implemented in FPGA hardware and analyzed. The tree is indeed, as expected, suitable in a merge sort traffic. Up to 15 nodes have been mapped to FPGA and verified. The implemented NUMA (non-uniform memory access)-model multiprocessors platform provides an area-efficient and dead-lock free communication.

Topology	#node	# total port	# Cycle	Max. Clock Freq. (MHz)	Min. Clock period (ns)	Area (%) Slice	BRAM
TREE	7	28	66824	168.3	5.943	30.1	5.3

(a) Merge Sort on Tree

Router	# port	#Cycle (header)	Area Slice (%)	Max. Clock Freq. (MHz)	Peak BW (Gbps)
TREE	4	13	2.7	180.4	2.9

(b) Tree Single Router

Component	Slice(%)
Single Buffer	0.57
Ariter	0.22
Controller	0.37

(c) Area of Components in Single Router (Buffer Size=8)

#element	#cycle
8	1054
32	4012
64	8188
128	16540
256	33244
512	66824

(d) Latency vs Load - 7 node Tree

Fig. 4. Experimental Results.

References

[1] International Technology Roadmap for Semiconductor - Interconnect, 2004 Update, pp. 1-21, 2004, http://www.itrs.net.

[2] G. D. Micheli and L. Benini, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, vol. 35, pp. 70–78, Jan. 2002.

[3] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A Scalable, Single-Chip Communications Architecture," in *IEEE Int'l Conference on Parallel Architectures and Compilation Techniques*, pp. 37–46, 2000.

[4] E. Rijpkema, K. Goossens, and A. Radŭlescu, "A Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip," in *Proceedings of Design, Automation and Test in Europe (DATE'03)*, pp. 350–355, 2003.

[5] C. Zeferino and A. Susin, "SoCIN: A Parameteric and Scalable Network-on-Chip," in *16th Symposium on Integrated Circuits and Systems Design (SBCCI'03)*, pp. 169–174, 2003.

[6] A. Andriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. Zeferino, "SPIN: a Scalable, Packet Switched, On-Chip Micro-network," in *Proceedings of Design, Automation and Test in Europe (DATE'03)*, pp. 70–73, 2003.

[7] P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Design of a Switch for Network on Chip Applications," in *Int'l Symposium on Circuits and Systems (ISCAS'03)*, pp. 217–220, 2003.

[8] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: an Infrastructure for Low Area Overhead Packet-switching Netwoks on Chip," in *Integration, the VLSI Journal*, pp. 69–93, 2004.

[9] A. Mello, L. Möller, N. Calazans, and F. Moraes, "MultiNoC: A Multiprocessing System Enabled by a Network on Chip," in *Proceedings of Design,Automation and Test in Europe (DATE'05)*, pp. 234–239, 2005.

[10] S. G. Akl, *Parallel Sorting Algorithms*. Academic Press, 1985.

[11] Xilinx, Inc., http://www.xilinx.com.