A Low-Area Multi-Link Interconnect Architecture for GALS Chip Multiprocessors

Zhiyi Yu and Bevan M. Baas

***Abstract*—A new inter-processor communication architecture for chip multiprocessors is proposed which has a low area cost, flexible routing capability, and supports globally asynchronous locally synchronous (GALS) clocking styles. To achieve a low area cost, the proposed statically-configurable asymmetric architecture assigns large buffer resources to only the nearest neighbor interconnect and much smaller buffer resources for long distance interconnect. To maintain flexible routing capability, each neighboring processor pair has multiple connecting links. The architecture supports long distance communication in GALS systems by transferring the source clock with the data signals along the entire path for write synchronization. Compared to a traditional dynamically-configurable interconnect architecture with symmetric buffer allocation and single-links between neighboring processor pairs, this implementation has approximately two times smaller communication circuitry area with a similar routing capability. Area and speed estimates are obtained with the physical design of seven chips in 0.18-μm CMOS.**

***Index Terms*—Chip multiprocessor, globally asynchronous locally synchronous (GALS), inter-processor interconnect, many-core, multi-core, network-on-chip (NoC).**

I. Introduction

INTEGRATING multiple processors into a single chip (known as chip multiprocessors or CMPs) has recently become easily achievable and common due to continuing advances in VLSI fabrication technologies [1], [2]. This fact makes interprocessor communication in chip multiprocessors an important design issue.

Wires in deep-submicrometer CMOS fabrication technologies are introducing greater: relative delay, relative power consumption, and timing and power variations which is causing traditional on-chip communication methods such as a global bus structures to meet considerable challenges. Researchers have proposed network-on-chip (NoC) solutions which use routers for inter-processor communication. Most research is based on dynamic packet-switched routing architectures [3], [4]. Another approach is the statically configurable *nearest-neighbor* interconnect architecture [5], [6], where each processor communicates with only its four nearest neighbors in 2-D meshes and long distance communication is accomplished by software in intermediate processors. Other designs [7] use both dynamic and static interconnects.

Although both dynamic routing architectures and static nearest neighbor interconnect architectures achieve significant success in specific areas, they have some limitations. Dynamic routing architectures are flexible, but normally require relatively large circuit area and power for communication circuitry. The static nearest neighbor interconnect architecture reduces area and power requirements significantly, but it results in relatively high latency for long distance communication.

Communications within chip multiprocessors for many applications, especially many digital signal processing (DSP) algorithms, are often largely localized [8], [9]: most communication is among nearest (or local) neighbors while a small portion is long distance. Motivated by this fact, we propose an *asymmetric* structure to obtain good tradeoffs between flexibility and cost by: treating the nearest neighbor communication and long distance communication differently, using more buffer resources for nearest neighbor connections, and using fewer buffer resources for long distance connections [10]. Together with the relatively simple static routing approach, this asymmetric architecture can achieve low area cost for communication circuitry.

Under the static asymmetric architecture, there are a couple of design options available such as the number of input ports (buffers) for the processing core; and the number of links between each neighboring processor pair. The area, speed, and performance of different design options are analyzed, and some conclusions based on the results are drawn. We found that increasing the number of links between processors is helpful to increase routing capability, but it dramatically increases processor area after a certain point which depends on implementation details. Two or three links are generally appropriate when each processor in the chip utilizes a simple single-issue processor architecture.

Moreover, the proposed architecture supports the globally asynchronous locally synchronous (GALS) [11] clocking style which allows each processor to operate in its own clock domain and avoids the design of a global clock tree, which can significantly simplify the clock system design and potentially reduce system power consumption. After examining the characteristics of different approaches, we propose a source synchronous method which transfers the clock with the data and control signals along the entire path to the destination processor.

Compared to traditional dynamically configurable interconnect architectures with symmetric buffer allocation and single

Manuscript received July 07, 2008; revised December 12, 2008. First published August 04, 2009; current version published April 23, 2010. This work was supported in part by Intel, by UC MICRO, by NSF Grant 0430090, by CAREER Award 0546907, by SRC GRC Grant 1598.001, by CSR Grant 1659.001, by ST Microelectronics, by Intellasys, by S Machines, by Artisan, and by State Key Laboratory of ASIC & System (Fudan University) ZD20080103 and KF2008405.

Z. Yu is with the State Key Laboratory of ASIC & System, Microelectronics Department, Fudan University, Shanghai 201203, China (e-mail: zhiyiyu@fudan.edu.cn).

B. M. Baas is with the Electrical and Computer Engineering Department, University of California, Davis, CA 95616 USA (e-mail: bbaas@ucdavis.edu).

Digital Object Identifier 10.1109/TVLSI.2009.2017912

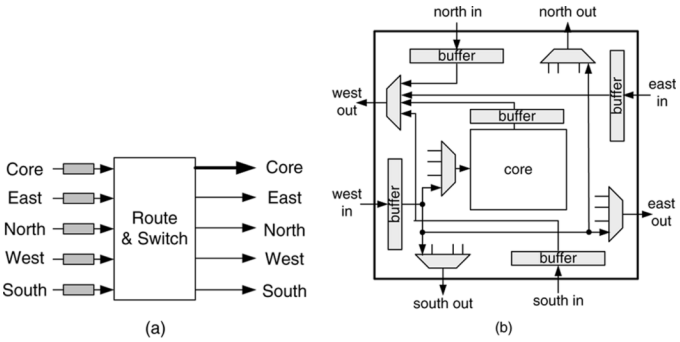


Fig. 1. (a) Illustration of interprocessor communication in a 2-D mesh and (b) a generalized communication routing architecture in which only signals related to the west edge are drawn.

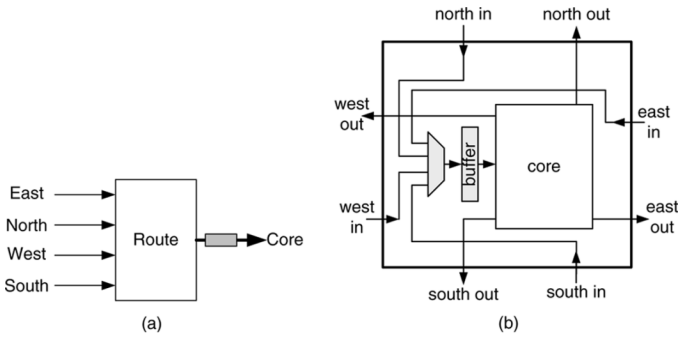


Fig. 2. Circuitry diagrams of the static nearest neighbor interconnect architecture. Data from four inputs are transferred only to the processing core to reduce the circuitry cost, and only a single buffer is needed.

links between each neighboring processor pair, a presented implementation example of the proposed architecture shows that it can reduce the communication circuitry area by approximately two times with similar routing capability.

The structure of the remainder of this paper is as follows. Section II describes the proposed statically configurable asymmetric interconnect architecture. Section III explores several design options. Section IV investigates the approaches to support the long distance GALS clocking style. Section V discusses an implementation example and its results. Finally, the conclusion summarizes the key contributions.

A. Background: Traditional Dynamic Routing Architecture

Fig. 1(a) shows the interprocessor communication in a typical 2-D mesh-connected chip multiprocessor using a router architecture. The router block in each processor receives data from the processor core and neighboring processors (east, north, west, and south) and then sends data to the processor core or neighboring processors. Since communication links are not always available for data transfers due to slow data processing speeds or link congestion, buffers are inserted at each input edge [12]. Fig. 1(b) shows a generalized diagram of the routing circuitry where only signals related to the west edge (*west in* and *west out*) are drawn. Input ports from each side feed data into a corresponding buffer, and the buffers supply data to the processor core or other output ports. Each output port selects data from the processor core and three input buffers. As the diagram shows, the communication logic includes five buffers and five muxes, and there is some control logic to support the communication flow control which is not drawn. Other implementations are possible; for example, each output port can also have a buffer, or each input buffer can be split into multiple virtual channels [13] to reduce communication congestion and hence reduce communication latency. The area of the communication circuitry is normally dominated by the buffers, and the logic in the four input/output edges is normally the same.

B. Background: Static Nearest Neighbor Interconnect

In many applications, communication in chip multiprocessors is localized or can be localized, which means data traffic going into the processor core is much larger than to the other paths. This is especially true for homogeneous processor arrays assuming tasks are mapped in a way that minimizes distances of inter-processor data traffic. To minimize communication circuit overhead, another inter-processor communication strategy is to implement only nearest neighbor interconnect logic, and thus, long distance communication is performed by software in the intermediate processors. Fig. 2(a) shows the concept of nearest neighbor interconnect and Fig. 2(b) shows its circuit diagram. All data from the input ports are transferred to the processing core, so instead of inserting a buffer at each input port, there is little change to system performance by inserting a buffer(s) only at the output(s) of the router to the processing core. Comparing Figs. 1 and 2, the static nearest neighbor interconnect reduces the number of buffers from five to one and the muxes for each output port are all avoided; resulting in more than five times smaller area. But clearly it has the limitation that long distance communication places a burden on intermediate processors.

II. Low-Area Interconnect Architecture

This section introduces the proposed statically configurable low-area asymmetric interconnect architecture.

A. Asymmetrically-Buffered Architectures

1) Asymmetric Data Traffic Typically Exists at the Router's Output Ports: The case of varying buffer allocation for *input buffered routers* to match asymmetric inter-processor data traffic loads has been shown to achieve some benefits [14].

In contrast, this paper presents asymmetric buffer allocation for *output buffered routers* [15] because we find the asymmetric data traffic on the routers' outputs are more uniform across different applications and hence the architecture is helpful across a wider range of applications. The asymmetric traffic focuses on the differences in traffic going to the processor core versus output ports connected to other processors.

Table I shows the data traffic of each processor for a nine-processor JPEG encoder as shown in Fig. 3, [16] which demonstrates the different asymmetric data traffic on the input-buffered and output-buffered routers.

Considering the router's *input* ports, although each processor shows a clear asymmetric communication data traffic load; the major input direction for different processors are different which makes the overall traffic at the input ports within a factor of five in this example—the relative input traffic for the east, north,

TABLE I
DATA TRAFFIC OF A 9-PROCESSOR JPEG ENCODER AS SHOWN IN FIG. 3 TO PROCESS ONE 8×8 BLOCK, ASSUMING FIVE-INPUT FIVE-OUTPUT ROUTERS AS SHOWN IN FIG. 1 IN EACH PROCESSOR. 80% OF THE DATA FROM INPUTS ARE DELIVERED TO THE PROCESSING CORE WHICH DOMINATES THE TRAFFIC AT THE ROUTERS' OUTPUT PORTS

	Network data words at input ports of router:				Network data words at output ports of router:				
	East	North	West	South	Core	East	North	West	South
Proc. 1	0	64	0	0	64	0	0	0	0
Proc. 2	0	64	0	0	64	0	0	0	0
Proc. 3	0	64	0	0	64	0	0	0	0
Proc. 4	0	0	64	0	64	64	0	0	0
Proc. 5	0	0	96	0	64	0	32	0	0
Proc. 6	0	0	0	64	1	0	0	63	0
Proc. 7	0	0	0	3	3	0	0	0	0
Proc. 8	63	0	0	0	63	0	0	0	0
Proc. 9	4	0	0	252	256	0	0	0	0
Total	67	192	160	319	643	64	32	63	0
Rel. input	9%	26%	22%	43%					
Rel. output					**80%**	8%	4%	8%	0%

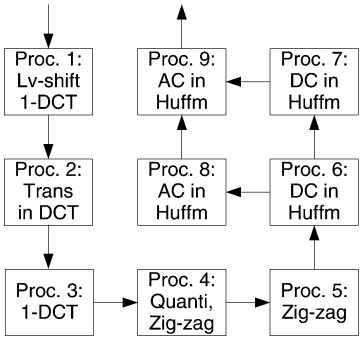


Fig. 3. Nine-processor implementation of a JPEG encoder core [16].

west, and south directions are 9%, 26%, 22%, and 43%, respectively. Therefore, to optimize buffers in this approach would require the customization of individual buffer sizes on each processor which would then unfortunately optimize the design for only one (or a small number) of applications.

On the other hand, considering the *output* ports, each processor shows a similar asymmetric data traffic: most of the data from the input ports are delivered to the core (for local processing) and very little is delivered to the edges (for long distance communication), and overall about 80% of the data are delivered to the core. Thus a single asymmetric output-buffered router can be widely suitable for different applications, which is important since multi-core chips utilizing NoC architectures are typically used widely across a number of application domains.

2) Proposed Asymmetric Architecture: We propose an architecture which has asymmetrically-buffered output ports as shown in Fig. 4 to achieve good tradeoffs between cost and flexibility. As shown in Fig. 4(a), instead of equally distributing buffer resources to each output port, we allocate a relatively large buffer to the processing core port, and smaller buffers (one or several registers) to the other ports. Fig. 4(b) shows the circuitry diagram where only signals related to the west edge (*west in* and *west out*) are drawn. This architecture's circuit area is similar to the nearest neighbor interconnect architecture, shown in Fig. 2, since it adds only a few registers and muxes. From the point of view of routing capability, this architecture is similar

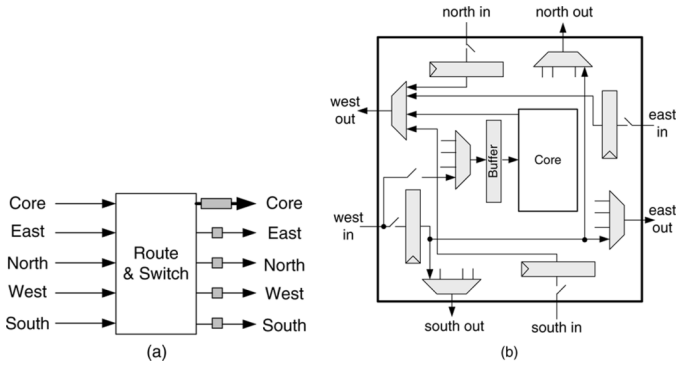


Fig. 4. Concept and circuitry diagram of the proposed inter-processor communication architecture; it has the asymmetric buffer resource for the long distance interconnect and the local core interconnect.

to the traditional dynamic routing architecture, shown in Fig. 1, since reducing the buffers in ports for long distance communication does not significantly affect system performance when the communication is localized. With its one large buffer for the processing core, the proposed architecture can save about five times the area compared to the traditional dynamic architecture shown in Fig. 1. If using two ports and two buffers per processor core as discussed in Section III-A, the required area is about two times lower.

B. Static Routing Versus Dynamic Routing

The inter-processor interconnect can be configured statically before runtime (static routing), or dynamically at runtime (dynamic routing). Dynamically-routed networks have been commonly used in multiprocessor systems such as those utilizing message passing methods. Moreover, dynamic networks have been rigorously studied in NoC research, but statically-configured architectures have been much less intensively studied. The key advantage of the static configuration approach is that for applications with predictable traffic, such as most DSP applications, it can provide an efficient solution with small area cost and communication latency. The dynamic configuration solution can effectively address more applications because of its flexibility, but it has non-negligible overhead in terms of the circuitry area and the communication latency; the main overhead comes from the routing path definition, the arbiter of multiple independent clock sources, and the signal recognition at the destination processor.

1) Dynamic Routing and Its Overhead: In dynamic routing, the data transfer path should be defined by the source processor and propagated to the corresponding downstream processor(s) or dynamically decided by intermediate processors. The circuitry to define and control the routing path has an area overhead, and to propagate the routing path might cost extra instructions and increase the clock cycles for the data transfer.

Since each link in the dynamic routing architecture is shared by multiple sources, an arbiter is required to allow only one source to access the link at one time. Furthermore, in GALS chip multiprocessors, this arbiter becomes more complex since it must handle multiple sources with unrelated clock domains. An obvious overhead is that some synchronization circuitry is required for the arbiter to receive the link-occupying request from

different sources, and some logic is required to avoid glitches when the occupying path changes.

Another important issue is how the destination processor can identify the source processors of the received data. Since data can travel through multiple processors with unknown clock domains, it is not possible to assume a particular order for the incoming data. One common method is that an address is assigned to each processor and sent along with the data, and the destination processor uses the address to identify the source processor through software or hardware.

Combining these overheads, the communication latency for dynamic routing between adjacent processors has been estimated to be typically larger than 20 clock cycles [17], and this value will increase further for GALS dynamic routing networks due to the additional synchronization latency.

2) Static Routing: Due to its smaller circuit area and excellent compatibility with GALS-clocked systems, we investigate only the static routing approach in the remainder of this paper.

Few multi-processor systems use static routing, and the Systolic [18] approach is one of the pioneers. Systolic systems contain synchronously-operating processors which "pump" data regularly through a processor array, and the data to be processed must reach the processing unit at the exact predefined time. Due to this strict requirement for data streams, the systolic architecture is well suited only for applications with highly regular communication patterns such as matrix multiplication.

Releasing the strict timing requirement of the data stream can significantly broaden the application domain. To release the systolic system's strict cycle-by-cycle timing requirements, each processor must "wait" for data when the data is late, and the data must "wait" to be processed when it comes early. Inserting a first-input–first-output (FIFO) with appropriate *full* and *empty* logic [19] at each input of the processing core can meet these requirements. Data is buffered in the FIFO when it comes early, the downstream processor is stalled when the FIFO is empty and there is a read request, and the upstream processor is stalled when the FIFO is full and there is a write request. In this way, the requirement for the data stream is only its order, not its exact arrival time.

RAW [17] is a chip multiprocessor with very low latency for interprocessor communication (three clock cycles) using both static routing and dynamic routing, but it achieves this goal with a large area cost of about 4 mm^2 in a 0.18-μm CMOS technology. The communication circuitry we propose is suitable for broad applications, with low latency (about five clock cycles), and low area overhead (about 0.1 mm^2 in 0.18-μm technology). The detailed implementation is described in Section V. Table II compares the interconnect architectures discussed in this section.

TABLE II
COMPARISON OF SEVERAL ROUTING APPROACHES. THE AREA DATA ARE FOR 0.18-μM CMOS TECHNOLOGY, AND THE VALUES CAN VARY IN SPECIFIC IMPLEMENTATIONS DUE TO MANY FACTORS

Routing method	Suitable applications	Latency (clk cycles)	Area (mm^2)
Static systolic	limited	~1	N/A
Dynamic	broad	>20	N/A
Static + dynamic [17]	broad	3	~4
Proposed asymmetric static	broad	~5	~0.1

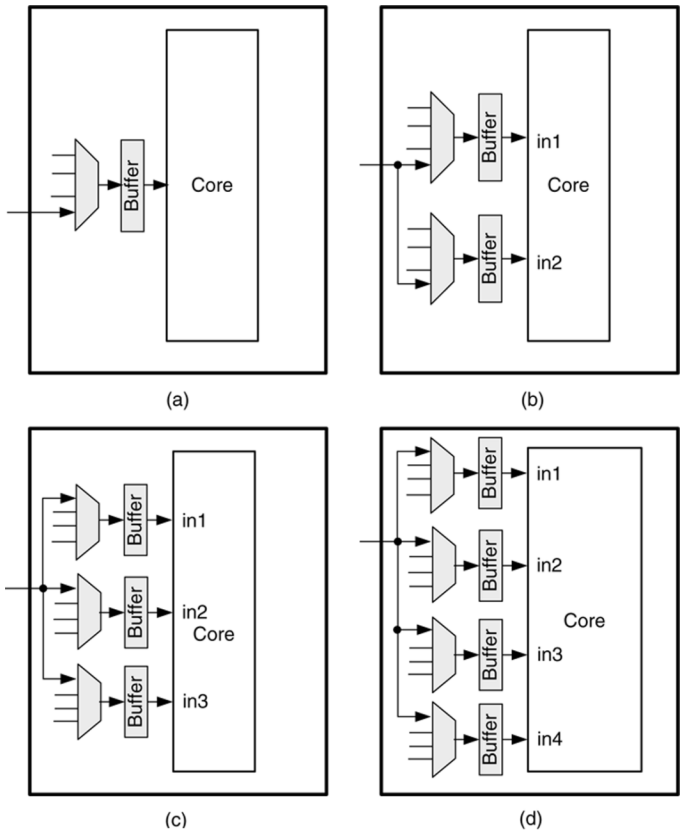


Fig. 5. Diagrams of architectures with various numbers of input ports (and therefore various numbers of buffers) for the processing core: (a) single port; (b) two ports; (c) three ports; and (d) four ports.

III. DESIGN SPACE EXPLORATION

Under the statically configurable asymmetric architecture discussed in Section II, there are several options for the communication logic realization. Two important options are investigated in this section, including the following:

- the number of ports (buffers) for the processing core;
- the number of links between each neighboring processor pair.

A. Single Port Versus Multiple Ports for the Processing Core

In Fig. 4, the processing core has one input port (and one corresponding buffer). When using the dynamic routing approach, a single port might be sufficient since the processing core can fetch data from all directions by dynamically configuring the input mux. If using the static routing approach, a single port means each processing core can fetch data from only one source, which might be inefficient when multiple sources are required. Using one, two, three, or four ports (buffers) for the processing core is considered, as shown in Fig. 5. The area of the communication circuitry scales roughly linearly with the number of ports (buffers).

1) Performance Evaluation: The effect of the number of buffer(s) to the routing capability is highly dependent on the application communication patterns. We use the worst case communication distance of fundamental and useful communication patterns, including one-to-one communication (in which two processors at opposite corners of the processor array communicate with each other), one-to-all broadcast (in which one corner

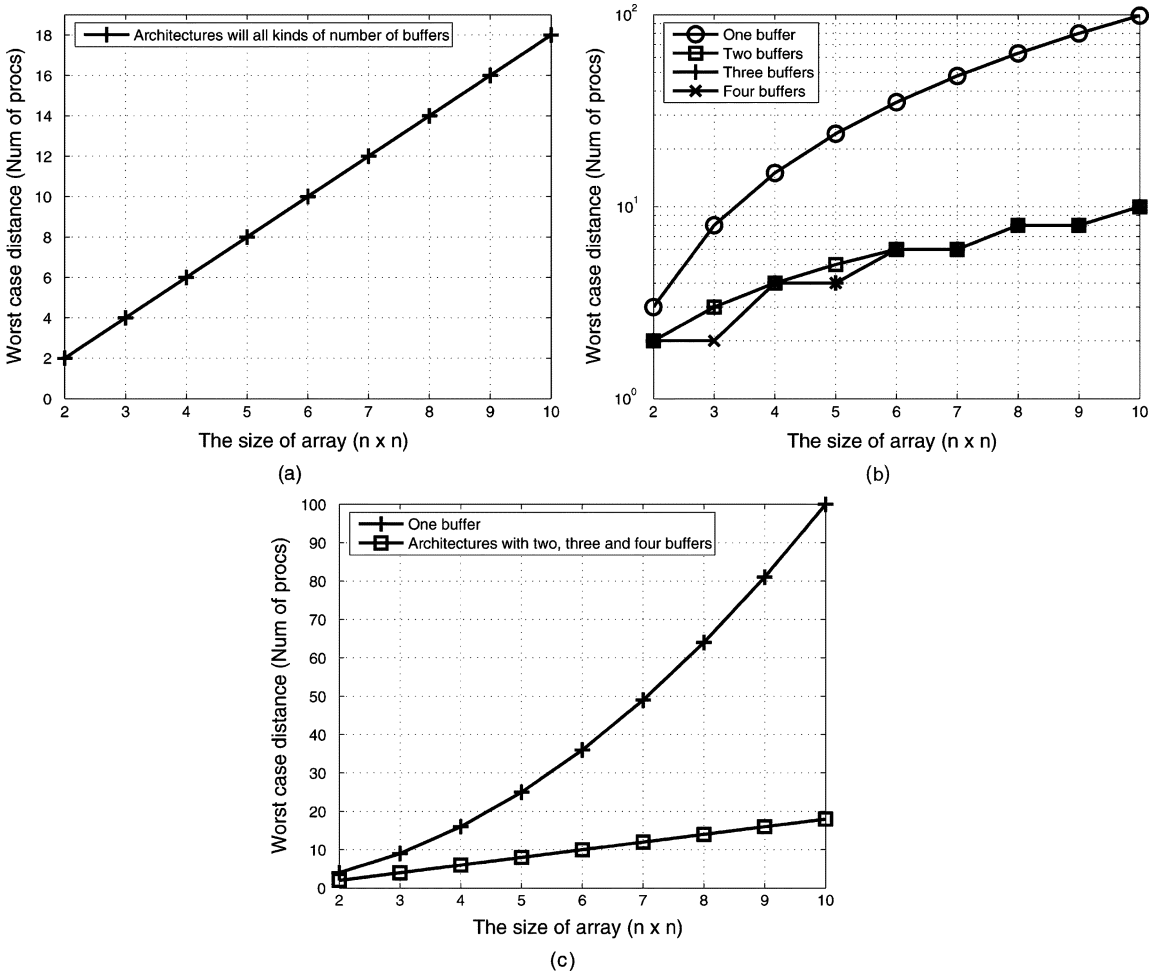


Fig. 6. Comparing the worst case communication distance across a 2-D mesh of processors for various architectures with different numbers of ports (buffers) for the processing core, by varying the size of the array and using different basic communication patterns: (a) one-to-one and one-to-all; (b) all-to-one; and (c) all-to-all.

processor sends data to all processors), all-to-one merge (in which all processors send data to a processor at the middle), and all-to-all communication to evaluate the performance of different architectures. Real applications can normally be modeled by a combination of these basic patterns.

For one-to-one and one-to-all communications where the destination processor(s) need(s) only one source, the single-port architecture has the same performance as other multiple-port architectures. The worst case distance is approximately $2n$ processors in an $n \times n$ array, as shown in Fig. 6(a).

For the all-to-one communication case, shown in Fig. 6(b), increasing the number of buffers (ports) has a benefit. For the single-port architecture, each processor can receive data from only one source and the furthest processor must propagate data through all n^2 processors, since all processors must be arranged in a linear array. For architectures with multiple ports (buffers), the communication can be distributed in multiple directions and the furthest processor only needs to propagate through order n processors. The architectures with 2, 3, or 4 ports (buffers) perform similarly; as shown in Fig. 6(b), the two-port architecture is slightly worse in the 5×5 array and the four-port architecture is slightly better in the 3×3 array.

Besides the worst case communication distance, another consideration is how fast the destination processor can *process* the received data. A single-issue simple processor can normally consume no more than two data words in each clock cycle, which means it has little benefit to accept more than two data words in each cycle, hence it also means to use three or four buffers for the processing core can not provide a significant benefit. Of course processor designs which process more than two input operands per cycle, such as VLIW or superscalar, would yield different results.

The all-to-all communication differs from the all-to-one communication in that each processor must reach all other processors. The architecture with one port (buffer) must arrange all processors linearly and the worst case distance is to propagate through all n^2 processors. Architectures with multiple buffers (ports) can communicate in two dimensions and their worst-case distance is through one row and one column of about $2n$ processors. Fig. 6(c) shows the results.

According to the results in Fig. 6 and considering the trade off between area and performance, using two buffers for the processing core is a good solution. A natural extension is that if processors are in a 3-D mesh topology, then the best solution will likely be a three-port architecture.

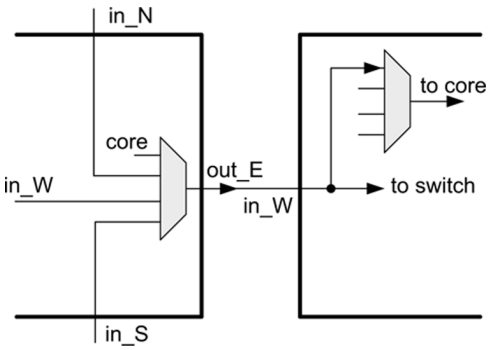


Fig. 7. Diagram of inter-processor connections with one link at each edge.

B. Single Link Versus Multiple Links

One of the key differences between on-chip and inter-chip interconnects is that there are more wire resources on chip, while inter-chip connections are normally limited by the available chip IO pins. One approach is to increase the wordwidth of each link in NoCs to take advantage of this fact [3]. We explore another option in this section of increasing the number of links at each edge to increase connection capability and flexibility. The difference between this multi-link architecture and virtual channels [13] is that virtual channels increase the number of buffers for each link while the multi-link architecture increases the number of connecting links between neighboring processors.

1) Single Link: When there is one link at each edge, each output link can potentially receive data from the other three edges and the processing core. Fig. 7 shows its diagram. It is a relatively simple and straightforward architecture.

2) Double Links: If using double links at each edge, there are eight sources for the processing core, and each edge has two outputs and each output potentially has seven sources (six from other input ports and one from core). Comparing the fully-connected two-link architecture shown in Fig. 8(a) and the single-link architecture in Fig. 7, the overhead of increasing the number of links is high not only because of necessary control logic, but more importantly because of semi-global wires inside each processor which affect system area, speed, and power significantly in submicrometer technologies.

Methods are available to simplify the fully connected architecture. Considering the router's logic at the east edge which receives data from *North, West, South*, and *Core* and sends to *East* output, we use $\{\text{in1\_}N, \text{in2\_}N, \text{in1\_}W, \text{in2\_}W, \text{in1\_}S, \text{in2\_}S, \text{sig\_core}, \text{out1\_}E, \text{and out2\_}E\}$ to define these signals. A large exploration space exists at a first glance since 7 inputs and 2 outputs have 2^{14} connecting options. Since the three input edges are symmetric, we group $\{\text{in1\_}N, \text{in1\_}W, \text{in1\_}S\}$ together as input *in1* and $\{\text{in2\_}N, \text{in2\_}W, \text{in2\_}S\}$ as *in2*, so the input number is reduced to 3 and the exploration space is reduced to $2^6 = 64$ as shown in Table III. In options 0–7, *out1* does not have any connections so they are not considered as two-link architectures. Option 8 is neglected for a similar reason. Option 9 only connects the processing core to the outputs and it is essentially the same as the nearest neighbor architecture. Option 10 can not be realized since *out2* is connected only with *in2* which means this link has no original source. Option 11 can be a potential choice, where *out1* is connected only to the core served as the nearest neighbor link, and *out2* is connected to *in2* and core served as the long distance connect link. Fig. 8(b) shows the circuit diagram; each edge contains only one 4-input mux. By examining all other options, we found that option 14 and 43 are also potential good choices. In option 14, *out1* (link1) receives data from the core while *out2* (link2) receives data from edges, both of them send data to the core and routers. Fig. 8(c) shows the circuit diagram; each edge contains one 6-input mux. In option 43, each link receives data from the core and a *corresponding* link (*out1* corresponds to *in1* while *out2* corresponds to *in2*), and sends data to the core and routers.

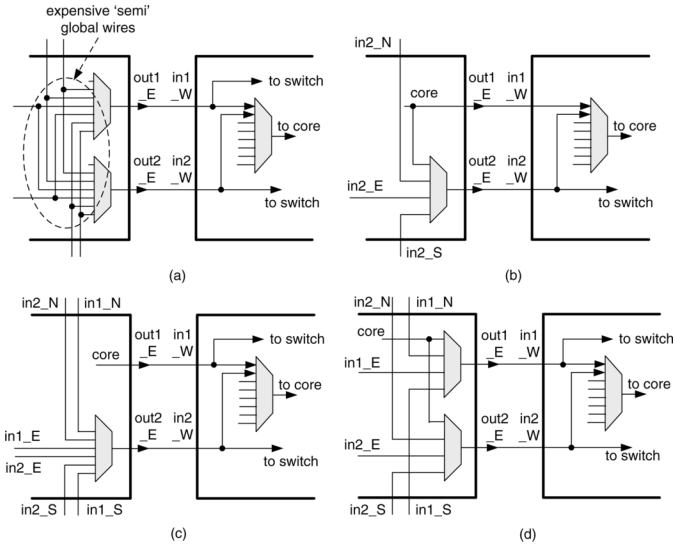


Fig. 8. Four inter-processor interconnect schemes utilizing double links: (a) fully connected; (b) separated nearest neighbor link and long distance link; (c) separated link from core and link from edges; and (d) connections exist between "corresponding" links. Methods (a), (b), (c), and (d) correspond to options 63, 11, 14, and 43 in Table III.

TABLE III
ALL POSSIBLE INTERCONNECT OPTIONS FOR DOUBLE-LINK ARCHITECTURES EXAMINING ALL COMBINATIONS WHERE CONNECTIONS EXIST OR DO NOT EXIST BETWEEN ANY INPUT (IN1, IN2, AND CORE) AND OUTPUT (OUT1 AND OUT2). *YES* MEANS A CONNECTION PATH EXISTS, *NO* MEANS NO CONNECTION PATH EXISTS, AND "—" MEANS "DON'T CARE" FIELDS FOR MULTIPLE MATCHES

Option number	in1-out1	in2-out1	core-out1	in1-out2	in2-out2	core-out2
0–7	No	No	No	–	–	–
8	No	No	Yes	No	No	No
9	No	No	Yes	No	No	Yes
10	No	No	Yes	No	Yes	No
11	No	No	Yes	No	Yes	Yes
12	No	No	Yes	Yes	No	No
13	No	No	Yes	Yes	No	Yes
14	No	No	Yes	Yes	Yes	No
15	No	No	Yes	Yes	Yes	Yes
16–31	No	Yes	–	–	–	–
32–39	Yes	No	No	–	–	–
40	Yes	No	Yes	No	No	No
41	Yes	No	Yes	No	No	Yes
42	Yes	No	Yes	No	Yes	No
43	Yes	No	Yes	No	Yes	Yes
44–47	Yes	No	Yes	Yes	–	–
48–62	Yes	Yes	–	–	–	–
63	Yes	Yes	Yes	Yes	Yes	Yes

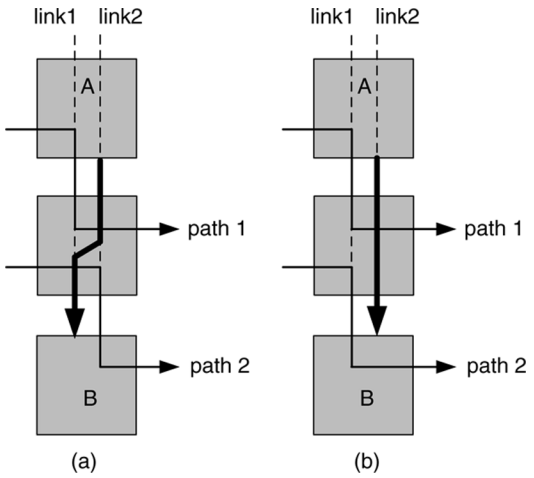


Fig. 9. Example route of a communication path from processor A to processor B (a) using a fully connected architecture as shown in Fig. 8(a), and (b) with reorganized conflicting paths using the simpler architecture shown in Fig. 8(d).

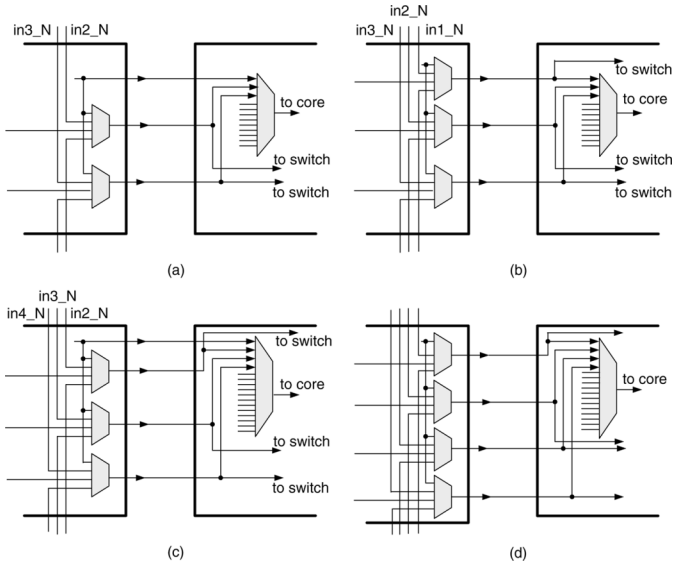


Fig. 10. Inter-processor interconnect with three and four links. (a) and (c): separated nearest neighbor link and long distance link, similar with Fig. 8(b); (b) and (d): connections exist between "corresponding" links; similar with Fig. 8(d).

Fig. 8(d) shows the circuit diagram; each edge contains two 4-input muxes.

In terms of the area cost and the routing flexibility of the four architectures shown in Fig. 8, architecture (a) has the most flexible connection while it has the largest cost; architecture (b) has the most strict connection limit while it has the smallest cost; and architecture (c) has the connection flexibility and cost in between (a) and (b). Architecture (d) has an area cost similar with (c), and interestingly, its routing capability is the same as architecture (a). This concept can be demonstrated by Fig. 9 where A needs to communicate with B, while path1 and path2 occupy some parts of the links between them. For the fully connected architecture, the path between A and B is easy to setup since it can first use link2 and then switch to link1 shown in Fig. 9(a). Using the architecture in Fig. 8(d) has difficulty at first glance but it can be handled by rerouting path1 and then using link2 as the path between A and B, achieving the same routing purpose as the fully connected architecture shown in Fig. 9(b).

According to these discussions, Fig. 8(d) architecture is a good option due to its routing flexibility and moderate circuitry cost; and Fig. 8(b) can also be a good option due to its small circuitry cost.

3) Three or More Links Per Edge: Increasing the number of links beyond two per edge increases the routing capability further. Fig. 10 shows architectures with three and four links extended from designs shown in Fig. 8(b) and (d) architectures. The circuitry clearly becomes more complex along with the increased number of links. Each edge contains 2 or 3 4-input muxes in three-link architectures and contains 3 or 4 4-input muxes in four-link architectures. Also, the number of sources for the processing cores becomes 12 and 16 in three-link and four-link architectures. Since the wires are all semi-global, the overhead of these additional connections has a significant effect on the system area and speed.

Since the architectures with various links have a large number of options and is not easy to judge according to the qualitative analysis. A more quantitative analysis is given in the following sections.

4) Area and Speed Estimates From Seven Chip Designs: Increasing the number of communication links requires additional control logic, which increases circuitry area and affects processor speed. Seven types of architectures will be investigated further which are as follows.

Type 1)Single-link in Fig. 7.
Type 2)Double-link in Fig. 8(b).
Type 3)Double-link in Fig. 8(d).
Type 4)Three-link in Fig. 10(a).
Type 5)Three-link in Fig. 10(b).
Type 6)Four-link in Fig. 10(c).
Type 7)Four-link in Fig. 10(d).

Type 2 corresponds to Option 11 and Type 3 corresponds to Option 43 in Table III.

All seven architectures were designed and written in verilog and synthesized with a 0.18-μm CMOS standard cell library. Synthesis area reports show communication logic area for the seven architectures are 0.013 mm^2, 0.032 mm^2, 0.047 mm^2, 0.058 mm^2, 0.067 mm^2, 0.075 mm^2, and 0.081 mm^2, respectively. However, the results from synthesis do not tell the entire story since they do not put the communication circuitry into an entire processor environment and consider physical layout effects. In deep submicrometer technologies, wires introduce non-negligible delay compared to logic gates; and complex wire connections require significant area for routing.

We therefore have designed these seven communication architectures into a simple single-issue processor with an original area of about 0.66 mm^2 per processor in a 0.18-μm technology [16]. The seven chips were synthesized from RTL code and taken through the entire physical layout design flow using Cadence Encounter. Fig. 11 shows the layouts of the seven processors. Standard cell area utilization in the physical design can have a strong impact on the design result. In terms of area, higher utilization is always good as long as the chip is routable. In terms of speed, too low of a utilization introduces long wires and reduces system speed; too high of a utilization can result in

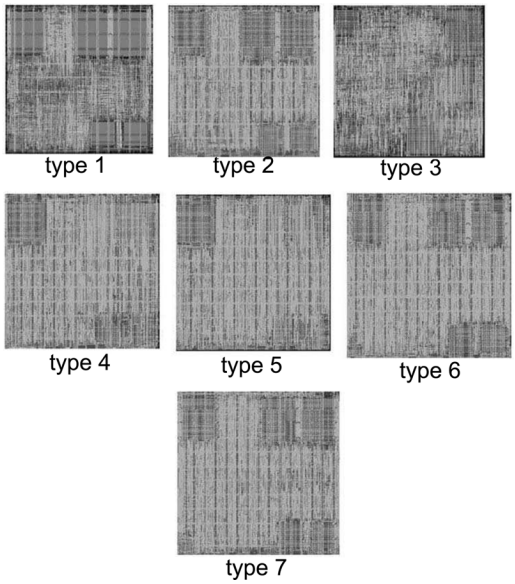


Fig. 11. Scaled layouts of seven processors containing different numbers of communication links; types 1–7 correspond to the architectures shown in Figs. 7, 8(b), (d) and the four in Fig. 10, respectively.

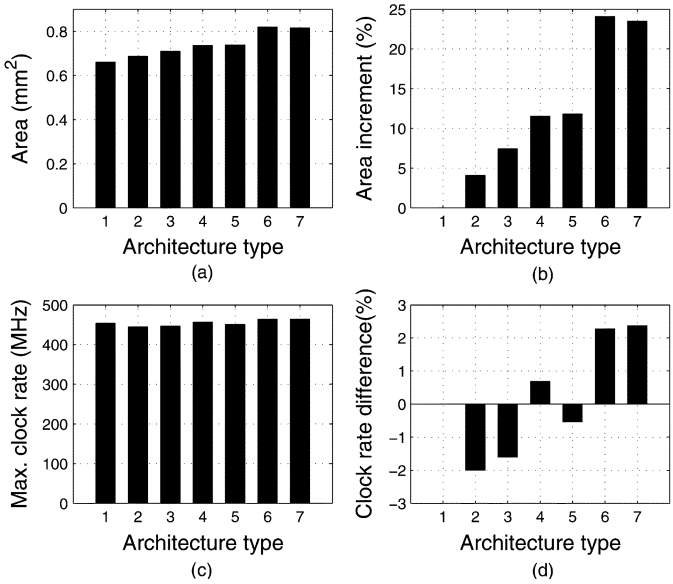


Fig. 12. Comparison of designed processors with the seven communication link types showing: (a) absolute area; (b) area increment relative to type 1; (c) absolute speed; and (d) speed relative to type 1. Types 1 to 7 correspond to the architectures shown in Figs. 7, 8(b), (d), and 10; type 6 and 7 (four-link architectures) have a noticeable area increase due to their semi-global wire effect on the processor area.

wiring congestion and complicated or impossible wire routing. We found a good approach was to set utilization to 70% initially and then allow it to increase to approximately 85% after clock tree insertion and in-place optimization.

Fig. 12(a) shows the area of these seven chips and Fig. 12(b) shows the relative area increment of each architecture compared to the type 1 architecture. Types 6 and 7 (four-link architectures) have a noticeable nearly 25% increase in area because they can not successfully complete routing until the area utilization is reduced to 64% and 65%, respectively. This result provides some interesting insight into how many global wires can fit into a chip. For a processor with a 0.66 mm^2 area and a 0.80 mm edge, assuming a minimum 1 μm pitch between IO pins, an optimistic estimation is that each edge can fit 800 IO pins consuming one perpendicular metal layer, beyond this range the processor size will become *IO pin or wire dominated*. This estimation can be true if these IO pins are all connected to short wires. For global wires used for communication routers, increasing the number of wires will quickly result in routing congestion and increase the processor size. In our example, each processor edge in four-link architectures has about 160 IO pins, much less than the optimistic 800 IO pins. Four or more communication link architectures are less desirable due to their high area cost.

Fig. 12(c) shows the processors' speeds and Fig. 12(d) shows the relative speed difference of each architecture compared to the type 1 architecture. Each processor has a similar speed and the difference is within $\pm 2\%$. Types 6 and 7 have a little faster speed compared to Type 1 because the released area helps to simplify the routing.

5) Performance: Again, we use basic communication patterns to evaluate the performance of architectures with different numbers of links.

The communication latency between two processors can be expressed as $k \times p$ clock cycles where p is the distance between source and destination and k is the clock cycles across one processor. For the static nearest neighbor interconnect architecture, crossing one processor requires extra instructions and the latency (k) is dependent on the asynchronous synchronization time and the processor pipeline depth; it is 13 in our experimental system. If the communication can use the router in the proposed routing architecture, k is 1 if the data is registered in each processor and less than 1 if some processors are bypassed as discussed in Section IV.

For one-to-one or one-to-all communication, each processor requires only one source so that the single-link architecture is sufficient and has the same communication latency with other architectures, as shown in Fig. 13(a). A little surprisingly, all architectures have the same latency for all-to-all communication, as shown in Fig. 13(c), because each processor needs data from both horizontal and vertical neighbor processors and both of the two buffers (ports) of each processor are occupied, prohibiting the usage of the additional links for long distance communication.

These architectures have different communication latencies in all-to-one communication as shown in Fig. 13(b). For the Type 1 (single-link) architecture, most or all of the link resources are occupied by the nearest neighbor interconnect and little can be used for the long distance communication, so the latency is relatively high. Increasing the number of links helps when the latency is limited by the link resources. Type 2 (double links with separated nearest neighbor link) has little advantage to Type 1 with a relatively much higher area, and Type 4 (three links with separated nearest neighbor link) has little advantage to Type 3 (double links) but with a relatively much higher area, so that Types 2 and 4 are not considered. The Type 3 architecture is about two times faster than the single-link architecture, which makes it a good candidate. Comparing Type 5 (three links) architecture with Type 3, they have the same

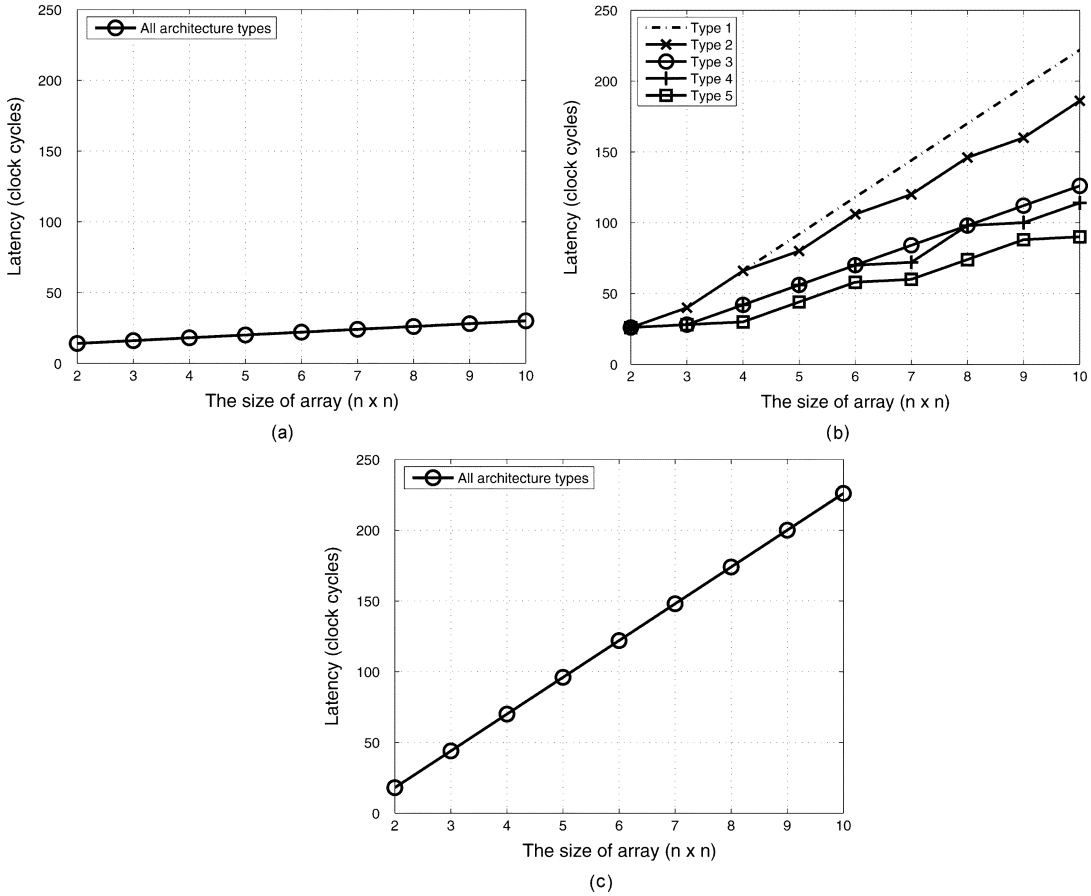


Fig. 13. Comparing the communication latency (clock cycles) of interconnect architectures Types 1 to 5, by varying the size of the array and using different basic communication patterns: (a) one-to-one and one-to-all; (b) all-to-one; and (c) all-to-all. Types 6 and 7 architectures are not included in the comparison due to their high area cost as shown in Fig. 12.

latency within a small communication domain (2×2 and 3×3 arrays), while the three-link architecture benefits when the array grows. For 4×4 to 6×6 arrays, the three-link architecture has about 25% smaller latency; for 7×7 to 9×9 arrays, it has about 30% smaller latency, and the benefit increases along with the larger array sizes.

6) More Advanced Fabrication Technology: Although the seven processors were designed in a single 0.18-μm technology, it stands to reason that the relative area and performance will scale well across many technology generations. The two-link Type 3 design in Section III-B4 has been fabricated and is fully functional in an advanced 65-nm technology [20], [21]. The logic for I/O and interconnect router occupies about 5% of the processor area, which is similar to the result we obtained in 0.18-μm technology.

IV. Support for GALS Clocking Styles

Traditional globally synchronous clocking circuits have become increasingly difficult to design with growing chip sizes, clock rates, relative wire delays, parameter variations, and clock power consumption. The GALS clocking style [11] separates processing blocks such that each block is clocked by an independent clock domain, thus avoiding the necessity of designing a global clock tree.

Some research exists in investigating clocking issues for networks on chip. Vangal *et al.* report an 80-core 2-D array with packet-switched routers that employ mesochronous clocking where each processor has the same clock frequency but with potentially different phases [4]. Liang *et al.* propose an asynchronous network-on-chip (FAUST) which uses a handshaking scheme to handle asynchronous communication [22]. Zhang *et al.* developed a reconfigurable DSP for wireless baseband digital signal processing, and handshake style GALS signaling was adopted to allow various modules to operate at different and dynamically-varying rates [23].

A. Source Synchronous Flow Control for General GALS Systems

GALS systems introduce modules with different clock domains and the communication between those modules requires special concerns. The methods can be classified into two categories. The first method is to use an asynchronous handshake [24] as shown in Fig. 14(a). The source sends a *request* signal and one single datum and waits until it receives an *acknowledge* signal from the destination before beginning another transaction. A corresponding round-trip latency exists for each data transfer in this method.

To sustain higher throughput, a *source synchronous flow control* method can be used as shown in Fig. 14(b), where the clock of the source processor travels along with the data and signals to control the writing into a buffer. Here the data can be transmitted at a rate of one word per clock cycle as long as the buffer

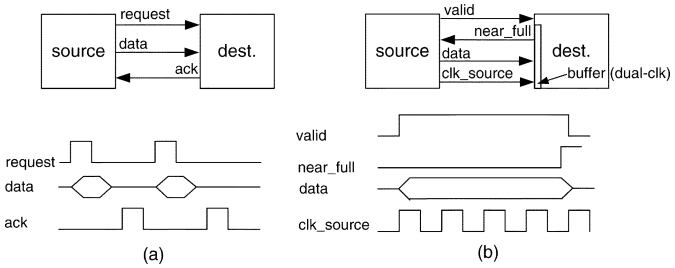


Fig. 14. Two strategies for communication in GALS systems. (a) Asynchronous handshake which requires couple of clock cycles for each transaction and (b) source synchronous flow control which can sustain one transaction per clock cycle.

is not full. The buffer in the destination should support reads and writes in different clock domains since its writing is controlled by the source processor while its reading is controlled by the destination processor, a good solution is using a dual-clock FIFO [19].

B. Source Synchronous Flow Control for Long Distance Communication

The design becomes more complex when considering long-distance communication. For the asynchronous handshake scheme, each bit of the signal might reach the destination in a very different time, and some specific logic is necessary to guarantee the arrival of all bits of the signal [25]. The logic overhead to handle this asynchronous communication is not negligible, and the propagation delay can be significant.

For the source synchronous scheme in long distance communication, in order to avoid different data bits reaching the destination in different clock periods, the circuits must meet several requirements; they must: prevent signal delays larger than the clock period, minimize crosstalk effects from long distance wires, and data signals will likely need to be registered (pipelined) in intermediate processors along its path to maintain high clock rates. Two options exist for this task, as shown in Fig. 15(a) and (b), where processor A sends data to processor C through processor B. The first option is to register the signals using intermediate processors' clocks and use a dual-clock FIFO, as shown in Fig. 15(a). This scheme should work well under most situations, but is not efficient due to significant additional circuits and increased path latency. In addition, if an intermediate processor's clock is running slowly, it will be a bottleneck to the link's throughput.

We propose to expand the source synchronous method that routes the initial source clock along the *entire* path, as shown in Fig. 15(b). In the case when the source processor is running at slow speed, the registers in the intermediate processors are unnecessary and the link latency can be reduced with non-pipelined paths as shown in Fig. 15(c). Besides the data and the clock, wires carrying information indicating the destination FIFO's full and empty status are also required to be transferred to guarantee correct FIFO operation.

C. Clock Propagation Delay Requires Special Consideration

The dual-clock FIFO design allows arbitrary clock skew and drift, which greatly simplifies the circuit design. The most important parameter of the clock used for the synchronization is no longer the skew or jitter, but is propagation delay.

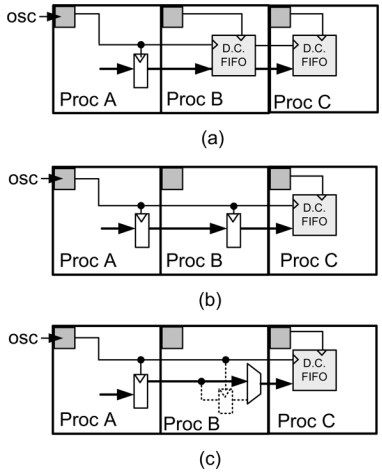


Fig. 15. Synchronization strategies for long distance communication between processors with unrelated clocks: (a) using intermediate processors' clocks with dual-clock FIFOs; (b) using source synchronous clocking with pipeline stages in each intermediate processors; and (c) using source synchronous clocking with selectable registering in intermediate processors.

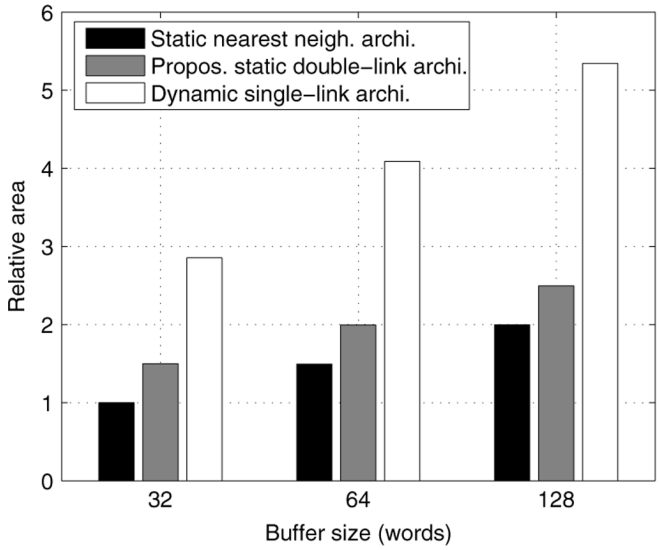


Fig. 16. Relative communication circuit area of several interconnection architectures. The static single-link nearest-neighbor-only and proposed static double-link asymmetric architecture contain two ports (buffers) for the processing core, and the dynamic routing architecture contains four buffers for the router's input ports with a single-link at each edge.

First, the delay of the clock wire must match the delay of the data wires to meet the setup and hold time requirements during register writes. Adding configurable delay gates in the data path [26] and the clock path is an excellent solution to this challenge. Second, the delay of clock (and data) should be minimized if possible, since increasing the communication delay not only increases the application computation latency, but can also reduce the application computation throughput.

V. Results Comparison

Different communication architectures, including the static nearest neighbor interconnect, the proposed double-link routing

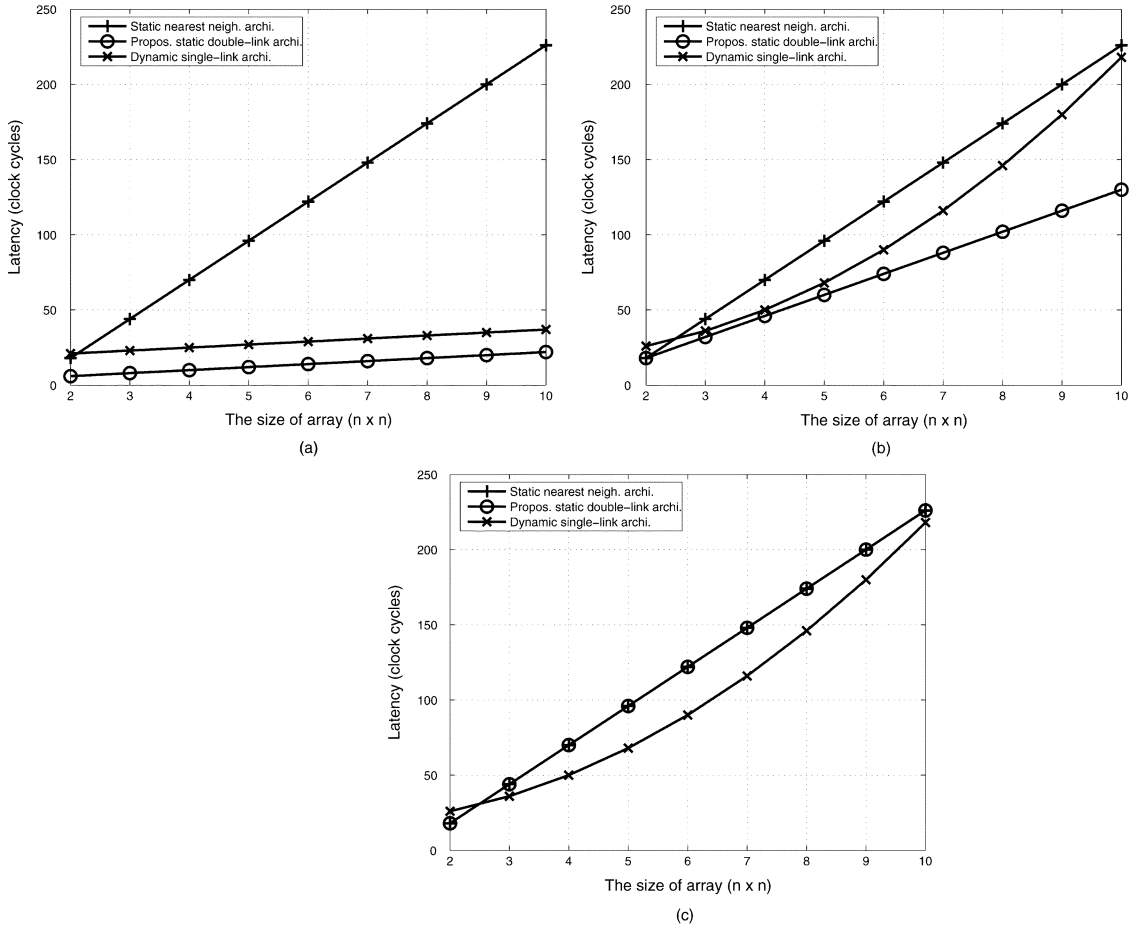


Fig. 17. Comparing the communication latency of the static nearest neighbor architecture, dynamic single-link routing architecture, and the proposed static double-link architecture, by varing the size of the array, and using basic communication patterns: (a) one-to-one and one-to-all; (b) all-to-one; (c) all-to-all.

architecture, and the traditional dynamic routing architecture, are compared in this section. The proposed communication circuitry uses the topology shown in Fig. 4, the static routing with two buffers for the processing core as shown in Fig. 5(b), and double links at each edge as shown in Fig. 8(d), and the extended source synchronous strategy shown in Fig. 15(c).

Besides the difference of the high level architecture, some details of the implementation also impact the exact results. The choice of the buffer sizes can affect system performance significantly [27], [28]. In this section, we assume FIFOs are large enough so that their effect on performance is minimized. In this design, the output from the processor core can be sent to any combination of four directions and can be reconfigured dynamically. The implemented example is compared to a traditional dynamically configurable interconnect architecture with symmetric buffer allocation and a single-link between each neighboring processor pair; we use 20 clock cycles as the communication latency [17], and we also assume the destination processor receives one data and its address separately in two clock cycles (sending the address together with the data could certainly be done with additional hardware but the relative results would be similar).

A. Area

Fig. 16 compares the communication circuitry area of three architectures (including buffers and control logic) with different sizes for each buffer. All areas are normalized to the area of the static nearest neighbor architecture. When the buffer is not large, the control logic plays an important role in determining area. For example, when the buffer size is 32 words, the static double-link architecture is about 1.5 times larger than the static nearest neighbor architecture. Along with the increased buffer size, the area of the communication circuitry will be dominated by the buffer size. When the buffer is 128 words, the proposed double-link architecture has approximately 25% larger area compared to the nearest neighbor architecture, while the traditional dynamic routing architecture is more than two times larger.

B. Performance Comparison

In this section, we analyze the performance of different implementations.

1) Performance of the Basic Communication Patterns: Fig. 17 shows the latency of the basic communication patterns mapped onto different architectures along with different array sizes. The one-to-one communication has the same latency as the one-to-all broadcast.

The proposed double-link routing architecture normally has significant savings in communication latency compared to the nearest neighbor architecture. The latency of the dynamic single-link routing architecture is similar to the static double-link architecture; it is a little worse in the one-to-one communication and all-to-one patterns; it is a little better in the

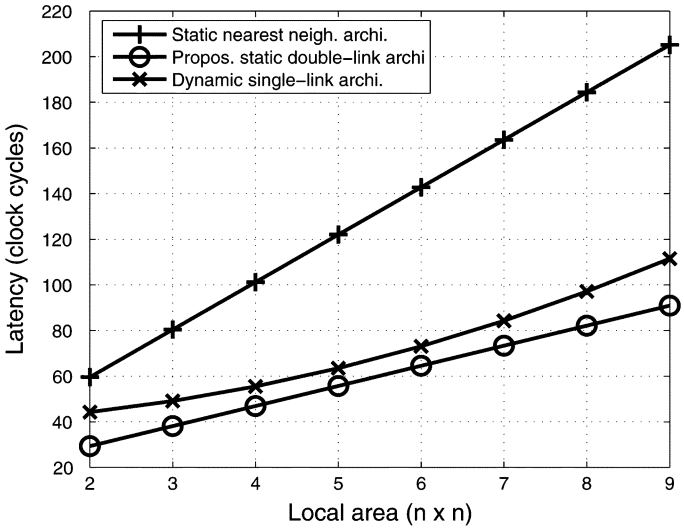


Fig. 18. Comparing the communication latency of the static nearest neighbor architecture, dynamic single-link routing architecture, and the proposed static double-link architecture by mapping various application models to a 10×10 array. The application models are obtained by uniformly combining four basic communication patterns and assuming 80% of the communication is within the local area, and the meaning of *local area* varies from 2×2 to 9×9 array.

all-to-all communication since this is where the flexibility of dynamic routing greatly helps performance.

2) Combining the Basic Patterns: Although communication patterns of real applications vary over a wide range of patterns, they can often be modeled using some combination of basic communication patterns. In this section we investigate one application model mapped onto a 10×10 array.

The modeled communication is organized uniformly by the four basic communication patterns and we assume 80% of the communication is within the local area which is the value often used in the literature [29]. In order to generally cover the meaning of localization for different communication patterns, eight data points present the situations where the definition of *local* area varies from 2×2 to 9×9 arrays. Fig. 18 shows how latency is impacted.

Not surprisingly, the static nearest neighbor architecture has the longest latencies, the proposed static double-link routing architecture is more than 2 times faster, and the dynamic single-link routing architecture is a little slower than the static double-link architecture.

VI. Conclusion

An asymmetric inter-processor communication architecture which assigns more buffer resources to the nearest neighbor interconnect and fewer buffer resources to the long distance interconnect is proposed. Static routing is emphasized due to its low cost and low communication latency. Inserting two ports (buffers) for the processing core and using two or three links at each edge can achieve good area/performance trade offs for chip multiprocessors consisting of simple single-issue processors, and the optimal number of links is expected to increase if a chip multiprocessor is built with larger processors. Compared to a traditional dynamically-configurable interconnect architecture with symmetric buffer allocation and single-links between neighboring processor pairs, this implementation has approximately two times smaller communication circuitry area with a similar routing capability. The proposed architecture also provides the ability to support long distance GALS communication with an extended source synchronous transfer method.

A varied group of seven designs with one, two, three, and four links per processor edge, and with the most promising interconnect strategies were designed and laid out in 0.18-μm CMOS and analyzed for their chip area, maximum clock rate, and communication latency over array sizes up to 100 processors for various communication patterns. Compared to the single-link processor on average, two-link designs require 5% more area, three-link designs require 12% more area, and four-link designs require 24% more area. In particular, the Type 3 two-link design requires 7% more area than the single-link design but performs all-to-one communication almost two times faster.

Acknowledgment

The authors would like to thank members of the VCL, Y.-P. Cheng, R. Krishnamurthy, M. Anders, S. Mathew, K. Bowman, A. Gatherer, and P. Kudva.

References

[1] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, W. Lee, A. Saraf, N. Shnidman, V. Strumpen, S. Amarasinghe, and A. Agarwal, "A 16-issue multiple-programcounter microprocessor with point-to-point scalar operand network," in *Proc. ISSCC*, Feb. 2003, pp. 170–171.

[2] S. W. Keckler, D. Burger, C. R. Moore, R. Nagarajan, K. Sankaralingam, V. Agarwal, M. S. Hrishikesh, N. Ranganathan, and P. Shivakumar, " A wire-delay scalable microprocessor architecture for high performance systems," in *Proc. ISSCC*, 2003, vol. 46, pp. 168–169.

[3] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. IEEE Int. Conf. Des. Autom.*, Jun. 2001, pp. 684–689.

[4] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28 TFLOPS networkon-chip in 65 nm CMOS," in *Proc. ISSCC*, Feb. 2007, pp. 98–99.

[5] B. Baas, Z. Yu, M. Meeuwsen, O. Sattari, R. Apperson, E. Work, J. Webb, M. Lai, T. Mohsenin, D. Truong, and J. Cheung, "AsAP: A fine-grain multi-core platform for DSP applications," *IEEE Micro*, vol. 27, no. 2, pp. 34–45, Mar./Apr. 2007.

[6] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, D. Truong, T. Mohsenin, and B. Baas, "AsAP: An asynchronous array of simple processors," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 695–705, Mar. 2008.

[7] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep./Oct. 2007.

[8] H. Zhang, M. Wan, V. George, and J. Rabaey, "Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs," in *Proc. IEEE Comput. Soc. Workshop VLSI*, Apr. 1999, pp. 2–8.

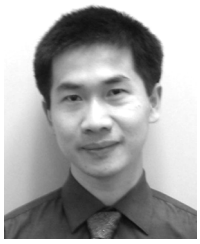
[9] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Effect of traffic localization on energy dissipation in NoC-based interconnect," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2005, pp. 1774–1777.

[10] Z. Yu and B. M. Baas, "Low-area interconnect architecture for chip multiprocessors," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2008, pp. 2857–2860.

[11] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, Oct. 1984.

[12] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-w TeraFLOPS processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

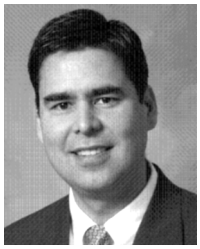
[13] W. J. Dally, "Virtual-channel flow control," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, no. 2, pp. 194–205, Mar. 1992.
[14] J. Hu, U. Y. Ogras, and R. Marculescu, "System-level buffer allocation for application-specific network-on-chip router design," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2919–2933, Dec. 2006.
[15] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Trans. Commun.*, vol. 35, no. 12, pp. 1347–1356, Dec. 1987.
[16] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, and B. Baas, "An asynchronous array of simple processors for DSP applications," in *Proc. ISSCC*, Feb. 2006, pp. 428–429.
[17] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar operand networks: On-chip interconnect for ILP in partitioned architecture," in *Proc. IEEE Int. Symp. High-Perform. Comput. Arch.*, Feb. 2003, pp. 341–353.
[18] H. T. Kung, "Why systolic architectures?," *Comput. Mag.*, vol. 15, no. 1, pp. 37–45, Jan. 1982.
[19] R. Apperson, Z. Yu, M. Meeuwsen, T. Mohsenin, and B. Baas, "A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 10, pp. 1125–1134, Oct. 2007.
[20] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, A. Tran, J. Webb, E. Work, Z. Xiao, and B. Baas, "A 167-processor 65 nm computational platform with perprocessor dynamic supply voltage and dynamic clock frequency scaling," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2008, pp. 22–23.
[21] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, A. Tran, Z. Xiao, E. Work, J. Webb, P. Mejia, and B. Baas, "A 167-processor 65 nm computational platform in 65 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 4, pp. 1130–1144, Apr. 2009.
[22] D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, Y. Durand, J. Durupt, D. Varreau, P. Vivet, P. Penard, A. Bouttier, and F. Berens, "A telecom baseband circuit based on an asynchronous network-on-chip," in *Proc. ISSCC*, Feb. 2007, pp. 258–259.
[23] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey, "A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing," *J. Solid-State Circuits*, vol. 35, no. 11, pp. 1697–1704, Nov. 2000.
[24] E. Beigne and P. Vivet, "Design of on-chip and off-chip interfaces for a GALS Noc architecture," in *Proc. Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, Mar. 2006, pp. 13–15.
[25] G. Campobello, M. Castano, C. Ciofi, and D. Mangano, "GALS networks on chip: A new solution for asynchronous delay-insensitive links," in *Proc. Eur. Event for Electron. Syst. Des. Test*, Apr. 2006, pp. 160–165.
[26] Z. Yu and B. Baas, "Implementing tile-based chip multiprocessors with GALS clocking styles," in *Proc. ICCD*, Oct. 2006, pp. 174–179.
[27] Z. Yu and B. Baas, "Performance and power analysis of glabally asynchronous locally synchronous multi-processor systems," in *Proc. IEEE Comput. Soc. Ann. Symp. VLSI (ISVLSI)*, Mar. 2006, pp. 378–384.
[28] Z. Yu and B. Baas, "High performance, energy efficiency, and scalability with GALS chip multiprocessors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 1, pp. 66–79, Jan. 2009.
[29] K. Lee, S. Lee, and H. Yoo, "Low-power network-on-chip for highperformance SoC design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, pp. 148–160, Feb. 2006.



Zhiyi Yu received the B.S. and M.S. degrees in electrical engineering from Fudan University, Shanghai, China, in 2000 and 2003, respectively, and the Ph.D. degree in electrical and computer engineering from the University of California, Davis, in 2007.

He is currently an Associate Professor with the State Key Laboratory of ASIC & System, Fudan University, Shanghai, China. From 2007 to 2008, he was with IntellaSys Corporation, Cupertino, CA, where he participated in the design of the many-core SEAForth chips which utilize stack-based processors with extremely small area and low power consmption. When in UC Davis, he was a key designer of the 36-core Asynchronous Array of simple Processors (AsAP) chip, and one of the designers of the 167-core second generation computational array chip. His research interests include high-performance and energy-efficient digital VLSI design with an emphasis on many-core processors.

Dr. Yu is on the Technical Program sub-Committee of the IEEE Asian Solid-State Circuit Conference (A-SSCC) in 2009, and on the Technical Program Committee of the IEEE International Conference on ASIC (ASICON) in 2009.



Bevan M. Baas received the B.S. degree in electronic engineering from California Polytechnic State University, San Luis Obispo, in 1987, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1990 and 1999, respectively.

From 1987 to 1989, he was with Hewlett-Packard, Cupertino, CA, where he participated in the development of the processor for a high-end minicomputer. In 1999, he joined Atheros Communications, Santa Clara, CA, as an early employee and served as a core member of the team which developed the first IEEE 802.11a (54 Mbps, 5 GHz) Wi-Fi wireless LAN solution. In 2003, he joined the Department of Electrical and Computer Engineering, University of California, Davis, where he is now an Associate Professor. He leads projects in architecture, hardware, software tools, and applications for VLSI computation with an emphasis on DSP workloads. Recent projects include the 36-processor Asynchronous Array of simple P rocessors (AsAP) chip, applications, and tools; a second generation 167-processor chip; low density parity check (LDPC) decoders; FF T processors; viterbi decoders; and H.264 video codecs. During the summer of 2006, he was a Visiting Professor in Intel's Circuit Research Lab.

Dr. Baas was a National Science Foundation Fellow from 1990 to 1993 and a NASA Graduate Student Researcher Fellow from 1993 to 1996. He was a recipient of the National Science Foundation CAREER Award in 2006 and the Most Promising Engineer/Scientist Award by AISES in 2006. He is an Associate Editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS and has served as a member of the Technical Program Committee of the IEEE International Conference on Computer Design (ICCD) in 2004, 2005, 2007, and 2008, and on the Program Committee of the HotChips Symposium on High Performance Chips in 2009. He also serves as a member of the Technical Advisory Board of an early stage technology company.