Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Systems Architecture 50 (2004) 81–104
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

Packetization and routing analysis of on-chip multiprocessor networks

Terry Tao Ye [a,*], Luca Benini [b], Giovanni De Micheli [c]

[a] Computer Systems Lab, Stanford University, Gates 334, Stanford, CA 94305, USA
[b] DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
[c] Computer Systems Lab, Stanford University, Gates 333, Stanford, CA 94305, USA

Abstract

Some current and most future systems-on-chips use and will use network architectures/protocols to implement on-chip communication. On-chip networks borrow features and design methods from those used in parallel computing clusters and computer system area networks. They differ from traditional networks because of larger on-chip wiring resources and flexibility, as well as constraints on area and energy consumption (in addition to performance requirements). In this paper, we analyze different routing schemes for packetized on-chip communication on a mesh network architecture, with particular emphasis on specific benefits and limitations of silicon VLSI implementations. A contention-look-ahead on-chip routing scheme is proposed. It reduces the network delay with significantly smaller buffer requirement. We further show that in the on-chip multiprocessor systems, both the instruction execution inside node processors, as well as data transaction between different processing elements, are greatly affected by the packetized dataflows that are transported on the on-chip networks. Different packetization schemes affect the performance and power consumption of multiprocessor systems. Our analysis is also quantified by the network/multiprocessor co-simulation benchmark results.

Keywords: Networks-on-chip; On-chip multiprocessors; On-chip communication; On-chip networks

1. Introduction

Driven by the advances of semiconductor technology, future *shared-memory multiprocessor systems-on-chip* (MPSoC) [1] will employ billions of transistors and integrate hundreds, or even more, of processing elements (PEs) on a single chip. The design of reliable, low-energy and high-performance interconnection of such elements will pose unprecedented problems. Traditional on-chip communication structures (i.e., the buses) used in many of today's SoC designs are limited by the throughput and energy consumption. Ad hoc wire routing in ASICs is facing more and more challenging interconnect issues (i.e., wire delay, signal

* Corresponding author. Fax: +1-650-725-9802.
E-mail addresses: taoye@stanford.edu (T.T. Ye), lbenini@deis.unibo.it (L. Benini), nanni@stanford.edu (G.D. Micheli).

1383-7621/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2003.07.005

integrity, etc.) in deep submicron domain [2]. Next generation MPSoC designs need a novel on-chip communication architecture that can provide scalable and reliable point-to-point data transportation [3–5].

As a new SoC design paradigm, on-chip network architectures, or *networks-on-chip* (NoC), will support novel solutions to many of today's SoC interconnect problems. For example, multiple dataflows can be supported concurrently by the same communication resources, data integrity can be enhanced by error correction and data restoration, and components are more modularized for IP reuse.

On-chip multiprocessor network architectures may adopt concepts and design methodologies from computer networks, namely from *system area networks* (SAN) and *parallel computer clusters* (PCP) (In particular, MPSoC networks have many performance requirements similar to those in parallel computer networks). Like computer networks, on-chip networks may take advantage of data packetization to ensure fairness of communication [6,7]. For example, in on-chip networks, packets may contain headers and payloads, as well as error correction and priority setting information. This strongly contrasts with the ad hoc physical wire routing used in (non-networked) ASICs. Packet routes can either be setup before transmission, and kept unchanged through the entire message transmission, or each packet may be free to choose an appropriate route to destination. In some cases, *contention* may pose problems, and proper arbitration schemes can be used to manage and optimize the network utilization.

Nevertheless, silicon implementation of networks requires a different perspective. On-chip network architectures and protocols have to deal with the advantages and limitations of the silicon fabric. Chip-level communication is localized between processing elements. On-chip networks do not need to follow the standard schemes for communication since they can use lighter and faster protocol layers. MPSoC processing elements exchange data through the on-chip interconnect wires that can handle separately, if desired, data and control signals. We believe these different aspects will require new methodologies for both the on-chip switch designs as well as the routing algorithm designs. In particular, we propose that the following directions in the MPSoC networks-on-chip design should be explored:

- *Data packetization*—On-chip multiprocessor systems consist of multiple processing elements (PEs) and storage arrays (SAs). We refer to the PEs and SAs as *nodes*. These nodes are interconnected by a dedicated on-chip network. The dataflow traffic on the network comes from the internode transactions. Therefore, the performance and power consumption of on-chip communication are not only determined by the physical aspects of the network (e.g., voltage swing, the wire delay and fan-out load capacitance, etc.), but also depend on the packetized data flows on the network. In particular, the interactions between different packets and processor nodes will greatly affect the system performance as well as power consumption.
- *Network architecture*—The on-chip network architecture should utilize the abundant wiring resources available on silicon. Control signals need not be serialized and transmitted along with data, as in the case of many computer cluster networks, but can run on dedicated control wires (Fig. 1). The usage of buffers should be limited to reduce the area and energy consumption.
- *Routing algorithm*—On-chip routing should use those algorithms that do not require substantial on-chip buffer usage. At the same time, the network state (including contention information) can be made available through dedicated control wires. From this perspective, it is possible

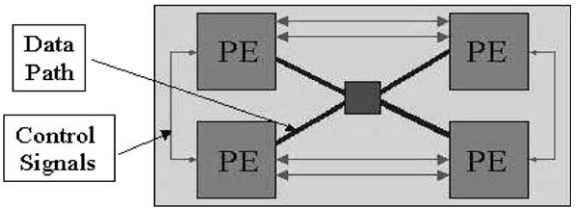


Fig. 1. Dedicated control wires and data paths for on-chip network.

to achieve *contention-look-ahead* to reduce contention occurrence and increase bandwidth.

The paper is organized as follows: Section 2 will first briefly describe the MPSoC and its on-chip network architectures. Because the on-chip dataflow greatly affects the system level performance and power consumption, we will analyze the source, packetization and routing issues of the on-chip data traffic in Section 3. Based on this analysis, in Section 4, we will propose our contention-look-ahead routing technique along with the on-chip switch architecture design. We describe the experimental platform in Section 5 and use this platform to compare the proposed contention-look-ahead routing technique with other routing schemes in Section 6. Packetized dataflows play a critical role in MPSoCs. In Sections 7 and 8, we will further perform quantitative analysis on the packet size impact on MPSoC performance as well as energy consumption. As a general design guideline, overall trends are also summarized qualitatively in Section 9.

2. MPSoC architecture and networks

The internode communication between multiprocessors can be implemented by either *message passing* or *memory sharing*. In the message passing MPSoCs, data transactions between nodes are performed explicitly by the communication APIs, such as *send*( ) or *receive*( ). These API commands require special protocols to handle the transaction, and thus create extra communication overhead. In comparison, in shared memory (in some case, shared level 2 cache) MPSoCs, data transactions can be performed implicitly through memory access operation [10]. Therefore, shared-memory MPSoC architectures have been widely used in many of today's high performance embedded systems. We will briefly describe some examples as follows.

2.1. Shared memory MPSoC examples

Daytona [6] is a single chip multiprocessor developed by Lucent. It consists of four 64-bit processing elements that generate transactions of different sizes. Daytona targets on the high performance DSP applications with scalable implementation choices. The internode communication is performed by the on-chip bus with split transactions. Piranha [8] project is developed by DEC/Compaq, it integrates eight alpha processors on a single chip and uses packet routing for the on-chip communication. The Stanford Hydra [9] chip contains four MIPS-based processors and uses shared level-2 cache for internode communication.

All these architectures utilize shared memory (or cache) approach to perform data transactions between processors, and thus achieve high performance with parallel data processing ability. In this paper, we will use the shared memory MPSoC platform to analyze different aspects of on-chip network architectures.

2.2. MPSoC architecture

A typical shared-memory on-chip multiprocessor system is shown in Fig. 2. It consists of several node processors or processing elements connected by an on-chip interconnect network. Each node processor has its own CPU/FPU and cache hierarchy (one or two levels of cache). A read miss in L1 cache will create an L2 cache access, and a miss in L2 cache will then need a memory access. Both L1 and L2 cache may use write-through or write-back for cache updates.

The shared memories are associated with each node, but they can be physically placed into one big memory block. The memories are globally addressed and accessible by the memory directory. When there is a miss in L2 cache, the L2 cache will send a request packet across the network asking for memory access. The memory with the requested address will return a reply packet containing the data to the requesting node. When there is a cache write-through or write-back operation, the cache block that needs to be updated is encapsulated in a packet and sent to the destination node where the corresponding memory resides.

Cache coherence is a very critical issue in shared-memory MPSoC. Because one data may have several copies in different node caches, when

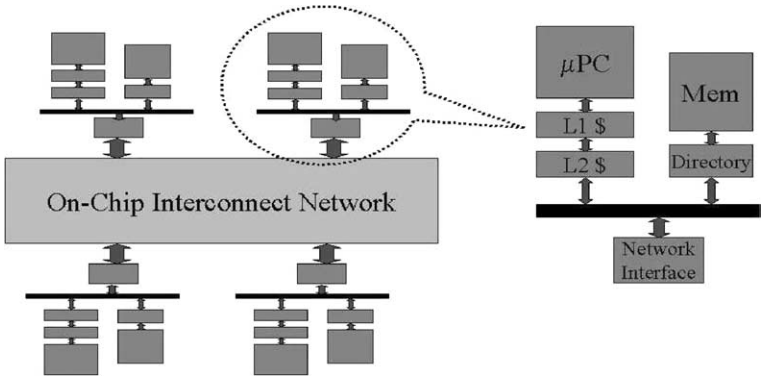


Fig. 2. MPSoC architecture.

the data in memory is updated, the stale data stored in each cache needs to be updated. There are two methods of solving the cache coherence problem: (1) *cache update* updates all copies in cache when the data in memory is updated; (2) *cache invalidate* invalidates all copies in cache. When the data is read the next time, the read will become a miss and consequently need to fetch the updated data from the corresponding memory.

2.3. On-chip networks

Because of different performance requirements and cost metrics, many different multiprocessor network topologies are designed for specific applications. MPSoC networks can be categorized as *direct networks* and *indirect networks* [11]. In direct network MPSoCs, node processors are connected directly with each other by the network. Each node performs dataflow routing as well as arbitration. In indirect network MPSoCs, node processors are connected by one (or more) intermediate node switches. The switching nodes perform the routing and arbitration functions. Therefore, indirect networks are also often referred to as *multistage interconnect networks* (MIN). Although some direct networks and indirect networks may be equivalent in functionality, e.g., if each node processor has one dedicated node switch, this node switch can either be embedded inside the node processor, or be constructed outside. Nevertheless, direct and indirect topologies have different impact on network physical implementation, we still distinguish them as two categories in this paper.

Many different on-chip network architectures have been proposed for different MPSoC designs. Examples include, but are not limited to, the following:

(1) The *mesh* and *torus* networks, or orthogonal networks, are connected in k-ary n-dimensional mesh (k-ary n-mesh) or k-ary n-dimensional torus (k-ary n-cube) formations. Because of the simple connection and easy routing scheme provided by adjacency, orthogonal networks are widely used in parallel computing platforms. Mesh and torus topologies can be implemented both as direct networks and indirect networks. A direct torus/mesh architecture was analyzed by [4] for the feasibility of on-chip implementation. In the architecture proposed in [4], each PE is placed as a tile and connected by the network (either a mesh or torus topology) (Fig. 3a). Each tile can perform packet routing and arbitration independently. The network interfaces are located on the four peripherals of each tile node.
(2) The *Nostrum* network is a two-dimensional indirect mesh network [13] (Fig. 3b). In this proposed on-chip implementation, a dedicated switch network is used to perform the routing function and acts as a network interface for each node.

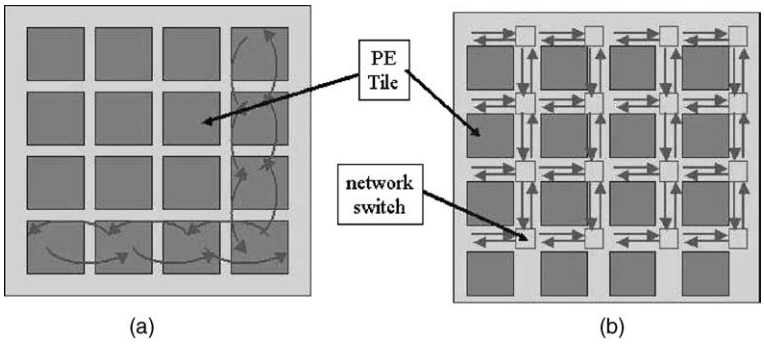


Fig. 3. The two-dimensional mesh networks. (a) Proposed by Dally et al. and (b) proposed by Kumar et al.

(3) The *Eclipse* (embedded chip level integrated parallel super computer) system [14] uses a sparse 2D mesh network, in which the number of switches is at least the square of the number of processing resources divided by four. Eclipse uses cacheless shared-memories, so it has no cache coherence problems, and communication will not jam the network even in the case of heavy random access traffic.
(4) The *SPIN* (scalable, programmable, integrated network) is an indirect network [15] (Fig. 4a). The network uses the *fat-tree* topology to connect each node processor. Compared with the two-dimensional mesh, in fat-tree networks, the point-to-point delays are bounded by the depth of the tree, i.e., in the topology in Fig. 4a, communication between any processor nodes requires at most three switch stages when an appropriate routing algorithm is applied.
(5) The *Octagon* network was proposed by [16] (Fig. 4b) in the context of network processor design. It is a direct network. Similar to that in the fat-tree topology, the point-to-point delay is also determined by the relative source/terminus locations, and communication between any two nodes (within an octagon subnetwork) requires at most two hops (intermediate links).

2.4. On-chip network characteristics

On-chip networks are fabricated on a single chip and benefit from data locality. In comparison, networked computers are physically distributed at different locations. Although many of the above on-chip network architectures adopt the topology from computer networks, e.g., system area networks and parallel computer clusters. Many assumptions in computer networks may no longer hold for on-chip networks.

(1) Wiring resources

In computer networks, computers are connected by cables. The number of wires encapsulated in a cable is limited (e.g., CAT-5 Ethernet

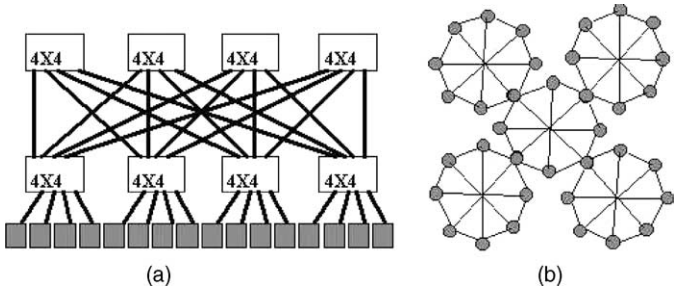


Fig. 4. (a) SPIN networks and (b) octagon networks.

cable has 8 wires, parallel cable in PC peripheral devices has 25 wires, etc.). Binding more wires in a cable is not physically and practically viable. Because of the wiring limitation, in many of today's computer networks, data are serialized in fixed quanta before transmission.

In comparison, the wire connection between components in SoC is only limited by the switching and routing resources. In today's 0.13 μm semiconductor process, the metal wire pitch varies from 0.30 μm to 0.50 μm, while 8 metal layers are available. Thus a 100 μm × 100 μm switch-box can accommodate hundreds of wires in any direction (i.e., layers). The cost of adding more routing layers continues to decrease as the VLSI process technology advances. Therefore, physical wire density is not the limiting factor for future SoC designs.

(2) Buffers on networks

Limited wiring resources tends to create contention and limit throughput. Computer networks use heavily buffers to compensate for wire limitation. Buffers provide temporary storage when contention occurs, or when the dataflow exceeds the network throughput. Network switches and routers use a fairly large amount of buffer spaces. These buffers are implemented with SRAMs and DRAMs. The buffer size can be as big as several hundred megabytes (e.g., in the case of network routers).

In comparison, on-chip networks should always balance the buffer usage with other architectural options, because on-chip buffers are implemented by SRAMs or DRAMs, both consume significant power during operation. Besides, on-chip SRAMs occupy a large silicon area, and embedded DRAMs increase the wafer manufacturing cost. Since buffers are expensive to implement and power-hungry during operation, on-chip networks should reduce the buffer size on the switches as much as possible.

Wiring resources and buffer usage are two important factors in designing MPSoC communication networks. Although the network performance and power consumption are also dependent upon many other factors, in the following sections we will explore the on-chip routing schemes that can best utilize the on-chip wires while minimizing the buffer usage.

3. On-chip network traffic

Before we start to discuss the characteristics of MPSoC interconnect networks, we need to study the traffic on the network. In particular, we should analyze the composition of the packetized dataflows that are exchanged between MPSoC nodes.

Packets transported on NoCs consist of three parts. The header contains the destination address, the source address, and the requested operation type (READ, WRITE, INVALIDATE, etc). The payload contains the transported data. The tail contains the error checking or correction code.

3.1. Sources of packets

Packets traveling on the network come from different sources, and they can be categorized into the following types:

(1) *Memory access request packet.* The packet is induced by an L2 cache miss that requests data fetch from memories. The header of these packets contains the destination address of the target memory (node ID and memory address) as well as the type of memory operation requested (memory READ, for example). The address of the L2 cache is in the header as well, as it is needed to construct the data fetch packet (in case of a READ). Since there is no data being transported, the payload is empty.

(2) *Cache coherence synchronization packet.* The packet is induced by the cache coherence operation from the memory. This type of packet comes from the updated memory, and it is sent to all caches, each cache will then update its content if it contains a copy of the data. The packet header contains the memory tag and block address of the data. If the synchronization uses the "update" method, the packet contains updated data as payload. If the synchronization uses the "invalidate" method, the packet header contains the operation type (INVALIDATE, in this case), and the payload is empty.

(3) *Data fetch packet.* This is the reply packet from memory, containing the requested data. The packet header contains the target address (the node ID of the cache requesting for the data). The data is contained in the packet payload.

(4) *Data update packet*. This packet contains the data that will be written back to the memory. It comes from L2 cache that requests the memory write operation. The header of the packet contains the destination memory address, and the payload contains the data.

(5) *IO and interrupt packet.* This packet is used by IO operations or interrupt operations. The header contains the destination address or node ID. If data exchange is involved, the payload contains the data.

3.2. Data segmentation and packet size

From the analysis in Section 3.1, we can see most packets travel between memories and caches, except those packets involved in I/O and interrupt operations. Although packets of different types originate from different sources, the length of the packets is determined by the size of the payload. In reality, there are two differently sized packets on the MPSoC network, *short packet* and *long packet*, as described below.

Short packets are the packets with no payloads, such as the memory access request packets and cache coherence packets (invalidate approach). These packets consist only header and tail. The request and control information can be encoded in the header section.

Long packets are the packets with payloads, such as the data fetch packets, the data update packets and the cache coherence packets used in the update approach. These packets travel between caches and memories. The data contained in the payload are either from cache block, or they are sent back to the node cache to update the cache block. Normally, the payload size equals the cache block size, as shown in Fig. 5.

Packets with payload size different than the cache block size will increase cache miss penalty. The reasons are two. (1) If each cache block is segmented into different packets, it is not guaranteed that all packets will arrive at the same time, and consequently the cache block cannot be updated at the same time. Especially when the cache block size is big enough (as in the case of the analysis in the following sections), it will take longer time to finish a cache update operation. (2)

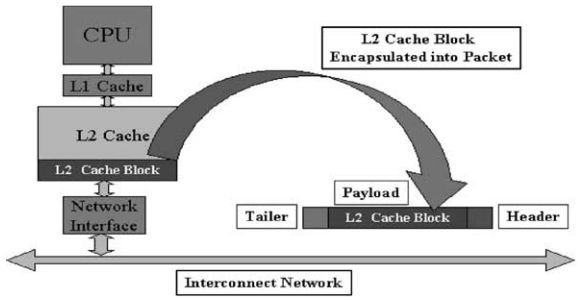


Fig. 5. Packet size and cache block Size.

If several cache blocks are to be packed into one packet payload, the packet needs to hold its transmission until all the cache blocks are updated. This will again increase the cache miss delay penalty.

In our analysis, we assume all the long packets contain the payload of one cache block size. Therefore, the length of the long packets will determine the cache block size of each node processor.

3.3. Packet switching techniques

In computer networks, many different techniques are used to perform the packet switching between different nodes. Popular switching techniques include *store-and-forward*, *virtual cut-through* and *wormhole*. When these switching techniques are implemented in on-chip networks, they will have different performance along with different requirements on hardware resources.

3.3.1. Store-and-forward switching

In many computer networks, packets are routed in a *store-and-forward* fashion from one router to the next. Store-and-forward routing enables each router to inspect the passing packets, and therefore perform complex operations (e.g., content-aware packet routing). When the packet size is big enough, i.e., the long packets, as analyzed above, store-and-forward routing not only introduces extra packet delay at every router stage, but it also requires a substantial amount of buffer spaces because the switches may store multiple complete packets at the same time.

In on-chip networks, storage resources are very expensive in terms of area and energy consumption. Moreover, the point-to-point transmission delay is very critical. Therefore, store-and-forward approaches are dis-advantageous for on-chip communication.

3.3.2. Virtual cut through switching

Virtual cut through (VCT) switching is proposed to reduce the packet delays at each routing stage. In VCT switching, one packet can be forwarded to the next stage before its entirety is received by the current switch. Therefore, VCT switching reduces the store-and-forward delays. However, when the next stage switch is not available, the entire packet still needs to be stored in the buffers of the current switch.

3.3.3. Wormhole switching

Wormhole routing was originally designed for parallel computer clusters [11] because it achieves the minimal network delay and requires less buffer usage. In wormhole routing, each packet is further segmented into *flits* (flow control unit). The header flit reserves the routing channel of each switch, the body flits will then follow the reserved channel, the tail flit will later release the channel reservation.

One major advantage of wormhole routing is that it does not require the complete packet to be stored in the switch while waiting for the header flit to route to the next stages. Wormhole routing not only reduces the store-and-forward delay at each switch, but it also requires much less buffer spaces. One packet may occupy several intermediate switches at the same time. Because of these advantages, wormhole routing is an ideal candidate switching technique for on-chip multiprocessor interconnect networks.

3.4. Wormhole routing issues

Since wormhole switching has many unique advantages for on-chip network implementation, we will discuss the *deadlock* and *livelock* issues in this context, although these issues exist in other routing schemes as well.

3.4.1. Deadlock

In wormhole routing, one packet may occupy several intermediate switches at the same time. Packets may block each other in a circular fashion such that no packets can advance, thus creating a deadlock.

To solve the deadlock problem, the routing algorithms have to break the circular dependencies among the packets. *Dimension-ordered* routing [11,12], with the constraints of *turn rules*, is one way to solve the deadlock: the packets always route on one dimension first, e.g., column first, upon reaching the destination row (or column), and then switch to the other dimension until reaching the destination. Dimension-ordered routing is deterministic: packets will always follow the same route for the same source-destination pair. Therefore, it cannot avoid contention. Whenever contention occurs, the packets have to wait for the channel to be free.

Another way to solve the deadlock problem is to use *virtual channels* [11,17]. In this approach, one physical channel is split into several virtual channels. Virtual channels can solve the deadlock problem while achieving high performance. Nevertheless, this scheme requires a larger buffer space for the waiting queue of each virtual channel. For example, if one channel is split into four virtual channels, it will use four times as much buffer spaces as a single channel. The architecture proposed in [4] requires about 10 K-bit of buffer space on each edge of the tile. The virtual channel arbitration also increases the complexity of circuit design.

3.4.2. Livelock

Livelock is a potential problem in many adaptive routing schemes. It happens when a packet is running forever in a circular motion around its destination. We will use the *hot potato* routing as an example to explain this issue.

Hot potato or *deflection* routing [19] is based on the idea of delivering a packet to an output channel at each cycle. It requires the assumption that each switch has an equal number of input and output channels. Therefore, input packets can always find at least one output exit. Under this routing scheme, when contention occurs and the

desired channel is not available, the packet, instead of waiting, will pick any alternative available channels to continue moving to the next switch. However, the alternate channels are not necessarily along the shortest routes.

In hot potato routing, if the switch does not serve as the network interface to a node, packets can always find a way to exit, therefore the switch does not need buffers. However, if the nodes send packets to the network through the switch, input buffers are still needed, because the packet created by the node also needs an output channel to be delivered to the network. Since there may not be enough outputs for all input packets, either the packets from one of the input or the packets from the node processor have to be buffered [11].

In hot potato routing, if the number of input channels is equal to the number of output channels at every switch node, packets can always find an exit channel and they are deadlock free. However, *livelock* is a potential problem in hot potato routing. Proper deflection rules need to be defined to avoid livelock problems. The deflected routes in hot potato routing increase the network delays. Therefore, performance of hot potato routing is not as good as other wormhole routing approaches [11]. This is also confirmed by our experiments, as shown in Section 6.

4. Contention-look-ahead routing

One big problem of aforementioned routing algorithms in Sections 3.3 and 3.4 is that the routing decision for a packet (or header flit) at a given switch ignores the status of the upcoming switches. A *contention-look-ahead* routing scheme is one where the current routing decision is helped by monitoring the adjacent switches, thus possibly avoiding blockages.

4.1. Contention awareness

In computer networks, contention information in neighboring nodes cannot be transmitted instantaneously, because internode information can only be exchanged through packets. In comparison, on-chip networks can take advantage of dedicated control wires to transmit contention information.

A contention-aware hot-potato routing scheme is proposed in [18]. It is based on a two-dimensional mesh NoCs. The switch architecture is similar to that in [13]. Each switch node also serves as network interface to a node processor (also called resource). Therefore, it has five inputs and five outputs. Each input has a buffer that can contain one packet. One input and one output are used for connecting the node processor. An internal FIFO is used to store the packets when output channels are all occupied. The routing decision at every node is based on the "stress values", which indicate the traffic loads of the neighbors. The stress value can be calculated based on the number of packets coming into the neighboring nodes at a unit time, or based on the running average of the number of packets coming to the neighbors over a period of time. The stress values are propagated between neighboring nodes. This scheme is effective in avoiding "hot spots" in the network. The routing decision steers the packets to less congested nodes.

In the next section, we will propose a wormhole-based contention-look-ahead routing algorithm that can "foresee" the contention and delays in the coming stages using a direct connection from the neighboring nodes. It is also based on a mesh network topology. The major difference from [18] is that information is handled in flits, and thus large and/or variable size packets can be handled with limited input buffers. Therefore, our scheme combines the advantages of wormhole switching and hot potato routing.

4.2. Contention-look-ahead routing

Fig. 6 illustrates how contention information benefits the routing decision. When the header flit of a packet arrives at a node, the traffic condition of the neighboring nodes can be acquired through the control signal wires. The traffic signal can be either a one-bit wire, indicating whether the corresponding switch is busy or free, or multiple-bit signal, indicating the buffer level (queue length) of the input waiting queue. Based on this information, the packet can choose the route to the next

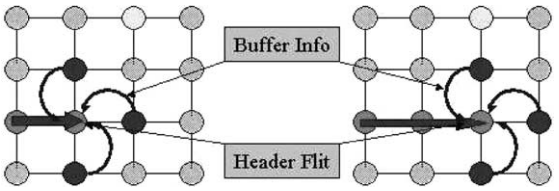


Fig. 6. Adaptive routing for on-chip networks.

available (or shortest queue) switch. The local routing decision is performed at every switch once the header flit arrives. It is stored to allow the remaining flits to follow the same path until the tail flit releases the switch.

There are many alternate routes to the neighboring nodes at every intermediate stage. We call the route that always leads the packet closer to the destination a *profitable route*. Conversely, a route that leads the packet away from the destination is called a *misroute* [11] (Fig. 7). In mesh networks, profitable routes and misroutes can be distinguished by comparing the current node ID with the destination node ID. In order to reduce the calculation overhead, the profitable route and misroute choices for every destination are stored in a look-up table, and the table is hard-coded once the network topology is set up.

Profitable routes will guarantee the shortest path from source to destination. Nevertheless, misroutes do not necessarily need to be avoided. Occasionally, the buffer queues in all available profitable routes are full, or the queues are too long. Thus, detouring to a misroute may lead to a shorter delay time. Under these circumstances, a misroute may be more desirable.

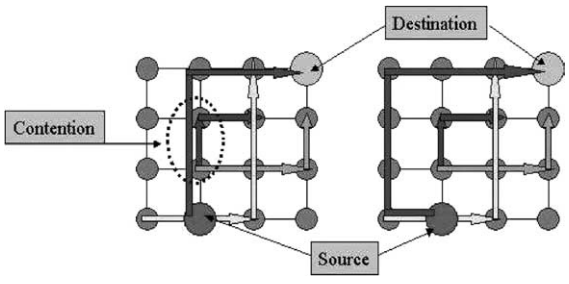


Fig. 7. Profitable route and misroute.

4.3. Wormhole contention-look-ahead algorithm

For any packet entering an intermediate switch along a path, there are multiple output channels to exit. We call C the set of output channels. For a two-dimensional mesh, $C = \{North, South, East, West\}$. We further partition C into profitable routes P and misroutes M. We define the buffer queue length of every profitable route $p \in P$ as Q_p. Similarly, we define the buffer queue length of every misroute $m \in M$ as Q_m.

Assume the flit delay of one buffer stage is D_B, and the flit delay of one switch stage is D_S. The delay penalty to take a profitable and a misroute is defined as D_{profit} and $D_{misroute}$, respectively, in the following equation (Eq. (1)).

$$D_{profit} = \min(Q_p, \forall p \in P) \times D_B \quad (1)$$

$$D_{misroute} = \min(Q_m, \forall m \in M) \times D_B + 2D_S \quad (2)$$

In a mesh network, when a switch routes a packet to a misroute, the packet moves away from its destination by one switch stage. In the subsequent routing steps, this packet needs to get back on track and route one more stage back towards its destination. Therefore, the delay penalty for a misroute is $2 \times D_S$, plus potential extra buffering delays at the misrouted stages. In our experiment, we use $2 \times D_S$ as the misroute penalty value. This value can be adjusted to penalize (or favor) more on misroute choices. In on-chip networks, the switch delay of one routing stage consists of the gate delays inside the switch logics plus the arbitration delays. The delay D_S can be estimated beforehand, and, without loss of generality, we assume the same D_S value for all switches in the network.

If all profitable routes are available and waiting queues are free, the packet will use dimension-ordered routing decision. If the buffer queues on all of the profitable routes are full or the minimum delay penalty of all the profitable routes is larger than the minimum penalty of the misroutes, it is more desirable to take the misroute. The routing decision evaluation procedure is described in the pseudo code below:

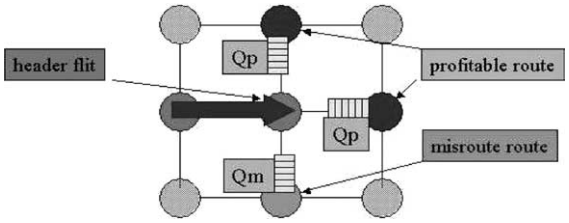


Fig. 8. Adaptive routing algorithm.

$$(D_{\text{profit}} \leqslant D_{\text{misroute}}) \quad \text{AND} \quad (Q_p \leqslant Q_{p_{\max}})? \\ \text{ProfitRoute} : \text{Misroute} \tag{3}$$

where $Q_{p_{\max}}$ is the maximum buffer queue length (buffer limit). Fig. 8 illustrates how the queue length information is evaluated at each stage of the routing process.

This routing algorithm is heuristic, because it can only "foresee" one step ahead of the network. It provides a local best solution but does not guarantee the global optimum. Nevertheless, we believe the proposed algorithm have many unique advantages. Compared to dimension-ordered routing, the proposed routing algorithm induces shorter delays on buffers because it will be smarter in avoiding contention. Compared to hot-potato routing, the proposed routing algorithm will route faster because it evaluates the delay penalties in the forthcoming stages. This can be verified experimentally, as shown in Section 6.

4.4. On-chip switch design

We have designed a two-dimensional mesh network to test the proposed routing scheme. The node processors are tiled on the floorplan (Fig. 9a). Each side of the tile has one input and one output. The switch also serves as network interface for the node PE located at the center of the tile (Fig. 9b). The four inputs and four outputs of each tile are interconnected as shown in Fig. 9c. The switch supports concurrent links from any input channels to any output channels.

Because of the wormhole switching approach, the switch network can have limited storage and can accommodate packets with variable sizes. Because packets are segmented into flits, only one flit is processed at each input in one cycle. Each switch needs to store only a fraction of the whole packet. Long packets can be distributed over several consecutive switches and will not require extra buffer spaces. In comparison, the hot potato routing switch network described in [13,18] needs to handle the whole packet at every switch.

If the local PE is the source of the packet, the same contention-look-ahead algorithm is used. If no output is available, the node will hold the packet transmission. If the node is the destination of the packet, it will "absorb" this packet. Incoming packets will take priority over those generated/absorbed by the local PE.

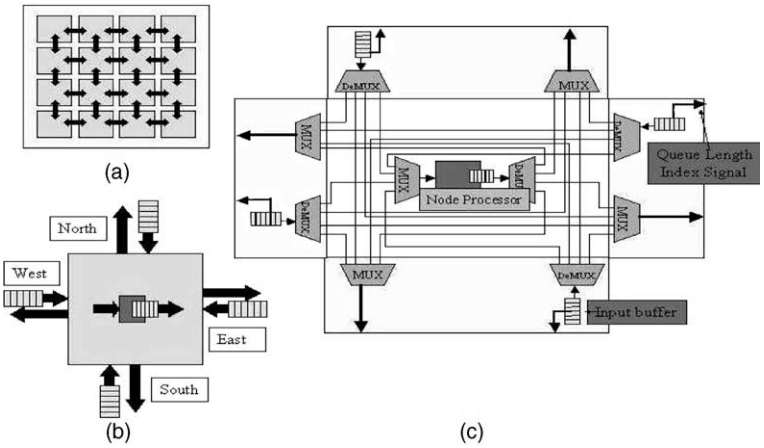


Fig. 9. Switch fabrics for on-chip networks.

The proposed switch network architecture and contention-look-ahead scheme can be applied to many existing wormhole routing algorithms. Because it foresees the contention occurrence and buffer queue length in the neighboring nodes, it helps the local nodes to make better decision to avoid potential livelock or deadlock problems.

The control signal wires are connected between any pair of neighboring nodes. The signal wires carry the input buffer queue length information of the corresponding route. The queue length value is encoded in a binary word, e.g., 1011 means the buffer queue length is 11 flit. The flit size is 64-bit, if each side of the tile uses a 2-flit buffer, with the internal queue included, the total buffer size for the switch is 640-bit.

The control portion of the routing algorithm, defined by Eqs. (1)–(3), is realized by a combinational logic module called *allocator*, shown in Fig. 10. The output channel is selected by DeMUX, and the selection is based on the comparator results of the delay penalty of each output channels. The delay penalty is either the buffer queue length of the corresponding input of the next node, or, in the case of a misroute channel, the sum of the queue length and $2 \times D_S$, which is the extra switch delay incurred with the misroute. Another DeMUX selects the misroute channels, because there could be multiple misroutes for a packet at each switch. This calculation involves two 4-input DeMUX delays, one adder delay and one comparator delay. It can be performed immediately after the address code in the header flit is available, thus minimizing the delay overhead. The switch also uses registers to store the decision taken by the header flit of a packet to keep a reserved path, until the tail flit resets it.

5. Experiment platform

We perform both qualitative as well as quantitative analysis on MPSoC and its on-chip networks. The quantitative analysis is measured from the benchmark results. Therefore, we will describe our experimental platform first before proceeding to the detailed analysis.

In multiprocessor systems-on-chip, the performance of node processors is closely coupled with the interconnect networks. On one hand, the delay of packet transmission on the network greatly affects the instruction execution of the node processors. On the other hand, the performance of the node processors will consequently affect the packet generation and delivery into the network. Therefore, comparison of different metrics of MPSoC system (e.g., execution time, energy/power consumption, etc.) requires an integrated simulation of the node processors as well as the on-chip networks.

5.1. Platform

We used RSIM as the shared-memory MPSoC simulation platform [20]. Multiple RISC processors can be integrated into RSIM. They are connected by a two-dimensional mesh interconnect network. The interconnect is 64-bit in width. Each

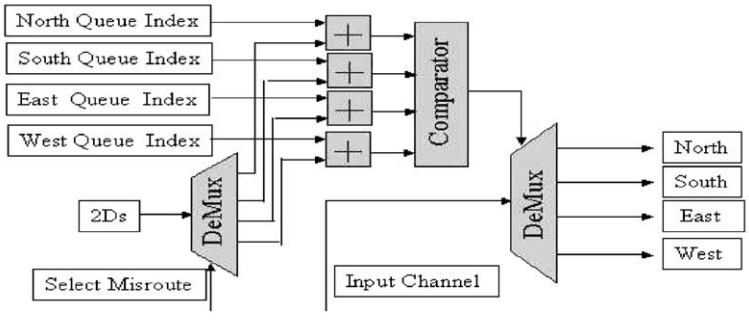


Fig. 10. Allocator circuit that implements the routing algorithm.

node processor contains two ALUs and two FPUs (floating point units), along with two levels of cache hierarchy. L1 cache is 16 KB, and L2 cache is 64 KB. Both L1 and L2 cache use write-through methods for memory updates. We use the *invalidate* approach for cache coherence synchronization. Wormhole routing is used, and the flit size is 8 byte.

RSIM integrates detailed instruction-level models of the processors and a cycle-accurate network simulator. Both the network packet delays and the instruction execution at every cycle of the node processors can therefore be traced and compared.

5.2. Benchmarks

In the following sections, we will quantitatively analyze the multiprocessor on-chip networks from different perspectives by testing different applications on our RSIM MPSoC simulation platform. A 4×4 mesh network is used in the experiments. We will first compare our proposed contention-look-ahead routing scheme with other routing algorithms, using the benchmarks *quicksort*, *sor*, *fft* and *lu*. To further analyze how different packetization schemes will affect the performance and power, we will then change the dataflow with different packet sizes. The packet payload sizes are varied from 16, 32, 64, 128–256 byte. Because the short packets are always 2-flit in length, the change of packet size is applied to long packets only. The benchmarks used in these comparison are *quicksort*, *sor*, *water*, *lu* and *mp3d*. Among these benchmarks, *water*, *lu* and *mp3d* applications are ported from the Stanford SPLASH project [21].

6. Routing algorithms comparison

The proposed on-chip network switch module as well as the routing algorithm were written in C and integrated into the RSIM routing function. Beside the interconnects on the network, adjacent processors are also connected by control wires. The control wires deliver the input buffer information to the adjacent switches.

The proposed contention-look-ahead routing algorithm was compared with dimension-ordered routing and hot potato routing. The experiments were performed with the following metrics: (1) performance improvement, (2) buffer reduction, and (3) routing with different packet sizes. As mentioned earlier, virtual channel wormhole routing requires substantial buffer spaces. Therefore, we did not consider the virtual channel approach in our experiments.

6.1. Performance improvements

Fig. 11 shows the average packet delay on the interconnect network under the three routing schemes. The packet size is 64 byte. Contention-look-ahead routing is compared with the dimension-ordered routing with different input buffer sizes (2-, 4-, 8- and 16-flit). The hot potato routing input buffer size is fixed and is equal to one packet. The delays of the other two routing schemes are normalized to the hot potato results. The packet delay is measured from the header flit entering the network until the tail flit leaves the network. Delays are expressed in clock cycles. In all four benchmarks, the hot potato routing scheme has the longest network delay. This can be explained with the following reason: The deflection in hot potato routing will create extra delays for each packet. Although the packets do not have to wait in the buffer queues, the extra latency associated with the deflections offsets the buffer delays avoided. The deflection latency will also increase as the packet size increases. Among the three routing schemes, the contention-look-ahead routing scheme achieves the shortest network delay under the same buffer size in all the benchmarks.

Larger buffer sizes help reducing the packet network delays. Although the buffer on the input channel of the switch is not big enough to store the entire packet, it can still reduce the number of intermediate switches a packet occupies when it is waiting for the next switch. This effect can also be seen from Fig. 11, as packet delays are reduced with larger buffer sizes. Nevertheless, the buffer size used in the experiments (2-, 4-, 8- and 16-flits) is still much less than that required by store-and-forward routing and virtual-cut-through routing.

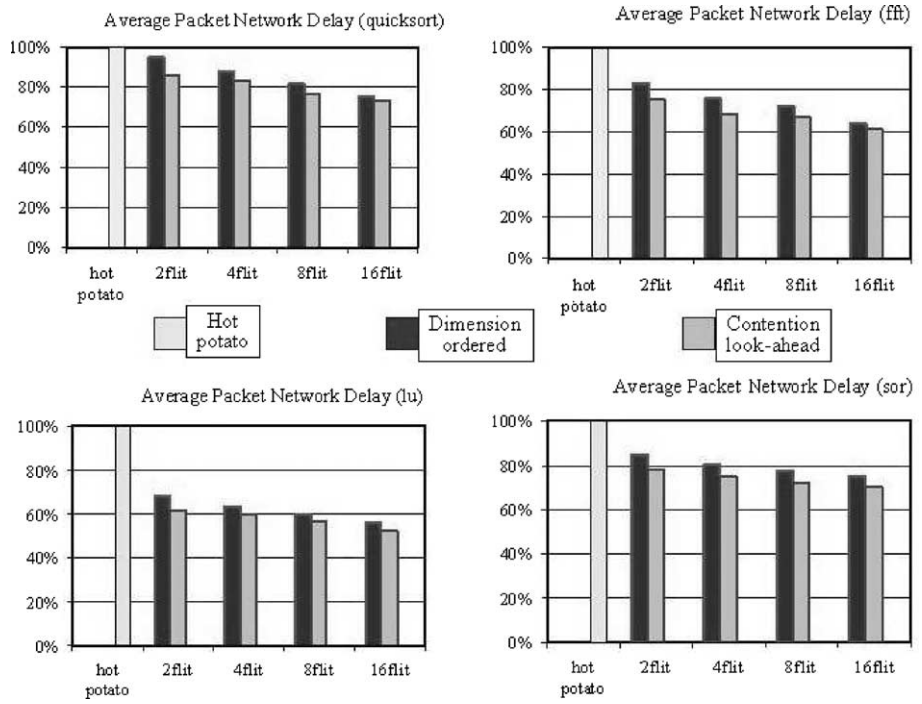


Fig. 11. Averaged packet network delays under different routing schemes (normalized to the hot-potato result).

The overall performance (total benchmark execution time) of the multiprocessor system follows the same trend as the network delays, because short network delays help to accelerate the execution process of node processors. Fig. 12 shows the results of the three routing schemes on the benchmarks. Again, results are normalized to the hot potato execution time. Hot potato routing has the longest execution time. Contention-look-ahead routing outperforms dimension-ordered routing in all cases.

6.2. Buffer reduction

In order to obtain deeper insight in the comparison between dimension-ordered routing and contention-look-ahead routing, we redraw the results from Fig. 12 in Fig. 13. The figure shows execution time reduction of each benchmark with various buffer sizes. With the proposed routing scheme, total running time on the multiprocessor platform can be reduced as much as 7.6%. In fact, contention-look-ahead routing shows larger improvement when the buffer sizes are small. As seen in Fig. 13, execution time reduction is more significant with smaller buffer size (2-flit, in the figure) than with larger buffer sizes (8-flit). This result is expected because larger buffers "help" the dimension-ordered routing to resolve the network contention and narrow its performance gap between the contention-look-ahead routing. Combining Figs. 12 and 13, we can see that in order to achieve the same performance (execution time), contention-look-ahead routing requires smaller buffers than dimension-ordered routing.

6.3. Routing with variable packet sizes

In wormhole routing, bigger packets block more channels, increase congestion and occupy the network for a longer time. We are interested to see how different routing schemes behave under variable packet sizes. Previous experiments indicate that contention-look-ahead routing achieves better performance with smaller buffer sizes, therefore, we set input buffers to be 2-flit.

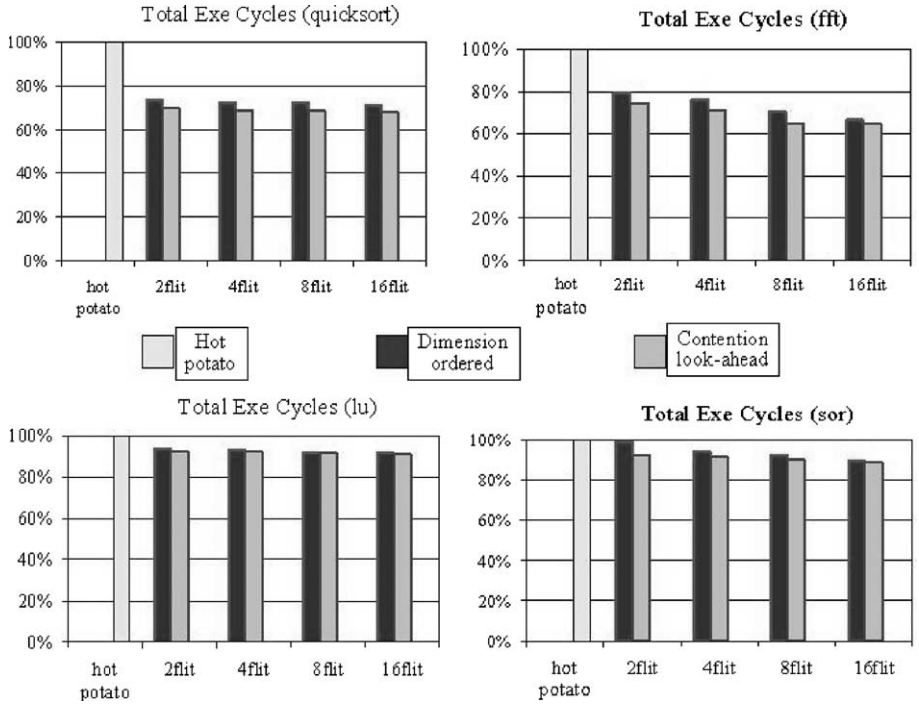


Fig. 12. Total execution time comparison between different routing schemes (normalized to the hot-potato result).

Fig. 13. Contention-look-ahead routing achieves better performance with smaller buffers.

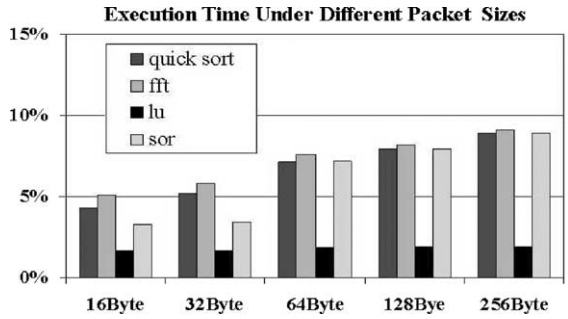


Fig. 14. Performance improvements under different packet sizes.

The packet size is then changed from 16, 32, 64, 128–256 byte. Because contention-look-ahead routing can avoid longer waiting latencies on the network, it will be more advantageous when more channels are blocked. This is confirmed by experiments. As seen from Fig. 14, the contention-look-ahead routing scheme achieves maximum improvement (9.1%) with bigger packets (256 byte). Improvement is normalized to the results of dimension-ordered routing.

7. Packet size and MPSoC performance

As mentioned in Section 1, MPSoC performance is determined by many factors. Changing

packet size affects these factors and, consequently, result in different performances. In the next few sections, we are going to analyze how different packet sizes will affect the MPSoC performance as well as system level energy consumption. We will use the proposed contention-look-ahead algorithm to perform the packet routing on the networks in our analysis.

7.1. Cache miss rate

Changing the packet payload size (for long packets) will change the L2 cache block size that can be updated in one memory fetch. If we choose a larger payload size, more cache contents will be updated. While running the same application, the cache miss rate will decrease. This effect can be observed from Fig. 15. As the packet payload size increases, both the L1 cache (Fig. 15a) and L2 cache (Fig. 15b) miss rates decrease. Decreased cache miss rate will reduce the number of packets needed for memory access.

7.2. Cache miss penalty

Whenever there is a L2 cache miss, the missed cache block needs to be fetched from the memories. The latency associated with this fetch operation is called miss penalty. When we estimate the cache miss penalty, we need to count all the delays occurring within the fetch operation. These delays include:

(1) Packetization delay—The delay associated with the packet generation procedure, e.g., encapsulating the cache content into packet payload.
(2) Interconnect delay—The signal propagation delay on the wire.
(3) Store-and-forward delay on each hop for one flit—The delay of one flit delivered from input port to output port of the switch node.
(4) Arbitration delay—The computation delay of the header flit to decide which output port to go.

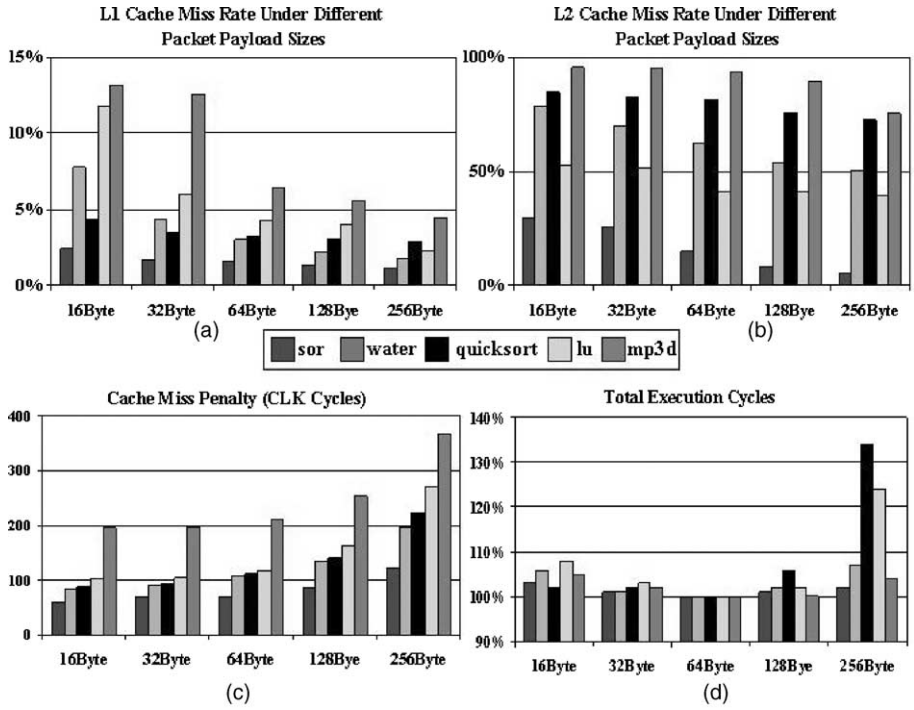


Fig. 15. Performance under different packetization schemes.

(5) Memory access delay—The READ or WRITE latencies when accessing the memory content.
(6) Contention delay—When contention occurs, the time the packets will hold transmission at the current stages, until the contention is resolved.

Among these six factors, (2)–(4) will not change significantly for packets with different sizes, because we use wormhole routing. However, delays on (1) and (5) will become longer because larger packets need longer time for packetization and memory access. Longer packets will actually cause more contention delay. This is because when wormhole routing is used, a longer packet will hold more intermediate nodes during its transmission. Other packets have to wait in the buffer, or choose alternative datapaths, which are not necessarily the shortest routes. Combining all these factors, the overall cache miss penalty will increase as the packet payload size increases, as shown from Fig. 15c.

7.3. Overall performance

The above analysis shows that although larger payload size helps to decrease the cache miss rate, it will increase the cache miss latency. Combining these two factors, there exists an optimal payload size that can achieve the minimum execution time, as seen from Fig. 15d. In order to illustrate the variation of performance, we normalized the figure to the minimum execution time of each benchmark. In our experiments, all five benchmarks achieve the best performance with 64 bytes of payload size.

8. Packet size and MPSoC energy consumption

In this section, we will analyze quantitatively the relationship between different packetization factors, and their impact on the power consumption. MPSoC power is dissipated on dynamic components as well as static components. The packetization will have impact mostly on the dynamic components, therefore, we will focus our analysis on the dynamic components only.

8.1. Contributors of MPSoC energy consumption

The MPSoC dynamic power consumption originates from three sources: the node power consumption, the shared memory power consumption and the interconnect network power consumption.

8.1.1. Node power consumption

Node power consumption comes from the operations inside each node processor, these operations include:

(1) *CPU and FPU operations.* Instructions such as *ADD*, *MOV*, *SUB etc* consume power because these operations toggle the logic gates on the datapath of processor.
(2) *L1 cache access.* L1 cache is built with fast SRAMs. When data is loaded or stored in the L1 cache, it consumes power.
(3) *L2 cache access.* L2 cache is built with slower but larger SRAMs. Whenever there is a read miss in L1 cache, or when there is write back from L1 cache, L2 cache is accessed, and consequently consumes power.

8.1.2. Shared memory power consumption

Data miss in L2 cache requires data to be fetched from memory. Data write back from L2 cache also needs to update the memory. Both operations will dissipate power when accessing the memories.

8.1.3. Interconnect network power consumption

Operations such as cache miss, data fetch, memory updates and cache synchronization all need to send packets on the interconnect network. When packets are transported on the network, energy is dissipated on the interconnect wires as well as the logic gates inside each switch. Both wires and logic gates need to be counted when we estimate the network power consumption.

Among the above three sources, the node power consumption and memory power consumption have been studied by many researchers [22]. In the following sections, we will only focus the analysis on the power consumption of interconnect networks. Later in this paper, when we combine

different sources of the power consumption and estimate the total MPSoC power consumption, we will reference the results from other research for the node processor and memory power estimation.

8.2. Network energy modeling

In this section, we will propose a quantitative modeling to estimate the power consumption of on-chip network communication. Compared with the statistical or analytical methods [23–25] used in many previous interconnect power modeling research, our proposed method provides insight on how on-chip network architectures can trade-off between different design options of CPU, cache and memories at the architectural level.

8.2.1. Bit energy of packet

When a packet travels on the interconnect network, both the wires and logic gates on the datapath will toggle as the bit-stream flips its polarity. In this paper, we use an approach similar to the one presented in [26,27] to estimate the energy consumption for the packets traveling on the network.

We adopt the concept of bit energy E_{bit} to estimate the energy consumed for each bit when the bit flips its polarity from the previous bit in the bit stream. We further decompose the bit energy E_{bit} into bit energy consumed on the interconnect wires $E_{W_{bit}}$ and the bit energy consumed on the logic gates inside the node switch $E_{S_{bit}}$, as described in the following equation (Eq. (4)).

$$E_{bit} = E_{W_{bit}} + E_{S_{bit}} \tag{4}$$

The bit energy consumed on the interconnect wire can be estimated from the total load capacitance on the interconnect. In our estimation, the total load capacitance is assumed to be proportional to the interconnect wire-length.

The bit energy consumed on the switch logic gates can be estimated from Synopsys Power Compiler simulation. Without loss of generality, we use random bit-stream as the packet payload content. We built each of the node switches in Verilog and synthesized the RTL design with 0.13 μm standard cell libraries. Then we applied different random input data streams to the inputs of the switch, and calculated the average energy consumption on each bit. Therefore, the value of E_{bit} will represent the average bit energy of a random bit-stream flowing through the interconnect network. Details of the estimation technique can be found in [26,27].

8.2.2. Packets and hops

When the source node and destination node are not placed adjacent to each other on the network, a packet needs to travel several intermediate nodes until reaching the destination. We call each of the intermediate stages a *hop* (Fig. 16).

In the mesh or torus network, there are several different alternate datapaths between source and destination, as shown in Fig. 16. When contention occurs between packets, the packets may be rerouted to different datapaths. Therefore, packet paths will vary dynamically according to the traffic condition. Packets with the same source and destination may not travel through the same number of hops, and they may not necessarily travel with the minimum number of hops.

The number of hops a packet travels greatly affects the total energy consumption needed to transport the packet from source to destination. For every hop a packet travels, the interconnect wires between the nodes will be charged and discharged as the bit-stream flows by, and the logic gates inside the node switch will toggle.

We assume a tiled floorplan implementation for MPSoC, similar to those proposed by [4,13], as shown in Fig. 16. Each node processor is placed inside a tile, and the mesh network is routed in a regular topology. Without loss of generality, we can assume all the hops in mesh network have the same interconnect length. Therefore, if we pre-

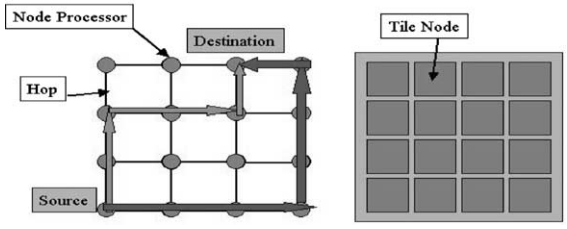


Fig. 16. Hops and alternate routes of packets.

calculate the energy consumed by one packet on one hop, E_{hop}, by counting the number of hops a packet travels, we can estimate the total energy consumed by that packet. As we mentioned earlier, the hop energy E_{hop} is the sum of the energy consumed on the interconnect wires connecting each hop, and the energy consumed on the switching node associated with that hop.

We use the hop histogram to show the total energy consumption by the packet traffic. In Fig. 17 below, histograms of the packets traveling between MPSoC processors are shown. The processors are connected by a two-dimensional mesh interconnect network. The histograms are extracted from the trace file of a *quicksort* benchmark. The histogram has n bins with $1, 2, \ldots, n$ hops, the bar on each bin shows the number of packets in each bin. We count long packets and short packets separately in the histograms.

Because E_{bit} represents the average bit energy of a random bit-stream, we can assume packets of the same length will consume the same energy per hop. Using the hop histogram of the packets, we can calculate the total network energy consumption with the following equation (Eq. (5)):

$$E_{packet} = \sum_{h=1}^{maxhops} h \times N(h)_{long} \times L_{long} \times E_{flit} + \sum_{h=1}^{maxhops} h \times N(h)_{short} \times L_{short} \times E_{flit} \quad (5)$$

where $N(h)_{packet}$ is the number of packets with h number of hops in the histogram. L_{long} and L_{short} are the lengths of long and short packets, respectively, in the unit of flit. E_{flit} is the energy consumption for one flit on each hop. Because the packets are actually segmented into flits when they are transported on the network, we only need to calculate the energy consumption for one flit, E_{flit}. The energy of one packet per hop E_{hop} can be calculated by multiplying the number of flits the packet contains.

8.2.3. Energy model calculation

We assume that each tile of node processor is 2 mm × 2 mm in dimension, and they are placed regularly on the floorplan, as shown in Fig. 16. We assume 0.13 μm technology is used, and the wire load capacitance is 0.50 fF per micron. Under these assumption, the energy consumed by one flit on one hop interconnect is 0.174 nJ.

The energy consumed in the switch for one hop is calculated from Synopsys Power Compiler. We calculate the bit energy on the logic gates in a way similar to that used in [26]. We use 0.13 μm standard cell library, and the energy consumed by one flit on one hop switch is 0.096 nJ. Based on these calculation, the flit energy per hop $E_{flit} = 0.27$ nJ.

8.3. Packetization and energy consumption

Eq. (5) in Section 8.2 shows that the power consumption of packetized dataflow on MPSoC network is determined by the following three factors: (1) the number of packets on the network, (2) the energy consumed by each packet on one hop, and (3) the number of hops each packet travels. Different packetization schemes affect these factors differently and, consequently, affect the network power consumption. We summarize these effects and list them below:

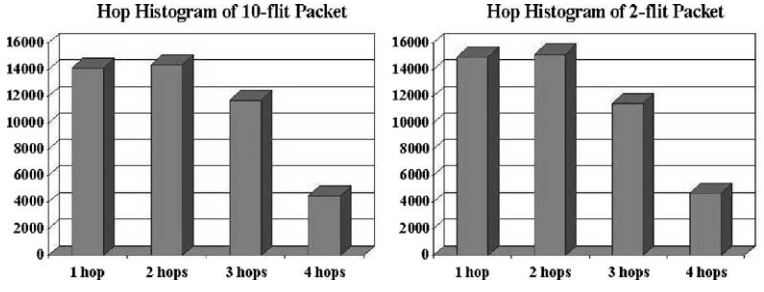


Fig. 17. Hop histogram of long and short packets.

(1) Packets with larger payload size will decrease the cache miss rate and consequently decrease the number of packets on the network. This effect can be seen from Fig. 18a. It shows the average number of packets on the network (traffic density) at one clock cycle (Y-axis indicates the packet count). As the packet size increases, the number of packets per clock cycle decreases accordingly. Actually, with the same packet size, the traffic density of different benchmarks is consistent with the miss penalty. By comparing Fig. 18a with Fig. 15c, we see that if the packet length stays the same, higher traffic density causes longer miss latency.
(2) Larger packet size will increase the energy consumed per packet, because there are more bits in the payload.
(3) As discussed in Section 7, larger packets will occupy the intermediate node switches for a longer time, and cause other packets to be re-routed to non-shortest datapaths. This leads to more contention that will increase the total number of hops needed for packets traveling from source to destination. This effect is shown in Fig. 18b which shows the average number of hops a packet travels between source and destination. As packet size (payload size) increases, more hops are needed to transport the packets.

Actually, increasing the cache block size will not decrease the cache miss rate proportionally [31]. Therefore, the decrease of packet count cannot compensate for the increase of energy consumed per packet caused by the increase of packet length. Larger packet size also increases the hop counts on the datapath. Fig. 20a shows the combined effects of these factors under different packet sizes. The values are normalized to the measurement of 16 byte. As packet size increases, energy consumption on the interconnect network will increase.

Although increase of packet size will increase the energy dissipated on the network, it will decrease the energy consumption on cache and memory. Because larger packet sizes will decrease the cache miss rate, both cache energy consumption and memory energy consumption will be reduced. This relationship can be seen from Fig. 19. It shows the energy consumption by cache and memory under different packet sizes. The access energy of each cache and memory instruction is estimated based on the work from [28,29]. The energy in the figure is normalized to the value of 256 byte, which achieves the minimum energy consumption.

The total energy dissipated on MPSoC comes from non-cache instructions (instructions that do not involve cache access) of each node processors, the caches and the shared memories as well as the interconnect network. In order to see the packetization impact on the total system energy consumption, we put all MPSoC energy contributors together and see how the energy changes under different packet sizes. The results are shown in Fig. 20b. From this figure, we can see that the overall MPSoC energy will decrease as packets size in-

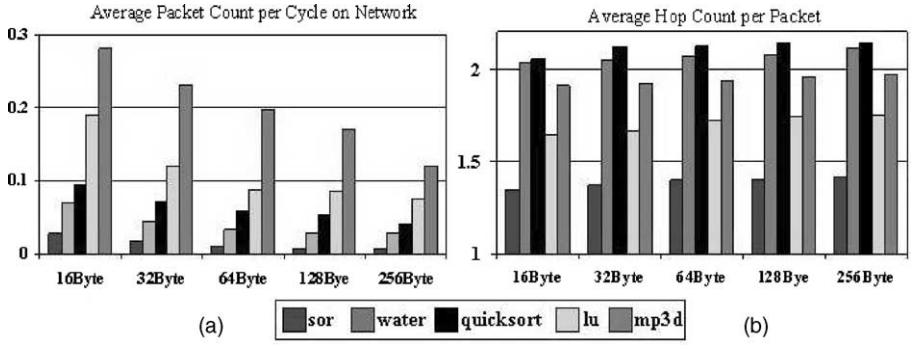


Fig. 18. Contention occurrence changes as packet payload size increases.

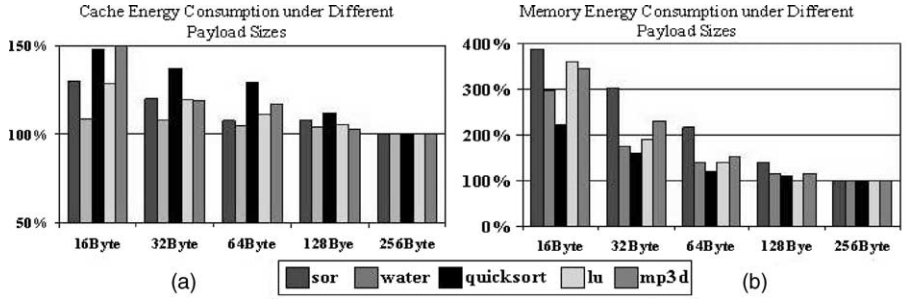


Fig. 19. Cache and memory energy decrease as packet payload size increases.

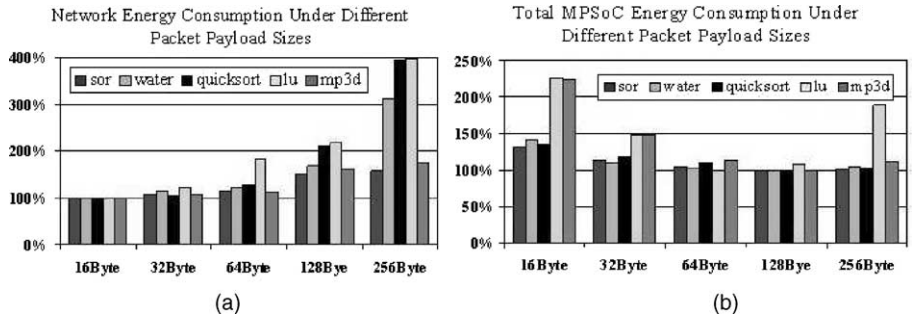


Fig. 20. Network and total MPSoC energy consumption under different packet payload sizes.

creases. However, when the packets are too large, as in the case of 256 byte in the figure, the total MPSoC energy will increase. This is because when the packet becomes too large, the increase of interconnect network energy will outgrow the decrease of energy on cache, memory and non-cache instructions. In our simulation, the non-cache instruction energy consumption is estimated based on the techniques presented in [30], and it does not change significantly under different packet sizes.

9. Packetization impact analysis

Although the specific measurement values in the experiments are technology and platform dependent, we believe the analysis will hold for different MPSoC implementations. We summarize our analysis qualitatively as follows (Fig. 21).

Large packet size decreases the cache miss rates of MPSoC but increases the miss penalty. The increase of miss penalty is caused by the increase of packetization delay, memory access delay, as well as contention delay on the network. As shown qualitatively in Fig. 21a, the cache miss rate saturates with the increase of packet size. Nevertheless, the miss penalty increases faster than linearly. Therefore, there exists an optimal packet size to achieve best performance.

The energy spent on the interconnect network increases as the packet size increases. Three factors play roles in this case (Fig. 21b). (1) Longer packets, i.e., larger cache lines, reduce the cache miss rate, hence reduce the packet count. Nevertheless, the packet count does not fall linearly with the increase of packet size. (2) The energy consumption per packet $\times$ hop increases in a linear fashion with the increase of packet length. If we ignore the overhead of packet header and tail, this increase is proportional to packet size. (3) The average number of hops per packet on the network also increases with the packet length. The combined effect causes the network energy to increase as the packet size increases.

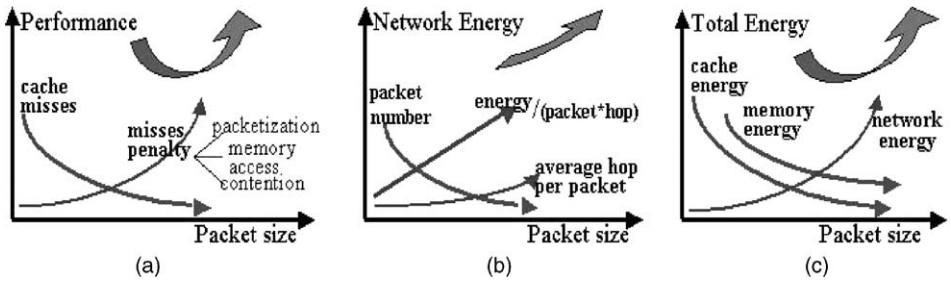


Fig. 21. Qualitative analysis of packet size impact.

The total MPSoC system energy is dominated by the sum of three factors as the packet size increases (Fig. 21c). (1) Cache energy will decrease. (2) Memory energy will decrease as well. (3) Network energy will increase over-linearly. In our benchmarks, the non-cache instruction energy does not change significantly. The overall trend depends on the breakdown among the three factors. Our experiments show that there exists a packet size that minimizes the overall energy consumption. Moreover, if the network energy contributes a major part of the total system energy consumption, which is expected to happen as VLSI technology moves to nano-meter domain, the MPSoC energy will eventually increase with the packet size.

10. Conclusion

The performance and energy consumption of shared-memory on-chip multiprocessor systems are highly dependent on the internode dataflow packetization schemes as well as on-chip network architectures. On-chip network communication can benefit from the abundant wiring resources as well as floor-planning locality among processing elements and switch nodes. In contrast, network routing strategies are limited by on-chip buffers that are expensive to implement and power-hungry during operation. In this paper, we proposed a contention-look-ahead routing scheme that exploits increased wiring resources while reducing buffer requirements. The scheme achieves better performance with significantly less buffer space usage than traditional low-buffer-space routing algorithms. We further introduced an on-chip interconnect network energy model, and then analyzed and quantified the effect of packet size variation on performance and energy consumption. Although the results are reported on a mesh network, the methodology presented in this paper is general and can be extended to cope with other on-chip network architectures.

Acknowledgements

We acknowledge the supports from the *MARCO GSRC* center, under the contract SA3276JB-2 and its continuation.

References

[1] W.O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, L. Gauthier, M. Diaz-Nava, A.A. Jerraya, Multiprocessor SoC platforms: a component-based design approach, IEEE Design and Test of Computers 19 (6) (2002) 52–63.

[2] R. Ho, K. Mai, M. Horowitz, The future of wires, Proceedings of the IEEE, April (2001) 490–504.

[3] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, D. Lindqvist, Network on chip: an architecture for billion transistor era, Proceeding of the IEEE NorChip Conference, November 2000, pp. 166–173.

[4] W. Dally, B. Toles, Route packets, not wires: on-chip interconnection networks, Proceedings of the 38th Design Automation Conference, June 2001, pp. 684–689.

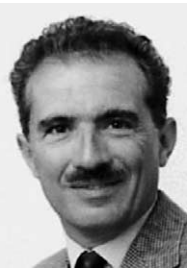
[5] L. Benini, G. De Micheli, Networks on chips: a new SoC paradigm, IEEE Computer 35 (1) (2002) 70–78.

[6] B. Ackland et al., A single chip, 1.6-billion,16-MAC/s multiprocessor DSP, IEEE Journal of Solid-State Circuits (2000) 412–424, March.

[7] E. Rijpkema, K.G.W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, E. Waterlander, Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip, Proceedings of Design Automation and Test Conference in Europe, March 2003, pp. 350–355.
[8] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, B. Verghese, Piranha: a scalable architecture based on single-chip multiprocessing, Proceedings of 27th Annual International Symposium on Computer Architecture, 2000, pp. 282–293.
[9] L. Hammond, B. Hubbert, M. Siu, M. Prabhu, M. Chen, K. Olukotun, The stanford hydra CMP, IEEE MICRO Magazine (2000) 71–84.
[10] D.E. Culler, J.P. Singh, A. Gupta, Parallel computer architecture: a hardware/software approach, Morgan Kaufmann Publishers, 1998.
[11] J. Duato, S. Yalamanchili, L. Ni, Interconnection Networks, an Engineering Approach, IEEE Computer Society Press, 1997.
[12] J. Wu, A deterministic fault-tolerant and deadlock-free routing protocol in 2-D meshes based on odd-even turn model, Proceedings of the 16th international conference on Supercomputing, 2002, pp. 67–76.
[13] S. Kumar, A. Jantsch, J. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrjä, A. Hemani, A network on chip architecture and design methodology, Proceedings of IEEE Computer Society Annual Symposium on VLSI, April 2002, pp. 105–112.
[14] M. Forsell, A scalable high-performance computing solution for networks on chips, IEEE Micro 22 (5) (2002) 46–55.
[15] P. Gherrier, A. Greiner, A generic architecture for on-chip packet-switched interconnections, Proceedings of Design Automation and Test in Europe, March 2000, pp. 250–255.
[16] F. Karim, A. Nguyen, S. Dey, On-chip communication architecture for OC-768 network processors, Proceedings of 38th Design Automation Conference, June 2001, pp. 678–683.
[17] W.J. Dally, H. Aoki, Deadlock -free adaptive routing in multicomputer networks using virtual channels, IEEE Transactions on Parallel and Distributed Systems (1993) 466–475, April.
[18] E. Nilsson, M. Millberg, J. Oberg, A. Jantsch, Load distribution with the proximity congestion awareness in a networks on chip, Prodeedings of Design Automation and Test in Europe, March 2003, pp. 1126–1127.
[19] U. Feige, P. Raghavan, Exact analysis of hot-potato routing, Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, October 1992, pp. 553–562.
[20] C.J. Hughes, V.S. Pai, P. Ranganathan, S.V. Adve, Rsim: simulating shared-memory multiprocessors with ILP processors, IEEE Computer 35 (2) (2002) 40–49.
[21] J.P. Singh, W. Weber, A. Gupta, SPLASH: stanford parallel applications for shared-memory, Computer Architecture News 20 (1) (1992) 5–44.
[22] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno. Efficient power estimation techniques for system-on-chip design, Proceedings of Design Automation and Test in Europe, March 2000, pp. 27–32.
[23] A.G. Wassal, M.A. Hasan, Low-power system-level design of VLSI packet switching fabrics, IEEE Transactions on CAD of Integrated Circuits and Systems (2001) 723–738, June.
[24] C. Patel, S. Chai, S. Yalamanchili, D. Shimmel, Power constrained design of multiprocessor interconnection networks, Proceedings of IEEE International Conference on Computer Design, 1997, pp. 408–416.
[25] D. Langen, A. Brinkmann, U. Ruckert, High level estimation of the area and power consumption of on-chip interconnects, Proceedings of 13th IEEE International ASIC/SOC Conference, September 2000, pp. 297–301.
[26] T.T. Ye, L. Benini, G. De Micheli, Analysis of power consumption on switch fabrics in network routers, Proceedings of the 39th Design Automation Conference, June 2002, pp. 524–529.
[27] J. Hu, R. Marculescu, Energy-aware mapping for tile-based NOC architectures under performance constraints, Proceedings of ASP-Design Automation Conference, January 2003, pp. 233–239.
[28] E. Geethanjali, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, Memory system energy: influence of hardware–software optimizations, Proceedings of International Symposium on Low Power Design and Electronics, July 2000, pp. 244–246.
[29] W.T. Shiue, C. Chakrabarti, Memory exploration for low power, embedded systems, Proceedings of the 36th Design Automation Conference, June, 1999, pp. 140–145.
[30] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, R. Zafalon, Energy estimation and optimization of embedded VLIW processors based on instruction clustering, Proceedings of 39th Design Automation Conference, June 2002, pp. 886–891.
[31] D.A. Patterson, J. Hennessy, Computer organization and design, the hardware/software interface, Morgan Kaufmann Publishers, 1998.

Terry Tao Ye is currently a PhD candidate in the Department of Electrical Engineering at Stanford University. His PhD research interests include on-chip interconnect communication network design, embedded system design and ASIC datapath design and optimization. Before his PhD study, he worked for four years at Synopsys Inc. as a senior R/D engineer. Terry Ye received his BSEE from Tsinghua University, Beijing.

Luca Benini is an associate professor in the Department of Electronics and Computer Science at the University of Bologna. His research interests include the design of portable systems and all aspects of computer-aided digital-circuit design, with special emphasis on low-power applications. Benini received a PhD in Electrical Engineering from Stanford University.



Giovanni De Micheli is a professor of Electrical Engineering and, by courtesy, of Computer Science, at Stanford University. His research interests include design technologies for integrated circuits and systems, with particular emphasis on synthesis, system level design, hardware and software co-design, and low-power design. De Micheli received a PhD in Electrical Engineering and Computer Science from the University of California at Berkeley.