Chapter 2
On-Chip Networks for Multicore Systems

Li-Shiuan Peh, Stephen W. Keckler, and Sriram Vangal

Abstract With Moore's law supplying billions of transistors, and uniprocessor architectures delivering diminishing performance, multicore chips are emerging as the prevailing architecture in both general-purpose and application-specific markets. As the core count increases, the need for a scalable on-chip communication fabric that can deliver high bandwidth is gaining in importance, leading to recent multicore chips interconnected with sophisticated on-chip networks. In this chapter, we first present a tutorial on on-chip network architecture fundamentals including on-chip network interfaces, topologies, routing, flow control, and router microarchitectures. Next, we detail case studies on two recent prototypes of on-chip networks: the UT-Austin TRIPS operand network and the Intel TeraFLOPS on-chip network. This chapter organization seeks to provide the foundations of on-chip networks so that readers can appreciate the different design choices faced in the two case studies. Finally, this chapter concludes with an outline of the challenges facing research into on-chip network architectures.

2.1 Introduction

The combined pressures from ever-increasing power consumption and the diminishing returns in performance of uniprocessor architectures have led to the advent of multicore chips. With growing number of transistors at each technology generation, and multicore chips' modular design enabling a reduction in design complexity, this multicore wave looks set to stay, in both general-purpose computing chips as well as application-specific systems-on-a-chip. Recent years saw every industry chip vendor releasing multicore products, with higher and higher core counts.

L.-S. Peh (✉)
Princeton University, Princeton, New Jersey 08544, USA
e-mail: peh@princeton.edu

As the number of on-chip cores increases, a scalable and high-bandwidth communication fabric to connect them becomes critically important [34]. As a result, packet-switched on-chip networks are fast replacing buses and crossbars to emerge as the pervasive communication fabric in many-core chips. Such on-chip networks have routers at every node, connected to neighbors via short local interconnects, while multiplexing multiple packet flows over these interconnects to provide high, scalable bandwidth. This evolution of interconnect, as core count increases, is clearly illustrated in the choice of a crossbar interconnect in the 8-core Sun Niagara [29], two packet-switched rings in the 9-core IBM Cell [23], and five packet-switched meshes in the 64-core Tilera TILE64 [3].

While on-chip networks can leverage ideas from prior multi chassis interconnection networks in supercomputers [1], clusters of workstations [21], and Internet routers [14], the design requirements facing on-chip networks differ so starkly in magnitude from those of prior multi chassis networks that novel designs are critically needed. On the plus side, by moving on-chip, the I/O bottlenecks that faced prior multi chassis interconnection networks are alleviated substantially: on-chip interconnects supply bandwidth that is orders of magnitude higher than off-chip I/Os, yet obviating the inherent overheads of off-chip I/O transmission. Unfortunately, while this allows on-chip networks to deliver scalable, high-bandwidth communications, challenges are abound due to highly stringent constraints. Specifically, on-chip networks must supply high bandwidth at ultra-low latencies, with a tight power envelope and area budget.

2.2 Interconnect Fundamentals

An on-chip network architecture can be defined by four parameters: its topology, routing algorithm, flow control protocol, and router micro architecture. The topology of a network dictates how the nodes and links are connected. While the topology determines all possible paths a message can take through the network, the routing algorithm selects the specific path a message will take from source to destination. The flow control protocol determines how a message actually traverses the assigned route, including when a message leaves a router through the desired outgoing link and when it must be buffered. Finally, the micro architecture of a router realizes the routing and flow control protocols and critically shapes its circuits implementation.

Throughout this section, we will discuss how different choices of the above four parameters affect the overall cost–performance of an on-chip network. Clearly, the cost–performance of an on-chip network depends on the requirements faced by its designers. Latency is a key requirement in many on-chip network designs, where network latency refers to the delay experienced by messages as they traverse from source to destination. Most on-chip networks must also ensure high throughput, where network throughput is the peak bandwidth the network can handle. Another metric that is particularly critical in on-chip network design is network power, which approximately correlates with the activity in the network as well as its complexity. Network complexity naturally affects its area footprint, which is a concern for on-chip networks where area devoted to communication detracts from the area available for processing and memory. Tolerating process variability and faults while delivering deterministic or bounded performance guarantees (including network reliability and quality-of-service) is a design constraint of increasing importance.

2.2.1 Interfacing an On-Chip Network to a Multicore System

Many different ways of interfacing the on-chip network with the multicore system have been developed. Explicit interfaces involve modifying the instruction set architecture (ISA) so that software (either the programmer or the compiler) can explicitly indicate when and where messages are to be delivered, and conversely when and from whom messages are to be received. For instance, adding *send* and *receive* instructions to the ISA allows a programmer/compiler to indicate message deliveries and receipts to the network. Implicit interfaces, on the other hand, infer network sends/receives without the knowledge of software. An example of an implicit interface is a shared-memory multicore system, in which memory loads and stores at one core implicitly trigger network messages to other cores for maintaining coherence among the caches.

Another dimension where network interfaces differ lies in where the network physically connects to a multicore system, which can be any point along the datapath of a multicore system. For example, a network can connect into the bypass paths between functional units, to registers, to level-1 cache, to level-2 and higher level caches, to the memory bus, or to the I/O bus. The further the network interface is from the functional units, the longer the delay involved in initiating a message. A network that interfaces through registers allows messages to be injected into the network once the register is written in the processor pipeline, typically with a latency on the order of a couple of cycles. A network that interfaces through the I/O bus requires heavier weight operations including interrupt servicing and pipeline flushing, requiring latencies on the order of microseconds.

Finally, while the logical unit of communication for a multicore system is a message, most on-chip networks handle communications in units of packets. So, when a message is injected into the network, it is first segmented into packets, which are then divided into fixed-length flits, short for flow control digits. For instance, a 128-byte cache line sent from a sharer to a requester will be injected as a message, and divided into multiple packets depending on the maximum packet size. Assuming a maximum packet size of 32 bytes, the entire cache line will be divided into four packets. Each packet consists of a head flit that contains the destination address, multiple body flits, and a tail flit that indicate the end of a packet; the head, body, and tail flits each contain a portion of the cache line. For instance, for a flit size of 64 bits, the packet will consist of $(32\times 8)/64 = 4$ flits: 1 head, 2 body and 1 tail, ignoring the extra bits needed to encode the destination, flow control information for the network, and the bits needed to indicate the action to be performed upon receiving the packet. Since only the head flit contains the destination, all flits of a packet follow the same route.

2.2.2 Topology

The effect of a topology on overall network cost–performance is profound. A topology determines the number of hops (or routers) a message must traverse as well as the interconnect lengths between hops, thus influencing network latency significantly. As traversing routers and links incurs energy, a topology's effect on hop count also directly affects network energy consumption. As for its effect on throughput, since a topology dictates the total number of alternate paths between nodes, it affects how well the network can spread out traffic and thus the effective bandwidth a network can support. Network reliability is also greatly influenced by the topology as it dictates the number of alternative paths for routing around faults. The implementation complexity cost of a topology depends on two factors: the number of links at each node (node degree) and the ease of laying out a topology on a chip (wire lengths and the number of metal layers required).

Figure 2.1 shows three commonly used on-chip topologies. For the same number of nodes, and assuming uniform random traffic where every node has an equal probability of sending to every other node, a ring (Fig. 2.1a) will lead to higher hop count than a mesh (Fig. 2.1b) or a torus [11] (Fig. 2.1c). For instance, in the figure shown, assuming bidirectional links and shortest-path routing, the maximum hop count of the ring is 4, that of a mesh is also 4, while a torus improves it to 2. A ring topology also offers fewer alternate paths between nodes than a mesh or torus, and thus saturates at a lower network throughput for most traffic patterns. For instance, a message between nodes A and B in the ring topology can only traverse one of two paths in a ring, but in a 3×3 mesh topology, there are six possible paths. As for network reliability, among these three networks, a torus offers the most tolerance to faults because it has the highest number of alternative paths between nodes.

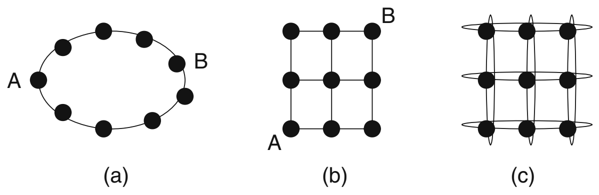


Fig. 2.1 Common on-chip network topologies: (**a**) ring, (**b**) mesh, and (**c**) torus

While rings have poorer performance (latency, throughput, energy, and reliability) when compared to higher-dimensional networks, they have lower implementation overhead. A ring has a node degree of 2 while a mesh or torus has a node degree of 4, where node degree refers to the number of links in and out of a node. A higher node degree requires more links and higher port counts at routers. All three topologies featured are two-dimensional planar topologies that map readily to a single metal layer, with a layout similar to that shown in the figure, except for tori which should be arranged physically in a folded manner to equalize wire lengths (see Fig. 2.2), instead of employing long wrap-around links between edge nodes. A torus illustrates the importance of considering implementation details in comparing alternative topologies. While a torus has lower hop count (which leads to lower delay and energy) compared to a mesh, wire lengths in a folded torus are twice that in a mesh of the same size, so per-hop latency and energy are actually higher. Furthermore, a torus requires twice the number of links which must be factored into the wiring budget. If the available wiring bisection bandwidth is fixed, a torus will be restricted to narrower links than a mesh, thus lowering per-link bandwidth, and increasing transmission delay. Determining the best topology for an on-chip network subject to the physical and technology constraints is an area of active research.

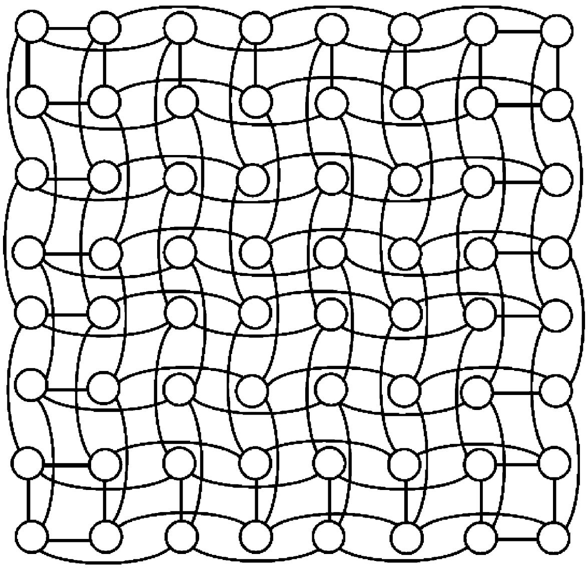


Fig. 2.2 Layout of a 8×8 folded torus

2.2.3 Routing

The goal of the routing algorithm is to distribute traffic evenly among the paths supplied by the network topology, so as to avoid hotspots and minimize contention, thus improving network latency and throughput. In addition, the routing algorithm is the critical component in fault tolerance: once faults are identified, the routing algorithm must be able to skirt the faulty nodes and links without substantially affecting network performance. All of these performance goals must be achieved while adhering to tight constraints on implementation complexity: routing circuitry can stretch critical path delay and add to a router's area footprint. While energy overhead of routing circuitry is typically low, the specific route chosen affects hop count directly, and thus substantially affects energy consumption.

While numerous routing algorithms have been proposed, the most commonly used routing algorithm in on-chip networks is dimension-ordered routing (DOR) due to its simplicity. With DOR, a message traverses the network dimension-by-dimension, reaching the coordinate matching its destination before switching to the next dimension. In a two-dimensional topology such as the mesh in Fig. 2.3, dimension-ordered routing, say X–Y routing, sends packets along the X-dimension first, followed by the Y-dimension. A packet traveling from (0,0) to (2,3) will first traverse two hops along the X-dimension, arriving at (2,0), before traversing three hops along the Y-dimension to its destination. Dimension-ordered routing is an example of a deterministic routing algorithm, in which all messages from node A to B will always traverse the same path. Another class of routing algorithms are oblivious ones, where messages traverse different paths from A to B, but the path is selected without regards to the actual network situation at transmission time. For instance, a router could randomly choose among alternative paths prior to sending a message. Figure 2.3 shows an example where messages from (0,0) to (2,3) can be randomly sent along either the Y–X route or the X–Y route. A more sophisticated routing algorithm can be adaptive, in which the path a message takes from A to B depends on network traffic situation. For instance, a message can be going along the X–Y route, sees congestion at (1,0)'s east outgoing link and instead choose to take the north outgoing link towards the destination (see Fig. 2.3).

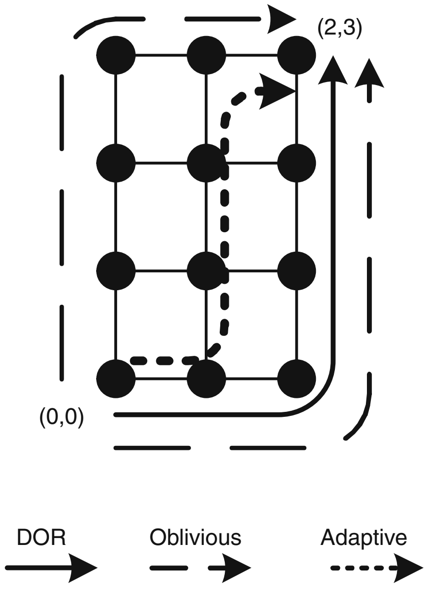


Fig. 2.3 *DOR* illustrates an X–Y route from (0,0) to (2,3) in a mesh, while *Oblivious* shows two alternative routes (X–Y and Y–X) between the same source–destination pair that can be chosen obliviously prior to message transmission. *Adaptive* shows a possible adaptive route that branches away from the X–Y route if congestion is encountered at (1,0)

In selecting or designing a routing algorithm, not only must its effect on delay, energy, throughput, and reliability be taken into account, most applications also require the network to guarantee deadlock freedom. A deadlock occurs when a cycle exists among the paths of multiple messages. Figure 2.4 shows four gridlocked (and deadlocked) messages waiting for links that are currently held by other messages and prevented from making forward progress: The packet entering router A from the South input port is waiting to leave through the East output port, but another packet is holding onto that exact link while waiting at router B to leave via the South output port, which is again held by another packet that is waiting at router C to leave via the West output port and so on.

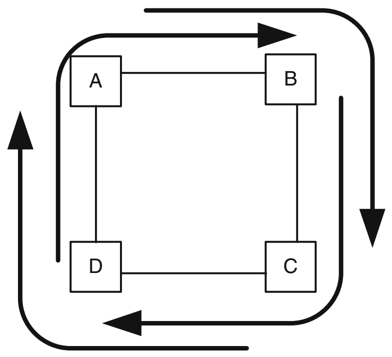


Fig. 2.4 A classic network deadlock where four packets cannot make forward progress as they are waiting for links that other packets are holding on to

Deadlock freedom can be ensured in the routing algorithm by preventing cycles among the routes generated by the algorithm, or in the flow control protocol by preventing router buffers from being acquired and held in a cyclic manner [13]. Using the routing algorithms we discussed above as examples, dimension-ordered routing is deadlock-free since in X–Y routing, there cannot be a turn from a Y link to an X link, so cycles cannot occur. Two of the four turns in Fig. 2.4 will not be permitted, so a cycle is not possible. The oblivious algorithm that randomly chooses between X–Y or Y–X routes is not deadlock-free because all four turns from Fig. 2.4 are possible leading to potential cycles in the link acquisition graph. Likewise, the adaptive route shown in Fig. 2.3 is a superset of oblivious routing and is subject to potential deadlock. A network that uses a deadlock-prone routing algorithm requires a flow control protocol that ensures deadlock freedom, as discussed in Sect. 2.2.4.

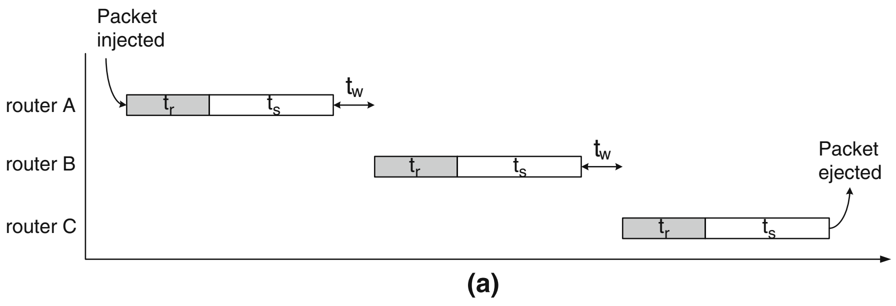
Routing algorithms can be implemented in several ways. First, the route can be embedded in the packet header at the source, known as source routing. For instance, the X–Y route in Fig. 2.3 can be encoded as <*E, E, N, N, N, Eject*>, while the Y–X route can be encoded as <*N, N, N, E, E, Eject*>. At each hop, the router will read the left-most direction off the route header, send the packet towards the specified outgoing link, and strip off the portion of the header corresponding to the current hop. Alternatively, the message can encode the coordinates of the destination, and comparators at each router determine whether to accept or forward the message. Simple routing algorithms are typically implemented as combinational circuits within the router due to the low overhead, while more sophisticated algorithms are realized using routing tables at each hop which store the outgoing link a message should take to reach a particular destination. Adaptive routing algorithms need mechanisms to track network congestion levels, and update the route. Route adjustments can be implemented by modifying the header, employing combinational circuitry that accepts as input these congestion signals, or updating entries in a routing table. Many congestion-sensitive mechanisms have been proposed, with the simplest being tapping into information that is already captured and used by the flow control protocol, such as buffer occupancy or credits.

2.2.4 Flow Control

Flow control governs the allocation of network buffers and links. It determines when buffers and links are assigned to which messages, the granularity at which they are allocated, and how these resources are shared among the many messages using the network. A good flow control protocol lowers the latency experienced by messages at low loads by not imposing high overhead in resource allocation, and pushes network throughput through enabling effective sharing of buffers and links across messages. In determining the rate at which packets access buffers (or skip buffer access altogether) and traverse links, flow control is instrumental in determining network energy and power. Flow control also critically affects network quality-of-service since it determines the arrival and departure time of packets at each hop. The implementation complexity of a flow control protocol includes the complexity of the router microarchitecture as well as the wiring overhead imposed in communicating resource information between routers.

In *store-and-forward* flow control [12], each node waits until an entire packet has been received before forwarding any part of the packet to the next node. As a result, long delays are incurred at each hop, which makes them unsuitable for on-chip networks that are usually delay-critical. To reduce the delay packets experience at each hop, *virtual cut-through* flow control [24] allows transmission of a packet to begin before the entire packet is received. Latency experienced by a packet is thus drastically reduced, as shown in Fig. 2.5. However, bandwidth and storage are still allocated in packet-sized units. Packets still only move forward if there is enough storage to hold the entire packet. On-chip networks with tight area and power constraints find it difficult to accommodate the large buffers needed to support virtual cut-through (assuming large packets).

Like virtual cut-through flow control, *wormhole* flow control [11] cuts through flits, allowing flits to move on to the next router as soon as there is sufficient buffering for this flit. However, unlike store-and-forward and virtual cut-through flow control, wormhole flow control allocates storage and bandwidth to flits that are smaller than a packet. This allows relatively small flit-buffers to be used in each router, even for large packet sizes. While wormhole flow control uses buffers effectively, it makes inefficient use of link bandwidth. Though it allocates storage and bandwidth



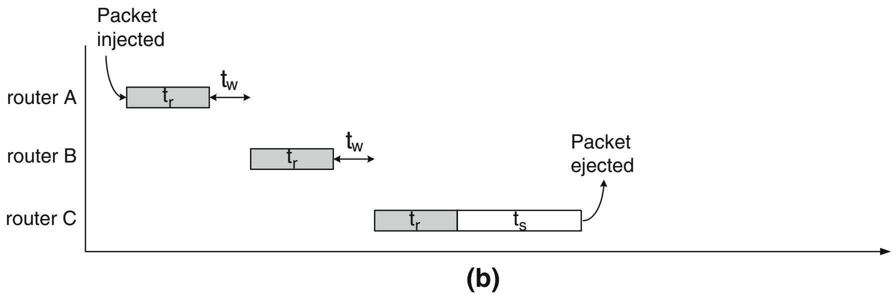


Fig. 2.5 Timing for (**a**) store-and-forward and (**b**) cut-through flow control at low loads, where t_r refers to the delay routing the head flit through each router, t_s refers to the serialization delay transmitting the remaining flits of the packet through each router, and t_w refers to the time involved in propagating bits across the wires between adjacent routers. Wormhole and virtual-channel flow control have the same timing as cut-through flow control at low loads

in flit-sized units, a link is held for the duration of a packet's lifetime in the router. As a result, when a packet is blocked, all of the physical links held by that packet are left idle. Throughput suffers because other packets queued behind the blocked packet are unable to use the idle physical links.

Virtual-channel flow control [9] improves upon the link utilization of wormhole flow control, allowing blocked packets to be passed by other packets. A virtual channel (VC) consists merely of a separate flit queue in the router; multiple VCs share the physical wires (physical link) between two routers. Virtual channels arbitrate for physical link bandwidth on a flit-by-flit basis. When a packet holding a virtual channel becomes blocked, other packets can still traverse the physical link through other virtual channels. Thus, VCs increase the utilization of the critical physical links and extend overall network throughput. Current on-chip network designs overwhelmingly adopt wormhole flow control for its small area and power footprint, and use virtual channels to extend the bandwidth where needed. Virtual channels are also widely used to break deadlocks, both within the network, and for handling system-level deadlocks.

Figure 2.6 illustrates how two virtual channels can be used to break a cyclic deadlock in the network when the routing protocol permits a cycle. Here, since each VC is associated with a separate buffer queue, and since every VC is time-multiplexed

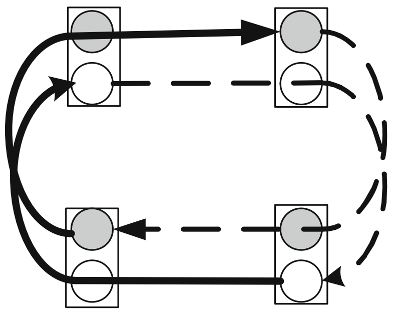


Fig. 2.6 Two virtual channels (denoted by *solid* and *dashed lines*) and their associated separate buffer queues (denoted as two *circles* at each router) used to break the cyclic route deadlock in Fig. 2.4

onto the physical link cycle-by-cycle, holding onto a VC implies holding onto its associated buffer queue rather than locking down a physical link. By enforcing an order on VCs, so that lower-priority VCs cannot request and wait for higher-priority VCs, there cannot be a cycle in resource usage. At the system level, messages that can potentially block on each other can be assigned to different message classes that are mapped onto different virtual channels within the network, such as request and acknowledgment messages of coherence protocols. Implementation complexity of virtual channel routers will be discussed in detail next in Sect. 2.2.5 on router microarchitecture.

Buffer backpressure: Unlike broadband networks, on-chip network are typically not designed to tolerate dropping of packets in response to congestion. Instead, buffer backpressure mechanisms stall flits from being transmitted when buffer space is not available at the next hop. Two commonly used buffer backpressure mechanisms are credits and on/off signaling. Credits keep track of the number of buffers available at the next hop, by sending a credit to the previous hop when a flit leaves and vacates a buffer, and incrementing the credit count at the previous hop upon receiving the credit. On/off signaling involves a signal between adjacent routers that is turned off to stop the router at the previous hop from transmitting flits when the number of buffers drop below a threshold, with the threshold set to ensure that all in-flight flits will have buffers upon arrival.

2.2.5 Router Microarchitecture

How a router is built determines its critical path delay, per-hop delay, and overall network latency. Router microarchitecture also affects network energy as it determines the circuit components in a router and their activity. The implementation of the routing and flow control and the actual router pipeline will affect the efficiency at which buffers and links are used and thus overall network throughput. In terms of reliability, faults in the router datapath will lead to errors in the transmitted

Portions reprinted, with permission, from "Express Virtual Channels: Towards the Ideal Interconnection Fabric", A. Kumar, L-S. Peh, P. Kundu and N. K. Jha, In Proceedings of International Symposium on Computer Architecture (ISCA), June 2007, ACM.

message, while errors in the control circuitry can lead to lost and mis-routed messages. The area footprint of the router is clearly highly determined by the chosen router microarchitecture and underlying circuits.

Figure 2.7 shows the microarchitecture of a state-of-the-art credit-based virtual channel (VC) router to explain how typical routers work. The example assumes a two-dimensional mesh, so that the router has five input and output ports corresponding to the four neighboring directions and the local processing element (PE) port. The major components which constitute the router are the input buffers, route computation logic, virtual channel allocator, switch allocator, and the crossbar switch. Most on-chip network routers are input-buffered, in which packets are stored in buffers only at the input ports.

Figure 2.8a shows the corresponding pipeline for this basic router (BASE). A head flit, on arriving at an input port, is first decoded and buffered according to its input VC in the buffer write (BW) pipeline stage. In the next stage, the routing logic performs route computation (RC) to determine the output port for the packet. The header then arbitrates for a VC corresponding to its output port in the

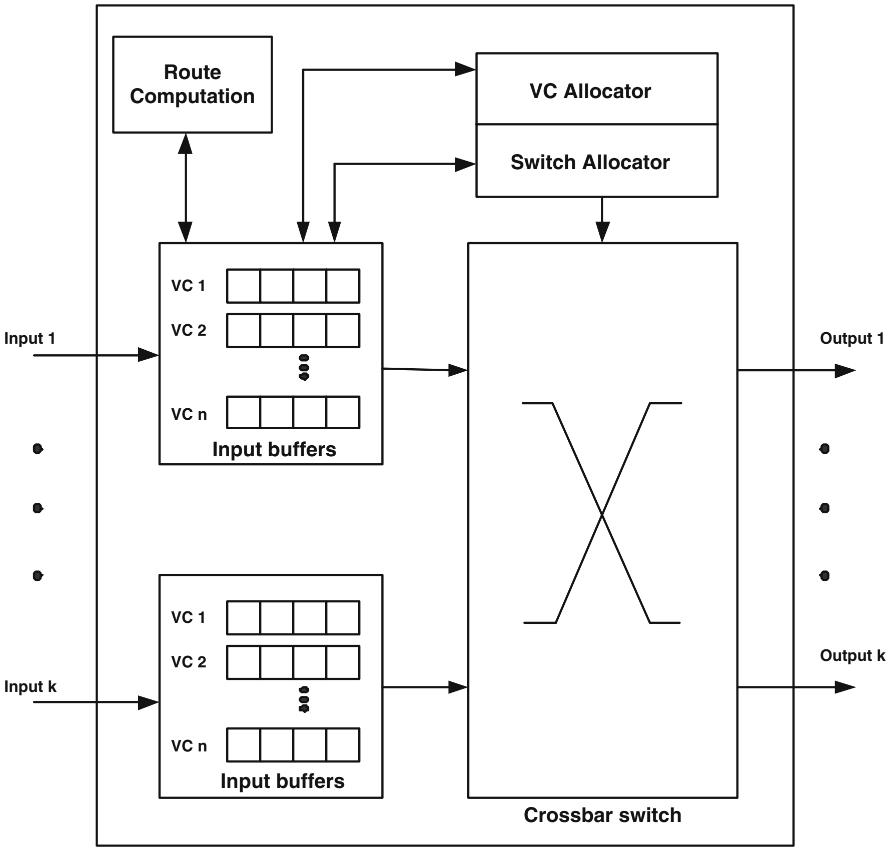


Fig. 2.7 A credit-based virtual channel router microarchitecture

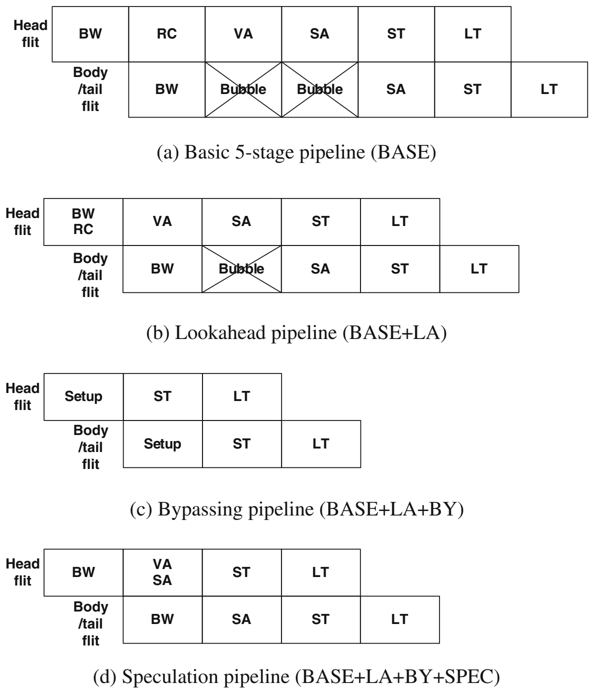


Fig. 2.8 Router pipeline [BW: Buffer Write, RC: Route Computation, VA: Virtual Channel Allocation, SA: Switch Allocation, ST: Switch Traversal, LT: Link Traversal]

VC allocation (VA) stage. Upon successful allocation of a VC, the header flit proceeds to the switch allocation (SA) stage where it arbitrates for the switch input and output ports. On winning the output port, the flit then proceeds to the switch traversal (ST) stage, where it traverses the crossbar. Finally, the flit is passed to the next node in the link traversal (LT) stage. Body and tail flits follow a similar pipeline except that they do not go through RC and VA stages, instead inheriting the VC allocated by the head flit. The tail flit, on leaving the router, deallocates the VC reserved by the header.

To remove the serialization delay due to routing, prior work has proposed *lookahead routing* [16] where the route of a packet is determined one hop in advance. Precomputing the route enables flits to compete for VCs immediately after the BW stage, with the RC stage essentially responsible for just the decoding of the route header encoded within the head flit. Figure 2.8b shows the router pipeline with lookahead routing (BASE+LA). *Pipeline bypassing* is another technique that is commonly used to further shorten the router critical path by allowing a flit to speculatively enter the ST stage if there are no flits ahead of it in the input buffer queue. Figure 2.8c shows the pipeline where a flit goes through a single stage of switch setup, during which the crossbar is set up for flit traversal in the next cycle while simultaneously allocating a free VC corresponding to the desired output port, followed by ST and LT (BASE+LA+BY). The allocation is aborted upon a port conflict. When the router ports are busy, thereby disallowing pipeline bypassing, aggressive *speculation* can be used to cut down the critical path [31, 32, 36].

Figure 2.8d shows the router pipeline, where VA and SA take place in parallel in the same cycle (BASE+LA+BY+SPEC). A flit enters the SA stage speculatively after BW and arbitrates for the switch port while at the same time trying to acquire a free VC. If the speculation succeeds, the flit directly enters the ST pipeline stage. However, when speculation fails, the flit must go through some of these pipeline stages again, depending on where the speculation failed.

The basic router pipeline is implemented using a set of microarchitectural building blocks whose designs are described below.

Router buffers: Router buffers tend to be built using SRAM or register file cells, depending on the buffer size and access timing requirements. The storage cells are then typically organized into queues for each VC, either as a physical circular buffer or a logical linked list, with head and tail pointers maintained dynamically.

Switch design: The hardware cost of crossbars in terms of area and power scale is $O((pw)^2)$, where p is the number of crossbar ports and w is the crossbar port width in bits. Different switch designs can be chosen to drive down p and/or w. Dimension slicing a crossbar [10] in a two-dimensional mesh uses two 3×3 crossbars instead of one 5×5 crossbar, with the first crossbar for traffic remaining in the X-dimension, and the second crossbar for traffic remaining in the Y-dimension. A port on the first crossbar connects with a port on the second crossbar, so traffic that turns from the X to Y dimension traverses both crossbars while those remaining within a dimension only traverses one crossbar. This is particularly suitable for the dimension-ordered routing protocol where traffic mostly stays within a dimension. Bit interleaving the crossbar (or double-pumping as explained in Sect. 2.5) targets w instead. It sends alternate bits of a link on the two phases of a clock on the same line, thus halving w.

Allocators and arbiters: The virtual-channel allocator (VA) resolves contention for output virtual channels and grants them to input virtual channels, while the switch allocator (SA) matches and grants crossbar switch ports to input virtual channels. An allocator essentially matches N requests to M resources, where the resources are VCs for the VA, and crossbar switch ports for the SA. An allocator that delivers high matching probability translates to more packets succeeding in obtaining virtual channels and passage through the crossbar switch, thereby leading to higher network throughput. Allocators must also be fast and pipelinable so that they can accommodate high clock frequencies.

Due to complexity concerns, simple separable allocators are typically used, with an allocator being composed of arbiters, where an arbiter is one that chooses one out of N requests to a single resource. Figure 2.9 shows how each stage of a 3:4 separable allocator (an allocator matching three requests to four resources) is composed of arbiters. For instance, a separable VA will have the first stage of the allocator (comprising four 3:1 arbiters) selecting one of the eligible output VCs, with the winning requests from the first stage of allocation then arbitrating for an output VC in the second stage (comprising three 4:1 arbiters). Different arbiters have been used in practice, with round-robin arbiters being the most popular due to their simplicity.

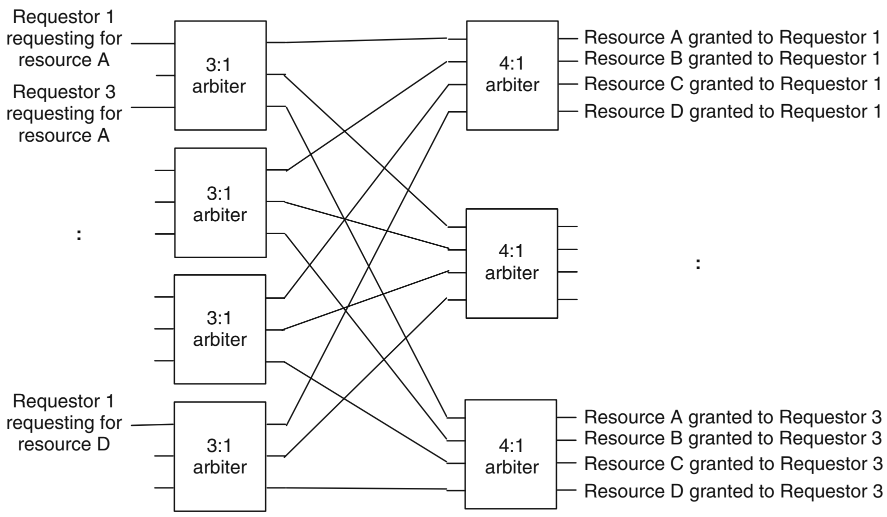

Requestor 1 requesting for resource A
Requestor 3 requesting for resource A
:
Requestor 1 requesting for resource D
3:1 arbiter
3:1 arbiter
3:1 arbiter
3:1 arbiter
4:1 arbiter
4:1 arbiter
4:1 arbiter
Resource A granted to Requestor 1
Resource B granted to Requestor 1
Resource C granted to Requestor 1
Resource D granted to Requestor 1
:
Resource A granted to Requestor 3
Resource B granted to Requestor 3
Resource C granted to Requestor 3
Resource D granted to Requestor 3

Fig. 2.9 A separable 3:4 allocator (three requestors, four resources) which consists of four 3:1 arbiters in the first stage and three 4:1 arbiters in the second. The 3:1 arbiters in the first stage decides which of the three requestors win a specific resource, while the 4:1 arbiters in the second stage ensure a requestor is granted just one of the four resources

2.2.6 Summary

In this section, we sought to present the reader with the fundamental principles of the field of on-chip networks, with a highly selective subset of basic architectural concepts chosen largely to allow the reader to appreciate the two case studies that follow. Substantial ongoing research is investigating all aspects of network architectures, including topology, routing, flow control, and router microarchitecture optimizations. For example, recent novel topologies proposed include the flattened butterfly [26], STmicroelectronics' spidergon topology [8], and Ambric's hierarchical mesh [6]. Several papers explored novel adaptive routing algorithms for on-chip networks [40, 17] and using routing to avoid thermal hotspots [39]. For flow control protocols, recent research investigated ways to bypass intermediate routers even at high loads [30,15], improving energy-delay-throughput simultaneously. Substantial work has gone into designing new router microarchitectures: decoupling arbitration thus enabling smaller crossbars within a router [27], dynamically varying the number of virtual channels depending on traffic load [33], and microarchitectural techniques that specifically target a reduction in router power [46].

2.3 Case Studies

The next two sections detail two chip prototypes of on-chip networks that are driven by vastly different design requirements. The TRIPS operand network is the principal interconnect between 25 arithmetic units, memory banks, and register banks in a distributed microprocessor. In particular, this network replaces both the operand bypass bus and the level-1 cache bus of a conventional processor. Because instruction interaction and level-1 cache delays are critical, this network is optimized for low-latency transmission with one hop per cycle. Furthermore, both sides of the network interface are integrated tightly into the pipelines of the execution units to enable fast pipeline forwarding.

In contrast, the Intel TeraFLOP chip's network connects the register files of 80 simple cores at a very high clock rate. The router in each tile uses five ports to connect to its four neighbors and the local PE, using point-to-point links that can deliver data at 20 GB/s. These links support mesochronous or phase-tolerant communication across tiles, paying a synchronization latency penalty for the benefit of a lightweight global clock distribution. An additional benefit of the low-power mesochronous clock distribution is its scalability, which enabled a high level of integration and single-chip realization of the TeraFLOPS processor. At the system level, software directly controls the routing of the messages and application designers orchestrate fine-grained message-based communication in conjunction with the computation. Table 2.1 summarizes the specific design choices of these networks for each aspect of on-chip network design, including the messaging layer, topology, routing, flow control, and router microarchitecture.

Both of these on-chip networks are quite different from interconnection networks found in large-scale multi-chip parallel systems. For example, the IBM BlueGene networks [1] housed within the multi-chassis supercomputer are faced with design requirements that differ by orders of magnitude in delay, area, power, and through-

Table 2.1 Summary of the characteristics of UT-Austin TRIPS operand network and the Intel TeraFLOPS network

	UT-Austin TRIPS OPN	Intel TeraFLOPS
Process technology	130 nm ASIC	65 nm custom
Clock frequency	400 MHz	5 GHz
Channel width	140 bits	38 bits
Interface	Explicit in instruction targets	Explicit send/receive instructions
	Interfaces with bypass paths	Interfaces with registers
	Single-flit messages	Unlimited message size
Topology	5×5 mesh	8×10 mesh
Routing	Y–X dynamic	Static source routing via software
Flow control	Control-data reservation	Two virtual channels
	On/off backpressure	On/off backpressure
Microarchitecture	Single-stage lookahead pipeline	Six-stage pipeline (including link traversal)
	Two 4-flit input buffer queues	Two 16-flit input buffer queues
	Round-robin arbiters	Separable Two-stage round-robin allocator
	Two 4×4 crossbars	5×5 double-pumped crossbar

put. The design choices in the BlueGene networks reflect this difference. First, BlueGene's network interfaces through the message passing interface (MPI) standard, whose high software overheads may not be compatible with an on-chip network with much tighter delay requirements. BlueGene's main network adopts a three-dimensional torus, which maps readily to its three-dimensional chassis form factor, but would create complexities in layout for two-dimensional on-chip networks. BlueGene's choice to support adaptive routing with a bubble escape channel and hints for source routing can lead to a complex pipeline that is hard for on-chip networks to accommodate. Its flow control protocol is virtual cut-through, with four virtual channels, each with 1 KB buffering, and its router pipeline is eight stages deep. Such a complex router whose area footprint is larger than a PowerPC 440 core and its associated double FPU unit is reasonable for a supercomputer, but not viable for on-chip networks.

2.4 UT-Austin TRIPS Operand Network

The TRIPS processor chip is an example of an emerging scalable architecture in which the design is partitioned into distributed tiles connected via control and data networks. TRIPS aims to provide a design scalable to many execution units with networks tightly integrated into the processing cores to enable instruction-level and thread-level parallelism. This section describes in detail the TRIPS operand network (OPN), which transports operands among processor tiles. Each processor's OPN is a 5×5 dynamically routed 2D mesh network with 140-bit links. The OPN connects a total of 25 distributed execution, register file, and data cache tiles. Each tile is replicated and interacts only with neighboring tiles via the OPN and other control networks. The OPN subsumes the role of several traditional microprocessor interconnect buses, including the operand bypass network, register file read and write interface, and the level-1 memory system bus. In order to serve in this function, we designed the OPN to have single-cycle per-hop latency and split control/data delivery to speed instruction wakeup at the destination. The OPN router also is small, with relatively little buffering and no virtual channels so that the effect on overall area is minimized.

2.4.1 TRIPS Overview

TRIPS is a distributed processor consisting of multiple tiles connected via several control and data networks. Figure 2.10 shows a photo micrograph of the TRIPS prototype processor chip (fabricated in a 130 nm ASIC process) and a block diagram

Portions reprinted, with permission, from "Implementation and Evaluation of a Dynamically Routed Processor Operand Network," P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. McDonald, S.W. Keckler, and D. Burger, International Symposium on Networks-on-Chips©2007, IEEE.

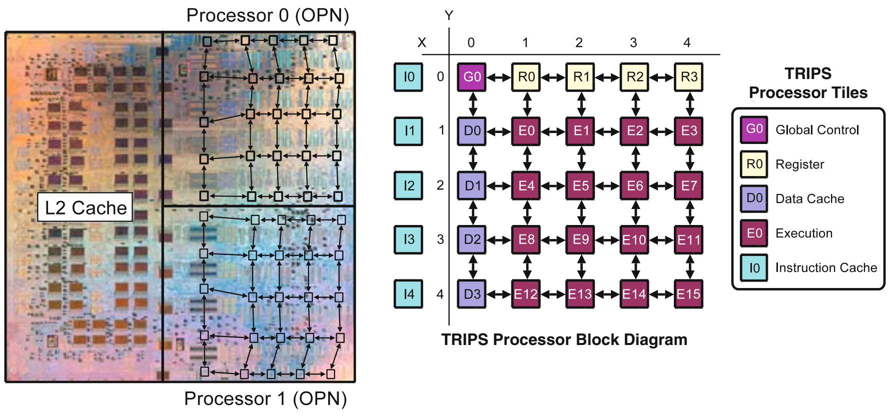


Fig. 2.10 TRIPS die photo and processor block diagram

of a TRIPS processor core. The on-chip memory system (configurable as a level-2 cache) appears on the left side of the chip micrograph and includes an on-chip network (OCN) tailored to deliver cache-lined sized items among the processors and data banks [18]. On the right side of the figure are the two processors, each with its own separate operand network, as indicated by the superimposed diagram. The right side of Fig. 2.10 shows a tile-level diagram of the processor with its OPN links. The processor contains five types of tiles: execution tiles (ET) which contain integer and floating-point ALUs and 10 KB of reservation stations; register tiles (RT) each of which contains a fraction of the processor register file; data tiles (DT) each of which contains a fraction of the level-1 data cache; instruction tiles (IT) each of which contains a fraction of the level-1 instruction cache; and a global control tile (GT) which orchestrates instruction fetch, execution, and commit. In addition, the processor contains several control networks for implementing protocols such as instruction fetch, completion, and commit in a distributed fashion. Tiles communicate directly only with their nearest neighbors to keep wires short and mitigate the effect of wire delay.

During block execution, the TRIPS operand network (OPN) has responsibility for delivering operands among the tiles. The TRIPS instruction formats contain target fields indicating to which consumer instructions a producer sends its values. At runtime, the hardware resolves those targets into coordinates to be used for network routing. An operand passed from producer to consumer on the same ET can be bypassed directly without delay, but operands passed between instructions on different tiles must traverse a portion of the OPN. The TRIPS execution model is inherently dynamic and data driven, meaning that operand arrival drives instruction execution, even if operands are delayed by unexpected or unknown memory or network latencies. Because of the data-driven nature of execution and because multiple blocks execute simultaneously, the OPN must dynamically route the operand across the processor tiles. Additional details on the TRIPS instruction set architecture and microarchitecture can be found in [5,37].

2.4.2 Operand Network Architecture

The operand network (OPN) is designed to deliver operands among the TRIPS processor tiles with minimum latency, and serves a similar purpose as the RAW scalar operand network [41]. While tight integration of the network into the processor core reduces the network interface latency, two primary aspects of the TRIPS processor architecture simplify the router design and reduce routing latency. First, because of the block execution model, reservation stations for all operand network packets are pre-allocated, guaranteeing that all OPN messages can be consumed at the targets. Second, all OPN messages are of fixed length, and consist of a single flit broken into header (control) and payload (data) phits.

The OPN is a 5×5 2D routed mesh network as shown in Fig. 2.10. Flow control is on/off based, meaning that the receiver tells the transmitter when there is enough buffer space available to send another flit. Packets are routed through the network in Y–X dimension-order with one cycle taken per hop. A packet arriving at a router is buffered in an input FIFO prior to being launched onward towards its destination. Due to dimension-order routing and the guarantee of consumption of messages, the OPN is deadlock free without requiring virtual channels. The absence of virtual channels reduces arbitration delay and makes a single-cycle router and link transmission possible.

Each operand network message consists of a control phit and a data phit. The control phit is 30 bits and encodes OPN source and destination node coordinates, along with identifiers to indicate which instruction to select and wakeup in the target ET. The data phit is 110 bits, with room for a 64-bit data operand, a 40-bit address for store operations, and 6 bits for status flags. We initially considered narrow channels and longer messages to accommodate address and data delivered by a store instruction to the data tiles. However, the single-flit message design was much simpler to implement and not prohibitively expensive given the abundance of wires.

The data phit always trails the control phit by one cycle in the network. The OPN supports different physical wires for the control and data phit; so one can think of each OPN message consisting of one flit split into a 30-bit control phit and a 110-bit data phit. Because of the distinct control and data wires, two OPN messages with the same source and destination can proceed through the network separated by a single cycle. The data phit of the first message and the control phit of the second are on the wires between the same two routers at the same time. Upon arrival at the destination tile, the data phit may bypass the input FIFO and be used directly, depending on operation readiness. This arrangement is similar to flit-reservation flow control, although here the control phit contains some payload information and does not race ahead of the data phit [35]. In all, the OPN has a peak injection bandwidth of 175 GB/s when all nodes are injecting packets every cycle at its designed frequency of 400 MHz. The network's bisection bandwidth is 70 GB/s measured horizontally or vertically across the middle of the OPN.

Figure 2.11 shows a high-level block diagram of the OPN router. The OPN router has five inputs and five outputs, one for each ordinal direction (N, S, E, and W) and one for the local tile's input and output. The ordinal directions' inputs each have two

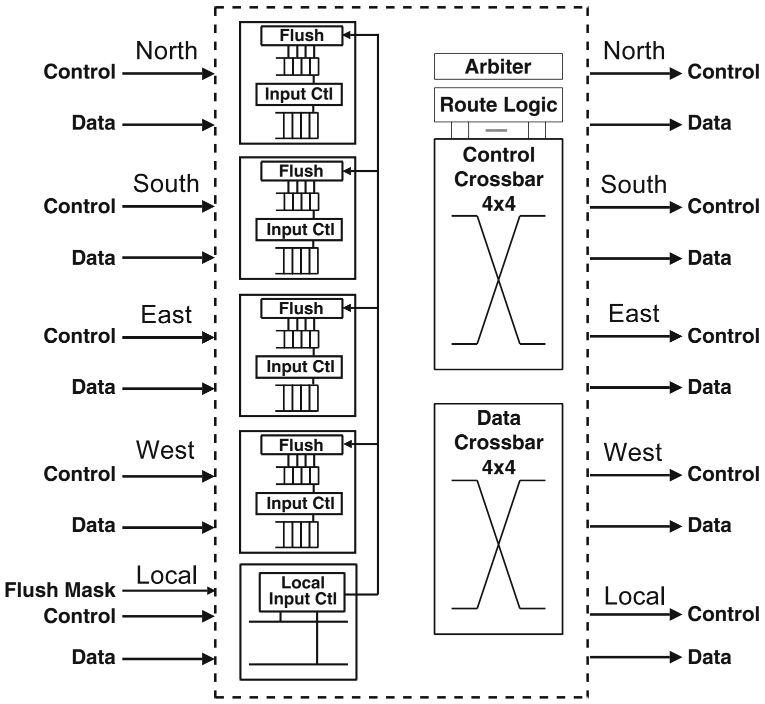


Fig. 2.11 OPN router microarchitecture

four-entry deep FIFOs, one 30 bits wide for control phits and one 110 bits wide for data phits. The local input has no FIFO buffer. The control and data phits of the OPN packet have separate 4×4 crossbars. Although the router has five input and output ports, the crossbar only needs four inputs and outputs because several input/output connections are not allowed, such as entering and exiting the router on the north port. All arbitration and routing is done on the control phit, in round-robin fashion among all incoming directions. The data phit follows one cycle behind the control phit in lock step, using the arbitration decision from its control phit.

2.4.3 OPN/Processor Integration

The key features of the TRIPS OPN that distinguish it from general purpose data transport networks are its integration into the pipelines of the ALUs and its support for speculative instruction execution.

ALU pipeline integration: Figure 2.12 shows the operand network datapath between the ALUs in two adjacent ETs. The instruction selection logic and the output latch of the ALU are both connected directly to the OPN's local input port, while the instruction wakeup logic and bypass network are both connected to the

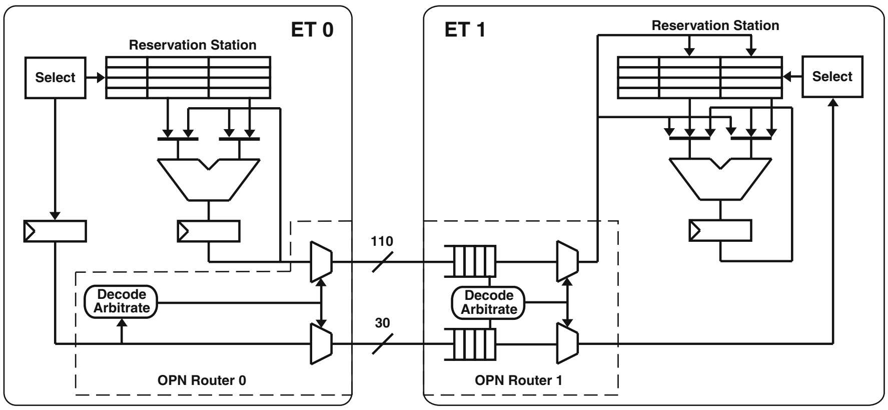


Fig. 2.12 Operand datapath between two neighboring ETs

OPN's local output. The steps below describe the use of the OPN to bypass data between two adjacent ALUs and feature the tight integration of the OPN and instruction execution. Operands transmitted greater distances travel through the network without disturbing the intervening execution unit. The timeline for message injection/reception between two adjacent ETs proceeds as follows:

- Cycle 0: Instruction wakeup/select on ET 0
  - ET0 selects a ready instruction and sends it to the ALU.
  - ET0 recognizes that the instruction target is on ET1 and creates the control phit.
- Cycle 1: Instruction execution on ET0
  - ET0 executes the instruction on the ALU.
  - ET0 delivers the control phit to router FIFO of ET1.
- Cycle 2: Instruction wakeup/select on ET1
  - ET0 delivers the data phit to ET1, bypassing the FIFO and depositing the data in a pipeline latch.
  - ET1 wakes up and selects the instruction depending on the data from ET0.
- Cycle 3: Instruction execution ET1
  - ET1 selects the data bypassed from the network and executes the instruction.

The early wakeup, implemented by delivering the control phit in advance of the data phit, overlaps instruction pipeline control with operand data delivery. This optimization reduces the remote bypass time by a cycle (to one cycle) and improves processor performance by approximately 11% relative to a design where the wakeup occurs when the data arrives. In addition, the separation of the control and data phits onto separate networks with shared arbitration and routing eliminates arbitration for the data phit and reduces network contention relative to a network that sends the header and payload on the same wires in successive cycles. This optimization is inexpensive in an on-chip network due to the high wire density.

The OPN employs round-robin arbitration among all of the inputs, including the local input. If the network is under load and chooses not to accept the control phit, the launching node captures the control phit and later the data phit in a local output buffer. The ET will stall if the instruction selected for execution needs the OPN and the ET output buffer is already full. However, an instruction that needs only to deliver its result to another instruction on the same ET does not stall due to OPN input contention. While OPN contention can delay instruction execution on the critical path of program execution, the compiler's instruction placement phase is effective at placing instructions to mitigate the distance that operands must travel and the contention they encounter.

Selective OPN message invalidation: Because the TRIPS execution model uses both instruction predication and branch prediction, some of the operand messages are actually speculative. On a branch misprediction or a block commit, the processor must flush all in-flight state for the block, including state in any of the OPN's routers. The protocol must selectively flush only those messages in the routers that belong to the flushed block. The GT starts the flush process by multicasting a flush message to all of the processor tiles using the global control network (GCN). This message starts at the GT and propagates across the GCN within 10 cycles. The GCN message contains a block mask indicating which blocks are to be flushed. Tiles that receive the GCN flush packet instruct their routers to invalidate from their FIFOs any OPN messages with block-identifiers matching the flushed block mask. As the invalidated packets reach the head of the associated FIFOs they are removed. While we chose to implement the FIFOs using shift registers to simplify invalidation, the protocol could also be implemented for circular buffer FIFOs. A collapsing FIFO that immediately eliminates flushed messages could further improve network utilization, but we found that the performance improvement did not outweigh the increased design complexity. In practice, very few messages are actually flushed.

2.4.4 Area and Timing

The TRIPS processor is manufactured using a 130 nm IBM ASIC technology and returned from the foundry in September 2006. Each OPN router occupies approximately 0.25 mm^2, which is similar in size to a 64-bit integer multiplier. Table 2.2 shows a breakdown of the area consumed by the components of an OPN router. The router FIFOs dominate the area in part because of the width and depth of the FIFOs. Each router includes a total of 2.2 kilobits of storage, implemented using standard cell flip-flops rather than generated memory or register arrays. Using shift FIFOs added some area overhead due to extra multiplexors. We considered using ASIC library-generated SRAMs instead of flip-flops, but the area overhead turned out to

Table 2.2 Area occupied by the components of an OPN router

Component	% Router Area	% E-Tile Area
Router input FIFOs	74.6(%)	7.9(%)
Router crossbar	20.3(%)	2.1(%)
Router arbiter logic	5.1(%)	0.5(%)
Total for single router	–	10.6(%)

Table 2.3 Critical path timing for OPN control and data phit

Component	Latency	(%) Path
Control phit path		
Read from instruction buffer	290 ps	13
Control phit generation	620 ps	27
ET0 router arbitration	420 ps	19
ET0 OPN output mux	90 ps	4
ET1 OPN FIFO muxing and setup time	710 ps	31
Latch setup + clock skew	200 ps	9
Total	2.26 ns	–
Data phit path		
Read from output latch	110 ps	7
Data phit generation	520 ps	32
ET0 OPN output mux	130 ps	8
ET1 router muxing/bypass	300 ps	19
ET1 operand buffer muxing/setup	360 ps	22
Latch setup + clock skew	200 ps	12
Total	1.62 ns	–

be greater given the small size of each FIFO. Custom FIFO design would be more efficient than either of these approaches.

A single OPN router takes up approximately 10% of the ET's area and all the routers together form 14% of a processor core. While this area is significant, the alternative of a broadcast bypass network across all 25 tiles would consume considerable area and not be feasible. We could have reduced router area by approximately 1/3 by sharing the FIFO entries and wires for the control and data phits. However, the improved OPN bandwidth and overall processor performance justifies the additional area.

We performed static timing analysis on the TRIPS design using Synopsys Primetime to identify and evaluate critical paths. Table 2.3 shows the delay for the different elements of the OPN control and data critical paths, matching the datapath of Fig. 2.12. We report delays using a nominal process corner, which matches the clock period of 2.5 ns obtained in the prototype chip. A significant fraction of the clock cycle time is devoted to overheads such as flip-flop read and setup times as well as clock uncertainty (skew and jitter). A custom design would likely be able to drive these overheads down. On the logic path, the control phit is much more constrained than the data phit due to router arbitration delay. We were a little surprised by the delay associated with creating the control phit, which involves decoding and encoding. This path could be improved by performing the decoding and encoding in a previous cycle and storing the control phit with the instruction before execution. We found that wire delay was small in our 130 nm process given the relatively short transmission distances. Balancing router delay and wire delay may be more challenging in future process technologies.

2.4.5 Design Optimizations

We considered a number of OPN enhancements but chose not to implement them in the prototype in order to simplify the design. One instance where performance can be improved is when an instruction must deliver its result to multiple consumers. The TRIPS ISA allows an instruction to specify up to four consumers, and in the current implementation, the same value is injected in the network once for each consumer. Multicast in the network would automatically replicate a single message in the routers at optimal bifurcation points. This capability would reduce overall network contention and latency while increasing ET execution bandwidth, as ETs would spend less time blocking for message injection. Another optimization would give network priority to those OPN messages identified to be on the program's critical path. We have also considered improving network bandwidth by having multiple, identical operand networks by replicating the routers and wires. Finally, the area and delay of our design was affected by the characteristics of the underlying ASIC library. While the trade-offs may be somewhat different with a full-custom design, our results are relevant because not all on-chip networked systems will be implemented using full-custom silicon. ASIC design for networks would also benefit from new ASIC cells, such as small but dense memory arrays and FIFOs.

2.4.6 Summary

The TRIPS Operand Network (OPN) is an effective replacement for both the global ALU bypass bus and the L1-cache bus in a tiled multicore processor. The OPN is designed for single-cycle per-hop latency and is tightly integrated into the tile pipelines to minimize network interface latency. Because the interconnection topology is exposed to software for instruction placement optimizations, the communication distances are low, averaging about two hops per message but with many nearest neighbor messages. Experiments on the prototype confirm the expectation that distributed processor performance is quite sensitive to network latency, as just one additional cycle per hop results in a 20% drop in performance. Increasing the link width in bits does not help this network since the messages already consist of only one flit. Tailoring the on-chip network to its particular use provided an efficient and high-performance design, a lesson that will carry over to multicore networks in general.

2.5 Intel TeraFLOPS Chip

The Intel TeraFLOPS processor was designed to demonstrate a monolithic on-chip network architecture capable of a sustained performance of 10^{12} floating-point operations per second (1.0 TFLOPS) while dissipating less than 100 W. The TeraFLOPS processor extends work on key network building blocks and integrates them into a 80-core on-chip network design using an effective tiled-design methodology. The 275 mm^2 prototype was designed to provide specific insights in new silicon design methodologies for large-scale on-chip networks (100+ cores), high-bandwidth interconnects, scalable clocking solutions, and effective energy management techniques. The core was designed to be small and efficient and to deliver a high level of floating-point performance. The number of cores reflects a trade-off between die size and reaching teraflop performance in a power-efficient manner.

The TeraFLOPS processor architecture contains 80 tiles (Fig. 2.13) arranged as a 8×10 2D array and connected by a mesh network that is designed to operate at 5 GHz [44]. A tile consists of a processing engine (PE) connected to a five-port router which forwards packets between tiles. The PE in each tile (Fig. 2.13) contains two independent, fully pipelined, single-precision floating-point multiply-accumulator (FPMAC) units, 3 KB of single-cycle instruction memory (IMEM), and 2 KB of data memory (DMEM). A 96-bit very long instruction word (VLIW) encodes up to eight operations per cycle. With a 10-port (6-read, 4-write) register file, the architecture allows scheduling to both FPMACs, simultaneous DMEM loads and stores, packet send/receive from mesh network, program control, synchronization primitives for data transfer between PEs, and dynamic sleep instructions.

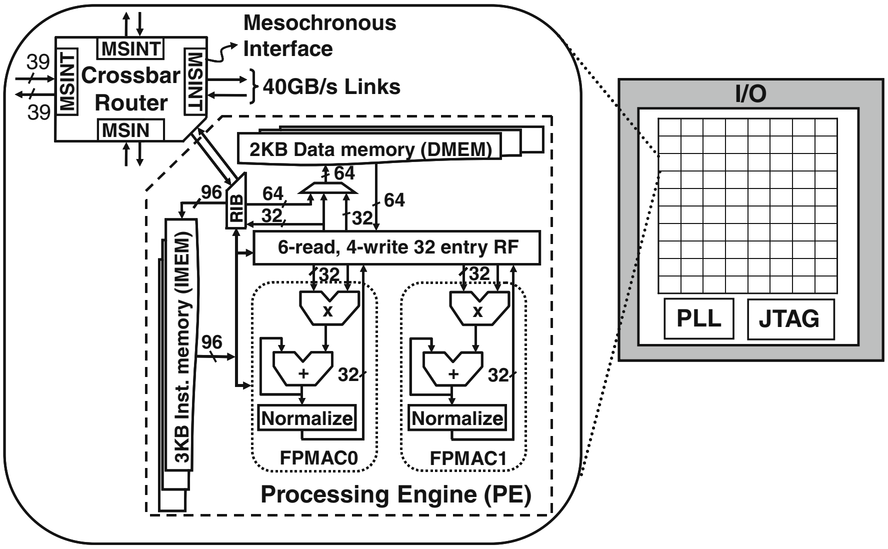


Fig. 2.13 Eighty tiles network-on-chip tile diagram and chip architecture

A router interface block (RIB) handles packet encapsulation between the PE and router. The fully symmetric architecture lets any PE send or receive instruction and data packets to or from any other PE.

The nine-stage pipelined FPMAC architecture uses a single-cycle accumulate algorithm [43], with base-32 and internal carry-save arithmetic with delayed addition. The algorithm reduces latency of dependent FPMAC instructions and enables a sustained multiply–add result (2 FLOPS) every cycle. Several optimizations allow accumulation to be implemented in just 15 fan-out-of-4 (FO4) logic stages. The dual FPMACs in each PE provide 20 GigaFLOPS of aggregate performance and are critical in achieving the goal of TeraFLOP performance.

2.5.1 On-Die Interconnect Architecture

The TeraFLOPS processor uses a 2D mesh because of the large number of cores, and operates at full core speed to provide high bisection bandwidth of 320 GB/s and low-average latency between nodes. The mesh network utilizes a five-port router based on wormhole switching and supports two identical logical lanes (better known as virtual channels), typically separated as instruction and data lanes to prevent short instruction packets from being held up by long data packets. The router in each tile uses five ports to connect to its four neighbors and the local PE, using point-to-point links that can forward data at 20 GB/s in each direction. These links support mesochronous or phase-tolerant communication across tiles, but at the cost of synchronization latency. This approach enables low-power global clock distribution which is critical in ensuring a scalable on-die communication fabric.

2.5.2 Packet Format, Routing, and Flow Control

The on-chip network architecture employs packets that are subdivided into "flits" or "Flow control unITs." Each flit contains six control bits and 32 data bits. Figure 2.14 describes the packet structure. The control bits for each flit include bits for specifying which lane to use for valid flits, and bits for specifying the start and end of a packet. The packet header (FLIT_0) enables a flexible source-directed routing scheme, where a user-specified sequence of 3-bit destination IDs (DIDs) determines the router exit port for each hop on the route. This field is updated at each hop. Each header flit supports a maximum of 10 hops. A chained header (CH) bit in the packet enables a larger number of hops. FLIT_1, that follows the header flit, specifies processing engine control information including sleep and wakeup control bits. The minimum packet size required by the protocol is two flits. The router architecture places no restriction on the maximum packet size.

The implementation choice of source-directed routing (or source routing as discussed in Sect. 2.2.3 enables the use of any of the multiple static routing schemes such as dimension-ordered routing. Throughout its static route, a packet is also

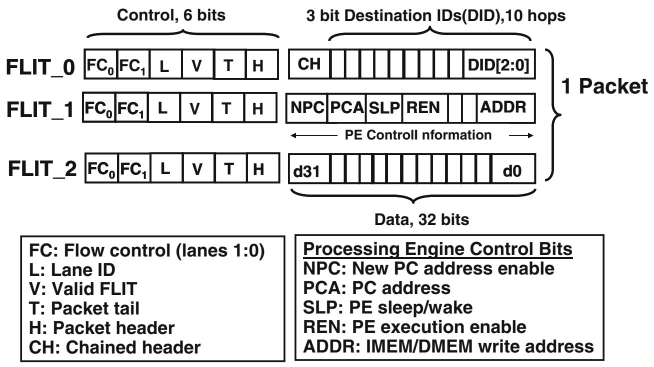


Fig. 2.14 TeraFLOPS processor on-chip network packet format and FLIT description

statically assigned to a particular lane, so virtual channels or lanes are not leveraged for flow control here, but used to avoid system-level deadlocks. Such static assignments result in simple routing and flow control logic, enabling a high frequency implementation.

Flow control and buffer management between routers utilizes an on/off scheme using almost-full signals, which the downstream router signals via two flow control bits (FC0–FC1), one for each lane, when its buffer occupancy reaches a specified threshold. The buffer almost-full threshold can be programmed via software.

2.5.3 Router Microarchitecture

A five-port two-lane pipelined packet-switched router core (Fig. 2.15) with phase-tolerant mesochronous links forms the key communication block for the 80-tile on-chip network architecture. Each port has two 39-bit unidirectional point-to-point links implementing a mesochronous interface (MSINT) with first-in, first-out (FIFO) based synchronization at the receiver. The 32 data bits enable transfer of a single-precision word in a single flit. The router uses two logical lanes (lane 0-1) and a fully non-blocking crossbar switch with a total bandwidth of 100 GB/s (32 bits $\times$ 5 GHz $\times$ 5 ports). Each lane has a 16 flit queue, arbiter, and flow control logic. A shared datapath allows crossbar switch re-use across both lanes on a per-flit basis.

The router uses a five-stage pipeline with input buffering, route computation, switch arbitration, and switch traversal stages. Figure 2.16 shows the data and control pipelines, including key signals used in the router design. The shaded portion between stages four and five represents the physical crossbar switch. To ensure 5 GHz operation, the router pipeline employs a maximum of 15 FO4 logic levels between pipe stages. The fall-through latency through the router is thus 1 ns at 5 GHz operation. Control overhead for switch arbitration can result in two additional cycles of latency, one cycle for port arbitration and one cycle for lane arbitration. Data flow through the router is as follows: (1) The synchronized input data

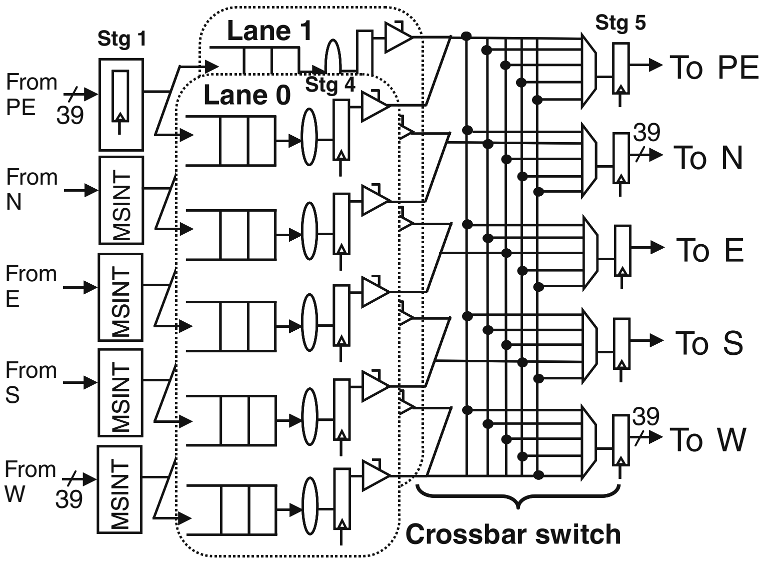
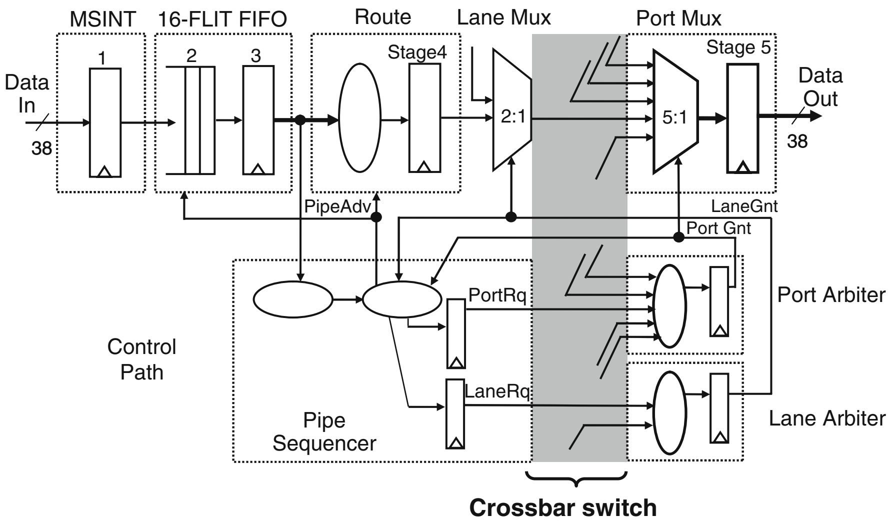


Fig. 2.15 Five-port two-lane shared crossbar router architecture



Synchronize in-coming packet	**Buffer Write**	**Buffer Read**	**Route Compute**	**Port & Lane Arbitration**	**Switch Traversal**	**Link Traversal**

Fig. 2.16 Router data and control pipeline diagram and pipelined routing of a packet

is first buffered into the queues. (2) Data coming out of the queues is examined by routing and sequencing logic to generate output binding requests based on packet header information. (3) A separable, two-stage, round-robin arbitration scheme is used, with arbiters replicated at each port for speed. (4) With five ports and two lanes, the crossbar switch requires a 10-to-1 arbiter and is implemented as a 5-to-1 port arbiter followed by a 2-to-1 lane arbiter. The first stage of arbitration within a particular lane essentially binds one input port to an output port, for the entire duration of the packet. If packets in different lanes are contending for the same physical output resource, then a second stage of arbitration occurs at a flit level granularity.

2.5.4 Interconnect Design Details

Several circuit and design techniques were employed to achieve the objective of high performance, low power, and short design cycle. By following a tiled methodology where each tile is completely self-contained, including C4 bumps, power, and global clock routing, all tiles could be seamlessly arrayed at the top level simply by abutment. This methodology enabled rapid turnaround of the fully custom designed processor. To enable 5 GHz operation, the entire core uses hand-optimized data path macros. For quick turnaround, CMOS static gates are used to implement most of the logic. However, critical registers in the FPMAC and at the router crossbar output utilize implicit-pulsed semi-dynamic flip-flops (SDFF) [44].

2.5.5 Double-Pumped Crossbar

Crossbar switch area increases as a square function O(n^2) of the total number of I/O ports and number of bits per port. A large part of the router area is dominated by the crossbar, in some cases as much as 53% of the total [47]. In this design, the crossbar data buses are double-pumped [45], also known as bit-interleaved, reducing the crossbar hardware cost by half. A full clock cycle is allocated for communication between stages 4 and 5 of the router pipeline. The schematic in Fig. 2.17 shows the 36-bit crossbar data bus double-pumped at the fourth pipe-stage of the router by interleaving alternate data bits using dual edge-triggered flip-flops, reducing crossbar area by 50%. One clock phase is allocated for data propagation across the crossbar channel. The master latch M0 for data bit zero ($i0$) is combined with the slave latch S1 for data bit one ($i1$) and so on. The slave latch S0 is placed halfway across the crossbar channel. A 2:1 multiplexer, controlled by clock, is used to select between the latched output data on each clock phase. At the receiver, the double-pumped data is first latched by slave latch (S0), which retains bit corresponding to $i0$ prior to capture by pipe-stage 5. Data bit corresponding to $i1$ is easily extracted using a rising edge flip-flop.

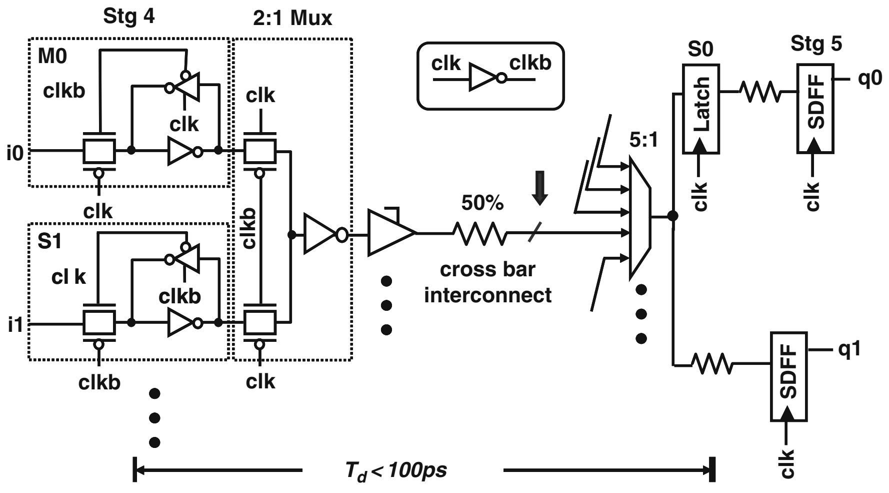


Fig. 2.17 Double-pumped crossbar switch schematic

2.5.6 Mesochronous Communication

The mesochronous interface (Fig. 2.18) allows for efficient, low-power global clock distribution as well as clock-phase-insensitive communication across tiles. Four of the five 2 mm long router links are source synchronous, each providing a strobe (Tx_clk) with 38 bits of data. To reduce active power, Tx_clk is driven at half the clock rate. A four deep circular FIFO built using transparent latches captures data on

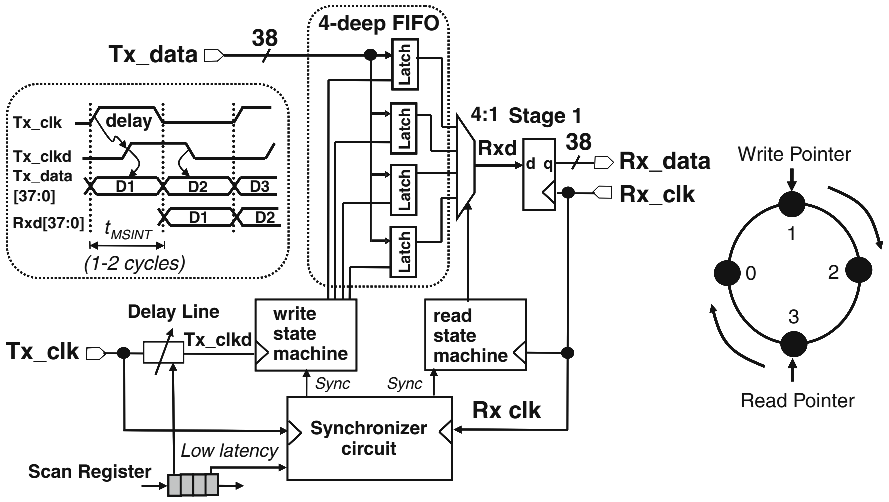


Fig. 2.18 Phase-tolerant mesochronous interface and timing diagram

both edges of the delayed link strobe at the receiver. The strobe delay and duty-cycle can be digitally programmed using the on-chip scan chain. A synchronizer circuit sets the latency between the FIFO write and read pointers to one or two cycles at each port, depending on the phase of the arriving strobe with respect to the local clock. A more aggressive low-latency setting reduces the synchronization penalty by one cycle.

2.5.7 Fine-Grain Power Management

Fine-grain clock gating and sleep transistor circuits [42] are used to reduce active and standby leakage power, which are controlled at full-chip, tile-slice, and individual tile levels based on workload. About 90% of FPMAC logic and 74% of each PE is sleep-enabled. The design uses NMOS sleep transistors to reduce frequency penalty and area overhead. The average sleep transistor area overhead (of the 21 sleep regions in each tile) is 5.4% of the fub area where it is placed. Each tile is partitioned into 21 smaller sleep regions with dynamic control of individual blocks based on instruction type. For instance, each FPMAC can be controlled through NAP/WAKE instructions. The router is partitioned into 10 smaller sleep regions with control of individual router ports (through software) (Fig. 2.19), depending on network traffic patterns. Enable signals gate the clock to each port, MSINT, and the links. In addition, the enable signals also activate the NMOS sleep transistors in the input queue arrays of both lanes. In other words, stages 1–5 of the entire port logic are clock gated, while within each lane, only the FIFOs can be independently power-gated (to reduce leakage power), based on port inactivity. The 360 μm NMOS sleep device in the queue is sized via circuit simulations to provide single-cycle wakeup and a 4.3× reduction in array leakage power with a 4% frequency

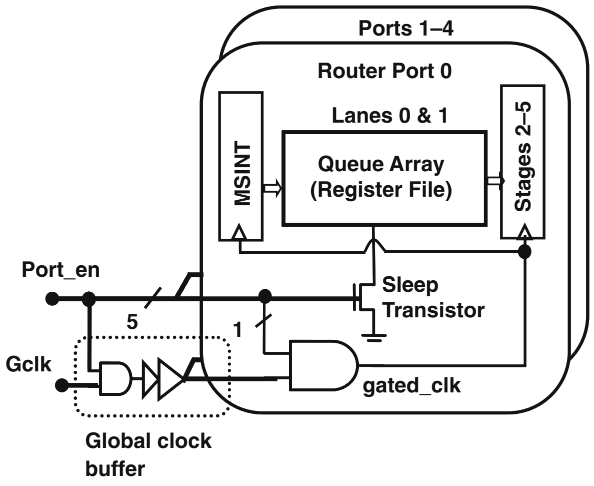


Fig. 2.19 Router and on-die network power management

impact. The global clock buffer feeding the router is finally gated at the tile level based on port activity.

2.5.8 Experimental Results

The TeraFLOPS processor is fabricated in a 65 nm process technology with eight layers of copper interconnect [2]. The 275 mm^2 fully custom design contains 100 million transistors. Each 3 mm^2 tile contains 1.2 million transistors with the processing engine accounting for 1 million (83%) and the router 17% of the total tile device count. The functional blocks of the tile and router are identified in the die photographs in Fig. 2.20. The tiled nature of the physical implementation is easily visible. The five router ports within each lane are stacked vertically, and the two lanes sandwich the crossbar switch. Note the locations of the inbound FIFOs and their relative area compared to the routing channel in the crossbar.

Silicon chip measurements at 80°C demonstrate that the chip achieves 1.0TFLOPS of average performance at 4.27 GHz and 1.07 V supply with a total power dissipation of just 97 W. Silicon data confirms that the 2D mesh operates at a maximum frequency of 5.1 GHz at 1.2 V. Corresponding measured on-chip network power per-tile with all router ports active is 924 mW. This number includes power dissipated in the router, MSINT, and links. Power reduction due to selective router port activation with combined clock gating and sleep transistor usage is given in Fig. 2.21a. Measured at 1.2 V, 80°C, and 5.1 GHz operation, the total network power per-tile can be lowered from a maximum of 924 mW with all router ports active to 126 mW, resulting in a 7.3× or 86.6% reduction. The 126 mW number represents the network leakage power per-tile with all ports and global router clock buffers disabled and accounts for 13.4% of total network power.

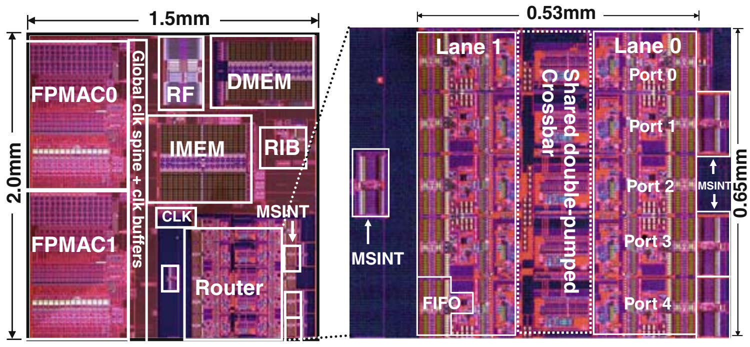


Fig. 2.20 Individual tile and five-port two-lane shared crossbar router layout in 65 nm CMOS technology. The mesochronous interfaces (MSINT) on four of the five ports are also shown

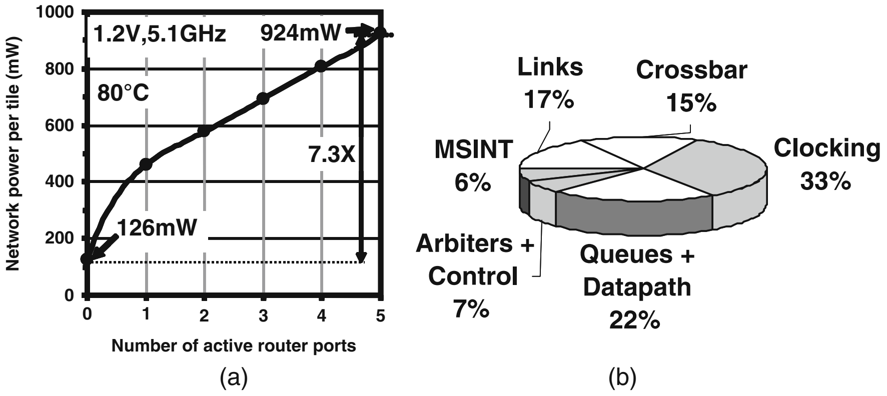


Fig. 2.21 (**a**) Measured on-die network power reduction benefit. (**b**) Estimated on-die communication power breakdown at 4 GHz, 1.2 V, and 110°C

The pie-chart in Fig. 2.21b shows the estimated on-die network power breakdown obtained at 4 GHz, 1.2 V supply, and at 110°C. At the tile level, the communication power, which includes the router, MSINT, and links is significant and dissipates 28% of the total tile power. Clocking power is the largest component due to the deep pipelines used and accounts for 33% of the total network power. The input queues on both lanes and datapath circuits is the second major component dissipating 22% of the communication power. The MSINT blocks result in 6% power overhead.

2.5.9 Summary

This section presented an 80-tile high-performance multicore processor implemented in a 65 nm process technology. The prototype integrates 160 low-latency FPMAC cores and features a single-cycle accumulator architecture for high throughput. Each tile contains a fast and compact router operating at core speed where the 80 tiles are interconnected using a 2D mesh topology providing a high bisection bandwidth of over 2.5 Tera-bits/s. When operating at 5 GHz, the router combines 100 GB/s of raw bandwidth with a per-hop fall-through latency of 1 ns. The design uses co-design of micro architecture, logic, circuits, and a 65 nm process to reach target performance. The proposed mesochronous clocking scheme provides a scalable and low-power alternative to conventional synchronous clocking. The fine-grained on-chip network power management features, controllable on a router per-port basis, allow on-demand performance while reducing active and stand-by leakage power. The processor delivers TeraFLOPS performance while consuming less than 100 W.

2.6 Conclusions

Research into on-chip networks emerged only in the past decade and substantial challenges clearly remain. To conclude this chapter, we discuss fundamental challenges for the design of on-chip networks and how developments in the underlying technology drivers and overarching multicore chip architectures can significantly affect the requirements for future on-chip networks.

2.6.1 Fundamental Challenges

Throughout this chapter, we have highlighted how on-chip network design faces immense pressure from tight design requirements and constraints: the need to deliver high bandwidth while satisfying stringent constraints in delay, area, power, and reliability. Techniques that extend bandwidth usually add complexity and thus area and power overhead, making on-chip network design difficult in the face of such tight constraints. Recent work has characterized the substantial energy-delay-throughput gap that exists between today's state-of-the-art on-chip network design and the ideal interconnect fabric of dedicated wires directly connecting every pair of cores [30]. Substantial innovations are needed in all aspects of on-chip network architecture (topology, routing, flow control, microarchitecture) to narrow this gap, as on-chip networks stand as a major stumbling block to realizing many-core chips.

2.6.2 Underlying Technology Drivers

While research in addressing the fundamental requirements facing on-chip network architectures is critically needed, one should keep in mind that innovations in underlying technology can further alter the landscape of on-chip network architectures. While current on-chip networks tend to utilize full-swing repeated interconnects, recent developments in interconnect circuits such as those tackling low-swing global signaling [19, 22, 25] can significantly alter current trade-offs and prompt new network architectures. Other more disruptive interconnect technologies for on-chip communications include optics [28, 38] and RF interconnects [7], which will significantly change the demands on the on-chip network architecture. Further, developments in off-chip I/O technologies, such as 3-D chip integration [4] and capacitive I/Os [20], can again have a large effect on on-chip network design, as the demand for on-chip network bandwidth can grow by orders of magnitude.

2.6.3 Overarching Multi-core Chip Architectures

While we have covered the fundamentals of on-chip networks in this chapter, the range of multicore architectures demanding such networks is immense. The debate over the granularity of the cores (large versus small) in multicore systems is far from settled. While many multicore systems are largely homogeneous, opportunities abound for chip-based systems composed of different types of cores with different communication demands. While some of today's systems employ shared memory with cache-to-cache transactions carried on the network, others employ a message-passing approach. Different types of systems will likely support different communication paradigms, resulting in different communication substrates. Different points on this large multicore chip design space require different on-chip networks, further challenging the on-chip network architect.

2.6.4 Summary

In this chapter, we introduced readers to the fundamentals of on-chip networks for multicore systems and presented case studies of two recent on-chip network implementations. Because of their different purposes, these two networks differed across many design aspects, indicating a strong need to customize on-chip network design. The field of on-chip networks is in its infancy providing ripe opportunities for research innovations. As we are only able to briefly motivate and discuss the challenges here, we refer interested readers to a recent article that summarizes the key findings of a workshop initiated by NSF in December 2006 to chart the research challenges in on-chip networks [34].

Acknowledgements Dr. Peh wishes to thank her entire Princeton research group, as well as students who have taken the ELE580 graduate course on interconnection networks as those research and teaching experiences helped significantly in the writing of this chapter. Her research has been kindly supported by the National Science Foundation, Intel Corporation, and the MARCO Gigascale Systems Research Center. Dr. Keckler thanks the entire TRIPS team, in particular Doug Burger, Paul Gratz, Heather Hanson, Robert McDonald, and Karthikeyan Sankarlingam, for their contributions to the design and implementation of the TRIPS operand network. The TRIPS project was supported by the Defense Advanced Research Projects Agency under contract F33615-01-C-4106 and by NSF CISE Research Infrastructure grant EIA-0303609. Dr. Vangal thanks the entire TeraFLOPS processor design team at Circuit Research Laboratories, Intel Corporation, for flawless execution of the design.

References

1. N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas. Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 49(2/3):265–276, 2005.
2. P. Bai, C. Auth, S. Balakrishnan, M. Bost, R. Brain, V. Chikarmane, R. Heussner, M. Hussein, J. Hwang, D. Ingerly, R. James, J. Jeong, C. Kenyon, E. Lee, S.-H. Lee, N. Lindert, M. Liu, Z. Ma, T. Marieb, A. Murthy, R. Nagisetty, S. Natarajan, J. Neirynck, A. Ott, C. Parker, J. Sebastian, R. Shaheed, S. Sivakumar, J. Steigerwald, S. Tyagi, C. Weber, B. Woolery, A.Yeoh, K. Zhang, and M. Bohr. A 65 nm Logic Technology Featuring 35 nm Gate Lengths, Enhanced Channel Strain, 8 Cu Interconnect Layers, Low-k ILD and 0.57 um^2 SRAM Cell. In *International Electron Devices Meeting (IEDM)*, pages 657–660, Dec 2004.

3. S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, and M. Reif. TILE64 processor: A 64-core SoC with mesh interconnect. In *International Solid State Circuits Conference*, Feb 2008.
4. S. Borkar. Thousand core chips: a technology perspective. In *Design Automation Conference*, pages 746–749, June 2007.
5. D. Burger, S. Keckler, K. McKinley, M. Dahlin, L. John, C. Lin, C. Moore, J. Burrill, R. McDonald, and W. Yoder. Scaling to the End of Silicon with EDGE Architectures. *IEEE Computer*, 37(7):44–55, July 2004.
6. M. Butts. Synchronization through communication in a massively parallel processor array. *IEEE Micro*, 27(5):32–40, Sep/Oct 2007.
7. M. F. Chang, J. Cong, A. Kaplan, M. Naik, G. Reinman, E. Socher, and S. Tam. CMP network-on-chip overlaid with multi-band RF-interconnect. In *International Conference on High-Performance Computer Architecture*, Feb 2008.
8. M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra. Spidergon: a novel on-chip communication network. In *International Symposium on System-on-Chip*, page 15, Nov 2004.
9. W. J. Dally. Virtual-channel flow control. In *International Symposium of Computer Architecture*, pages 60–68, May 1990.
10. W. J. Dally, A. Chien, S. Fiske, W. Horwat, R. Lethin, M. Noakes, P. Nuth, E. Spertus, D. Wallach, D. S. Wills, A. Chang, and J. Keen. Retrospective: the J-machine. In *25 years of the International Symposium on Computer Architecture (selected papers)*, pages 54–58, 1998.
11. W. J. Dally and C. L. Seitz. The torus routing chip. *Journal of Distributed Computing*, 1:187–196, 1986.
12. W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, San Francisco, CA, 2004.
13. J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks*. Morgan Kaufmann Publishers, San Francisco, CA, 2003.
14. W. Eatherton. The push of network processing to the top of the pyramid. Keynote speech, International Symposium on Architectures for Networking and Communications Systems.
15. N. Enright-Jerger, L.-S. Peh, and M. Lipasti. Circuit-switched coherence. In *International Symposium on Networks-on-Chip*, April 2008.
16. M. Galles. Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip. In *Hot Interconnects 4*, Aug 1996.
17. P. Gratz, B. Grot, and S. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *International Conference on High-Performance Computer Architecture*, pages 203–214, Feb 2008.
18. P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger. Implementation and Evaluation of On-Chip Network Architectures. In *International Conference on Computer Design*, pages 477–484, Sep 2006.
19. R. Ho, K. Mai, and M. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4), Apr 2001.
20. D. Hopkins, A. Chow, R. Bosnyak, B. Coates, J. Ebergen, S. Fairbanks, J. Gainsley, R. Ho, J. Lexau, F. Liu, T. Ono, J. Schauer, I. Sutherland, and R. Drost. Circuit techniques to enable 430 GB/s/mm^2 proximity communication. *International Solid-State Circuits Conference*, pages 368–609, Feb 2007.
21. Infiniband trade organization. http://www.infinibandta.org/
22. A. P. Jose, G. Patounakis, and K. L. Shepard. Pulsed current-mode signaling for nearly speed-of-light intrachip communication. *Proceedings of the IEEE*, 41(4):772–780, April 2006.
23. J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. *IBM Journal of Research and Development*, 49(4/5):589–604, 2005.
24. P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

25. B. Kim and V. Stojanovic. Equalized interconnects for on-chip networks: Modeling and optimization framework. In *International Conference on Computer-Aided Design*, pages 552–559, November 2007.
26. J. Kim, J. Balfour, and W. J. Dally. Flattened butterfly topology for on-chip networks. In *International Symposium on Microarchitecture*, pages 172–182, December 2007.
27. J. Kim, C. A. Nicopoulos, D. Park, N. Vijaykrishnan, M. S. Yousif, and C. R. Das. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *International Symposium on Computer Architecture*, pages 4–15, June 2006.
28. N. Kirman, M. Kirman, R. K. Dokania, J. F. Martinez, A. B. Apsel, M. A. Watkins, and D. H. Albonesi. Leveraging optical technology in future bus-based chip multiprocessors. In *International Symposium on Microarchitecture*, pages 492–503, December 2006.
29. P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, March/April 2005.
30. A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha. Express virtual channels: Towards the ideal interconnection fabric. In *International Symposium on Computer Architecture*, pages 150–161, June 2007.
31. S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 network architecture. *IEEE Micro*, 22(1):26–35, Jan/Feb 2002.
32. R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *International Symposium on Computer Architecture*, pages 188–197, June 2004.
33. C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das. ViChaR: A dynamic virtual channel regulator for network-on-chip routers. In *International Symposium on Microarchitecture*, pages 333–346, December 2006.
34. J. D. Owens, W. J. Dally, R. Ho, D. N. J. Jayasimha, S. W. Keckler, and L.-S. Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108, Sep/Oct 2007.
35. L.-S. Peh and W. J. Dally. Flit-reservation flow control. In *International Symposium on High-Performance Computer Architecture*, pages 73–84, Jan 2000.
36. L.-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. In *International Conference on High-Performance Computer Architecture*, pages 255–266, January 2001.
37. K. Sankaralingam, R. Nagarajan, P. Gratz, R. Desikan, D. Gulati, H. Hanson, C. Kim, H. Liu, N. Ranganathan, S. Sethumadhavan, S. Sharif, P. Shivakumar, W. Yoder, R. McDonald, S. Keckler, and D. Burger. The Distributed Microarchitecture of the TRIPS Prototype Processor. In *International Symposium on Microarchitecture*, pages 480–491, December 2006.
38. A. Shacham, K. Bergman, and L. P. Carloni. The case for low-power photonic networks on chip. In *Design Automation Conference*, pages 132–135, June 2007.
39. L. Shang, L.-S. Peh, A. Kumar, and N. K. Jha. Thermal modeling, characterization and management of on-chip networks. In *International Symposium on Microarchitecture*, pages 67–78, Decemeber 2004.
40. A. Singh, W. J. Dally, A. K. Gupta, and B. Towles. Goal: a load-balanced adaptive routing algorithm for torus networks. In *International Symposium on Computer Architecture*, pages 194–205, June 2003.
41. M. B. Taylor, W. Lee, S. P. Amarasinghe, and A. Agarwal. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architecture. In *International Symposium on High-Performance Computer Architecture*, pages 341–353, Feb 2003.
42. J. Tschanz, S. Narendra, Y. Ye, B. Bloechel, S. Borkar, and V. De. Dynamic Sleep Transistor and Body Bias for Active Leakage Power Control of Microprocessors. *IEEE Journal of Solid-State Circuits*, 38(11):1838–1845, Nov 2003.
43. S. Vangal, Y. Hoskote, N. Borkar, and A. Alvandpour. A 6.2-GFLOPS Floating-Point Multiply-Accumulator with Conditional Normalization. *IEEE Journal of Solid-State Circuits*, 41(10):2314–2323, Oct 2006.
44. S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-Tile Sub-100 W TeraFLOPS Processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, Jan 2008.

45. S. Vangal, A. Singh, J. Howard, S. Dighe, N. Borkar, and A. Alvandpour. A 5.1 GHz 0.34 mm^2 Router for Network-on-Chip Applications. In *International Symposium on VLSI Circuits*, pages 42–43, June 2007.
46. H.-S. Wang, L.-S. Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. In *International Symposium on Microarchitecture*, pages 105–116, Nov 2003.
47. H. Wilson and M. Haycock. A Six-port 30-GB/s Non-blocking Router Component Using Point-to-Point Simultaneous Bidirectional Signaling for High-bandwidth Interconnects. *IEEE Journal of Solid-State Circuits*, 36(12):1954–1963, Dec 2001.