Trends Toward On-Chip Networked Microsystems

Timothy Mark Pinkston *(Senior Member, IEEE)* and Jeonghee Shin
SMART Interconnects Group
Electrical Engineering–Systems Department
University of Southern California
Los Angeles, CA 90089-2562, USA
E-mail: {tpink, jeonghee}@charity.usc.edu

***Abstract*— This survey paper identifies some trends in the application, implementation technology, and processor architecture areas. A taxonomy which captures the influence of these trends on processor *microsystems* is presented, and the communication needs of various classes of these architectures is also briefly surveyed. We observe a trend toward on-chip networked microsystems derived from logically and physically partitioning the processor architecture. Partitioning the architecture logically enables the parallelism offered by growing application workloads to be well exploited. Partitioning the architecture physically enables the scaling properties of the underlying implementation technology to continue to provide increasing performance and not be encumbered by chip-crossing wire delay, which no longer is negligible. The impact on future research directions of this paradigm shift in the way microsystems are designed and intraconnected is briefly highlighted.**

***Index Terms*— interconnection network, microprocessor, microsystem, on-chip network, partitioned architecture, system-on-chip.**

I. Introduction

THE 2003 International Technology Roadmap for Semiconductors (ITRS'03)[1] projects that within 10 years (by 2013), high-volume processor chips will have approximately 1.5 *billion* transistors per cm^2 running at over 20 GHz. Achieving this milestone would signify an inevitable paradigm shift in parallel computing away from the current notion of "macro" systems consisting of multiple (and possibly heterogeneous) chips toward the notion of highly parallel and integrated "micro" systems implemented within single chips, otherwise known as systems-on-chips (SoCs). Although SoCs can implement microsystems consisting of a wide variety of intellectual property (IP) cores, we restrict ourselves to those which implement general-purpose processor microsystems.

Microprocessor designers are faced with the challenge of building integrated systems which fully utilize abundant transistor and wiring resources while operating at increased clock frequencies. At the same time, they are dealing with the concomitant increase in on-chip communication requirements and bottlenecks. Wiring delay, failures, and overall design/verification complexity among other things (including power consumption) are causing designers to re-think the architecture of these so-called microsystems, especially the communication subsystem. The extent to which techniques developed for the communication subsystem within traditional macrosystems can be directly applied to the interconnect problem in emerging microsystems is not readily known. What's more, the unique solutions needed to address certain microsystem-specific interconnect problems have yet to be discovered.

This work was supported in part by NSF grants CCR-0209234 and CCR-0311742, and by WiSE funding at USC.

In this survey paper, we highlight some of the critical issues involved in designing billion transistor microsystems and focus on the communication subsystem: the on-chip interconnection network. We first consider some trends in the areas of applications, implementation technology, and architecture in Section II. Then, in Section III, we survey emerging processor microsystem architectures and discuss their communication needs. We introduce a new classification scheme for processor microsystems based on the way the architecture is partitioned, and we identify some common attributes of their on-chip communication subsystem. This is followed by Section IV in which possible future research directions in the area of on-chip networks for microsystems are presented. Finally, Section V concludes the paper.

II. Trends in Microsystem Design

To gain a deeper understanding of microsystem design issues and trends, it is useful to revisit the basic interrelationships between applications, architecture, and implementation technology in this context. Applications place demands on the processor architecture and implementation technology to deliver performance by defining *what* system functions should be supported by the hardware and software. The processor architecture defines *how* system functions are supported in both hardware and software (i.e., the compiler and programming model). The implementation technology determines the *extent* to which system functions are supported in hardware—in the context of microsystems, within the real estate of a single chip. The capabilities and limitations of an architecture and those of the technology in realizing the architecture ultimately influence the achievable application performance. By observing various trends in these three areas, likely directions for future microsystem design can be revealed.

A. Application Trends

Commercial applications are becoming more and more data-centric, oriented towards needing peak aggregate throughput for executing multiple problems simultaneously rather than

requiring peak response time for executing any single problem. Although response time is important for many technical workloads, peak throughput in teraflops or petaflops has for some time been a convenient measure for performance, e.g., in workloads encompassing the scientific-engineering domain such as bio/molecular processing, human genome, pharmaceutical design (protein folding), weather/event forecasting, computational fluid dynamics, 3D plasma modeling, and other simulation and grand challenge applications. However, as commercial applications represent the largest and fastest growing market segment for high-performance computing [2], systems that achieve peak aggregate throughput will likely dominate.

Throughput-oriented commercial workloads include database, on-line transaction/financial processing, network processing, data (audio/video) streaming, rendering, and various other web and data center applications. As discussed in [2], [3], these applications are typified as having large instruction and data footprints which increase communication costs, i.e., miss rates at various levels in the memory hiearchy. They also are largely integer intensive and highly data-dependent, with little exploitable instruction-level parallelism (ILP). More importantly, they are trivially parallelizable into "thread" or "process" logical work units. Such parallelism exists in applications when there is some degree of independence between sequences of instructions such that execution of those instruction sequences can be overlapped in time and/or space. The granularity of those instruction sequences or threads can be as fine as basic blocks of instructions delimited by branches in control flow to as coarse as traces of instructions spanning many branches or even the entire program.

Commercial applications, therefore, would benefit greatly from architectures capable of exploiting modest amounts of ILP and significant amounts of thread-level parallelism (TLP) (or process-level parallelism (PLP)). In database applications, for example, TLP can be used to hide communication latency as relatively independent transactions or queries can be initiated in the same time frame by different clients in response to cache/memory misses and disk I/O requests. ILP alone cannot hide this latency as effectively.

B. Implementation Technology Trends

On the implementation side, device scaling and chip sizing continue to allow an exponential growth rate in the number of transistors and wires that can be integrated per unit area on a CMOS chip.[1] Obvious advantages are that more system functions required by applications can be implemented in on-chip hardware, and local switching speeds (i.e., clock frequencies) spanning a fixed number of densely packed gates increases in proportion to technology scaling. These tendencies as projected by ITRS-2003 are shown in Figure 1. However, several challenges also arise, many of which have been well documented in recent literature—namely, global and semi-global wire delay, design and verification complexity, fault containment/tolerance, and power consumption [4]. All but power consumption are touched upon below.[2]

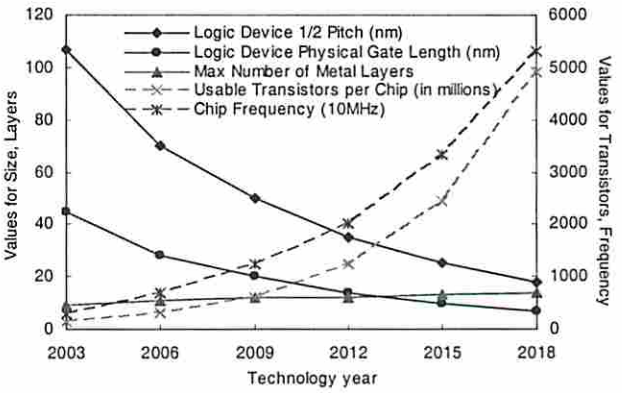


Fig. 1. ITRS-2003 Technology Projections [1].

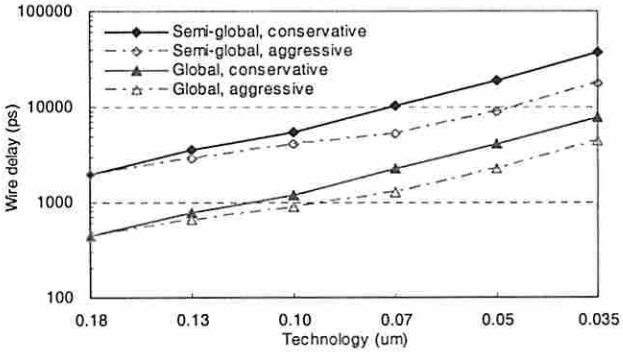


Fig. 2. Unrepeated wire delay (picoseconds) spanning 10 mm distance [4].

Computation speed is no longer limited by device switching time—which decreases with scaling—but by non-local wiring delay. This remains true even with the use of highly conductive copper wires insulated by low-k dielectric material to reduce electrical "crosstalk." Copper is about 50% more conductive than aluminum but only about 40% more conductive when walled to keep from diffusing into surrounding oxide. However, like any other conductor, it still suffers from quadratic delay growth with increasing interconnect distance. Using the delay models provided in [4], Figure 2 shows this quadratic wire delay growth as a range (conservative to aggressive assumptions) for several process generations. Here, for a fixed-length 10 mm long wire, the interconnect distance relative to each generation's shrinking gate length increases. That is, the interconnect distance spans proportionally more gates for each process shrink. Semiglobal wires are assumed to have a pitch of 8λ (i.e., middle metal layers 3 and 4) and global wires are assumed to have a pitch of 16λ (i.e., top metal layers 5 and 6), where λ is a technology-independent parameter that represents half the drawn gate length. As summarized in Table I, a chip-crossing global wire signal can reach only about .4% of the

[1] In this paper, we restrict ourselves to microsystems built from CMOS technology as opposed to other more exotic nanotechnologies, e.g., MEMs, molecular and quantum computing technologies, etc.

[2] Power consumption issues are not considered in this work. For more discussion on the impact of technology scaling and architectural techniques on power consumption, the interested reader is directed to [5], [6], [7].

TABLE I
EFFECT OF TECHNOLOGY SCALING ON INTRA-CHIP INTERCONNECT PARAMETERS

Parameter	Technology Scaling					
	180 nm	130 nm	100 nm	70 nm	50 nm	35 nm
technology year	2000	2002	2003	2006	2009	2012
FO4 delay (picoseconds) [4]	90 psec	65 psec	50 psec	35 psec	25 psec	17.5 psec
16 FO4 worst-case clock (GHz) [4]	.7 GHz	1 GHz	1.25 GHz	1.8 GHz	2.5 GHz	3.6 GHz
ITRS-2003 projected clock (GHz) [1]	—	—	2.9 GHz	9.3 GHz	12.4 GHz	20.1 GHz
chip edge (1000's of λ's) [4]	211.1 kλ	318.4 kλ	456 kλ	711.4 kλ	1,096 kλ	1,720 kλ
chip edge (mm) [4]	19 mm	20.7 mm	22.8 mm	24.9 mm	27.4 mmm	30.1 mm
repeated diagonal chip-crossing (16 FO4 clks)	1 clk	1.5 clks	2.3 clks	3.6 clks	5.8 clks	10.7 clks
repeated diagonal chip-crossing (ITRS-2003 clks)	—	—	5.3 clks	18.6 clks	28.8 clks	59.7 clks
unrepeated edge crossing (16 FO4 clks)	1.1 clks	3.4 clks	7.8 clks	25 clks	77 clks	256 clks
unrepeated edge crossing (ITRS-2003 clks)	—	—	18 clks	129 clks	382 clks	1428 clks
semiglobal wires/edge/metal layer (pitch = 8λ)	26.4k	39.8k	57k	88.9k	137k	215k
global wires/edge/metal layer (pitch = 16λ)	13.2k	19.9k	28.5k	44.4k	68.5k	107.5k
# tiles on the chip (53.3kλ × 53.3kλ tiles)	16 tiles	36 tiles	64 tiles	169 tiles	400 tiles	1024 tiles

chip's length within a clock cycle time of 16 FO4 gate delays[3] (i.e., 3.6 GHz clock frequency) in 35nm technology.[4] Only about .07% (extrapolated) of the chip's length can be reached with a much higher 20 GHz clock as projected by ITRS-2003 for that same process generation.

For such chip-crossing wires with length that is independent of each process generation, repeaters can be used to make wiring delay grow approximately linearly with interconnect distance, as opposed to quadratically. Repeaters essentially serve as gain stages between fixed-length wire segments that scale with gate delay. Even with repeaters, wire delay can still be many factors higher than gate delay and dominate the overall delay experienced by a logic signal transmitted between gates across a chip. Again, using the delay models provided in [4] and summarized in Table I, a repeated chip-crossing global signal can reach only about 10% of the chip's length within a 16 FO4 clock cycle time in 35nm technology under worst case environmental conditions. This percentage worsens to 1.8% assuming the much higher ITRS-2003 projected clock frequency of 20 GHz. For the clock cycle time not to be slowed down to this chip-crossing critical path length, communication across semiglobal and global distances would have to be pipelined by the architecture.

Figure 3 shows the reachable distance per clock for signals transmitted on repeated semiglobal and global wires juxtaposed to chip edge-to-edge distance, assuming 16 FO4 clock cycle times for each process generation. Figure 4 shows this same information but in terms of logical spans in kλ's—that is, in terms of the number of 1000's of half gate lengths that can be spanned per clock. These figures taken directly from [4] convey some very useful information that leads to an important conclusion. Although the reachable distance per clock decreases with every technology generation, the achievable logical span remains almost constant across generations. That is, about the same number of logic gates can be crossed as technology scales since the sizes of those gates and the wires interconnecting them also scale. This means that wire delay becomes problematic only when the logic complexity increases beyond a certain point: when the logical span of the functional block being implemented grows to encompass more of the many λ's available through scaling, beyond some wire-limited upper bound. This observation therefore embraces the notion of keeping the logic complexity of functional blocks

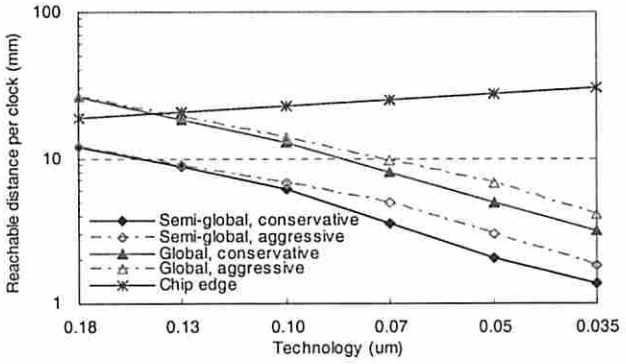


Fig. 3. Reachable distance (mm) per 16 FO4 clock for repeated wires [4].

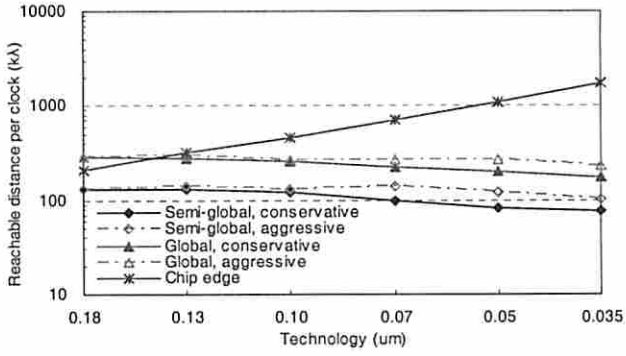


Fig. 4. Reachable distance (kλ's) per 16 FO4 clock for repeated wires [4].

[3]The designation "FO4" (or fan-out of four) is the delay (e.g., in picoseconds) of an inverter loaded by four identical inverters for a given technology.

[4]Worst case environmental conditions of high temperature and low supply voltage were assumed.

below that critical bound through modular design or "tiling."[5] This has the added benefit of bounding overall chip design and verification complexity, which would otherwise grow with technology scaling.

Modularized implementation helps to mitigate the wire-delay problems associated with technology scaling. It also enables the architecture to deal effectively with other technology scaling problems, such as the increased potential for device fabrication defects and failures owing to use damage. Electromigration, for example, causes the breaking of conductor lines over the chip's lifetime due to electron bombardment of metal atoms as electrons flow through devices and wires. Technology scaling increases the current density in conductor lines (particularly copper-based wires), which leads to greater electromigration effects, especially near *vias* which connect two metal layers. If designed for high dependability to work in the presence of defects and faults, the architecture may be able to salvage affected chips, leading to higher yield and/or increased chip lifetime.

Clearly, the growing capacity for device integration impacts architectural decisions in terms of how to utilize those resources most advantageously. What architecture best increases chip functionality while not negatively affecting achievable clock frequencies, communication latency, design/verification effort and fault resilience? While this remains an open question, what is becoming clear is the following: with implementations reaching the 10s of nanometer technology scales, interconnect delay and integrity issues have risen to the point of criticality and must now be considered first-class citizens in a microsystem's architecture.

C. Architecture Trends

Following after the trends in applications and implementation technology as discussed above, processor architectures are being designed more modularly and to exploit parallelism at higher levels beyond that which can be achieved through aggressive single thread pipelining and multiple instruction/data issuing. Current approaches are based on logically and/or physically partitioning the architecture. That is, architectures are partitioned into multiple "logical" work units, multiple "physical" work units, or combinations of both logical and physical work units. Logical partitioning into logical work units of *threads* allows the architecture to exploit the thread-level parallelism inherent in applications. Physical partitioning into physical work units of compute/operation *clusters* or *processor cores* enables the architecture to exploit the scaling properties of the underlying technology. These design trends follow naturally—as the architecture's capability to exploit parallelism increases, subsystem components naturally tend to become less tightly coupled, both logically and physically.

As mentioned in the context of implementation technology, modular design bodes well in mitigating chip-crossing wiring delay as resources comprising high affinity functional blocks can be partitioned, grouped together into small replicable physical work units, and distributed across the chip. As Table I shows, it would take an estimated 11 cycles (16 FO4 clocks) or 60 cycles (ITRS-2003 clocks) to cross the diagonal of a chip implemented in 35nm process technology using repeaters on the global wires. If repeaters were not used (this would be atypical), it would take an estimated 256 cycles (16 FO4 clocks) or 1428 cycles (ITRS-2003 clocks) in the same technology. Centralized designs employing monolithic global structures across the chip, thus, would suffer enormous pipeline latencies for most instructions executed. Such long latencies would be encountered less frequently in physically partitioned designs. For example, in tile-based designs, longer latencies would be experienced only in those cases when inter-tile communication is required.

Modular design into logical and physical work units also enables defects and faults to be tolerated more easily by the architecture through isolation, redundancy, and reconfiguration at the circuit and/or functional block level (i.e., adaptive, self-correcting, self-repairable microsystems as suggested in [1]). For example, replicating tiles across the chip can increase yield as the entire chip need not be discarded if only one or a few fabrication defects occur. Instead, all that is needed is the ability to deactivate and disconnect the affected tiles from the rest of the microsystem. This technique has been used by DRAM and SRAM designers for years to raise yield in the presence of a certain number of fabrication flaws/defects (e.g., by including redundant memory cells). In the same way, the dependability of the microsystem can be increased with reconfiguration, allowing the chip to survive faults that may occur once deployed. Partitioning the architecture in this way, however, increases the need for more explicit communication across the system at both the macroscopic and microscopic levels. More onus is placed on the hardware and/or software (i.e., compiler and run-time kernel) to provide a consistent and efficient single-system image.

As integration of functional blocks onto a single chip continues to increase, interconnection complexities that had once existed primarily at the macroscopic level between multiple chips will transfer to the microscopic level within a single chip. As shown in Table I, as many as 1024 Raw architecture tiles [8] may be implementable in 35nm technology. Support for low latency, high throughput and fault tolerant communication will therefore be critical within the microsystem's interconnection network architecture to interconnect the tiles. Some recent microprocessor chips (e.g., the Alpha 21364 [9], IBM POWER4 [10], and the now canceled Compaq Piranha [2]) have opted to integrate on-chip router switches to enable seemless upward scalability to larger multiprocessor macrosystems built from smaller microsystem modules. While this may be one way of utilizing the abundant chip resources to improve macrosystem performance, it does not ease the scaling problems within the single chip microsystem. Extending the notion of an inteconnection network to within the chip—that is, in the form of an *on-chip network*—may, in fact, be the only viable architectural approach to dealing with escalating microsystem technology scaling problems.

A number of researchers have recognized the advantages of on-chip networks [11], [12], [13], [14], [15], [16], [17], [18], [7], [19], [20]. One of the first to make a solid case was

[5]The tile size in Table I is equivalent to that used in the Raw architecture [8], which is 4mm × 4mm in .15μm technology for 16 tiles per chip.

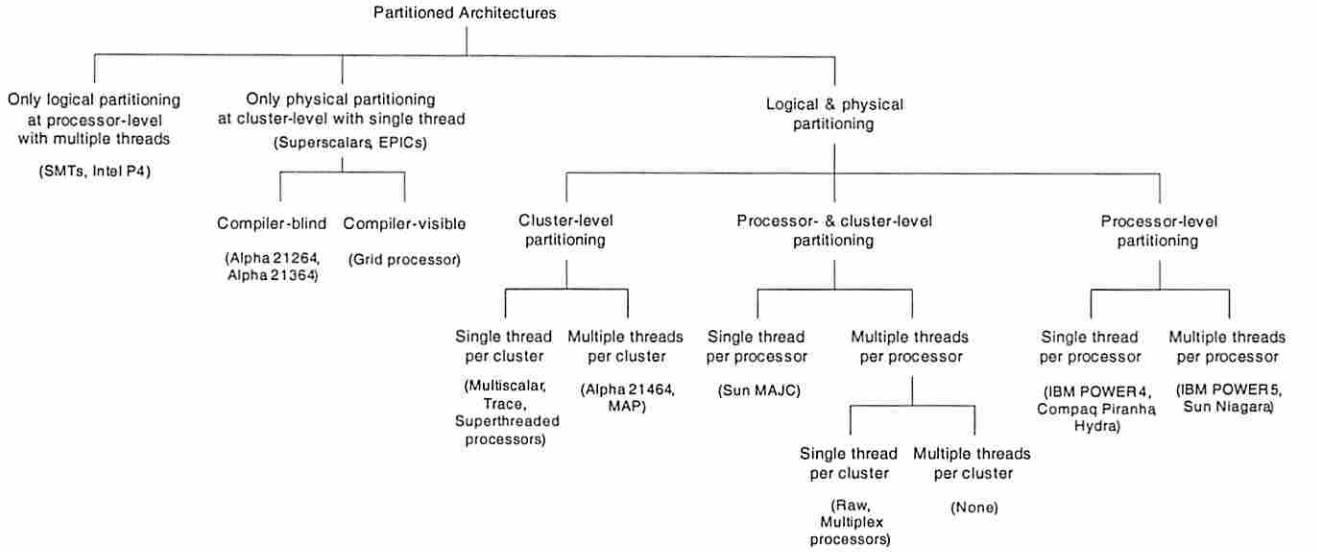


Fig. 5. A taxonomy of processor microsystem architectures based on the notion of logical and physical partitioning.

Dally in [21], [22]. There, the notion of replacing dedicated and bus-based global wiring with a general-purpose on-chip interconnection network that routes packets was first proposed. This allows sharing of wiring resources between many communication flows, and it facilitates modularity with replicable router and channel resources across the chip. It also provides better fault isolation and tolerance than a shared bus; a single fault in a network wire or buffer will not halt all transmissions. Moreover, an on-chip network can reduce the wiring complexity in a tiled design as the paths between tiles can be precisely defined and optimized early on in the design process. This enables the power and performance characteristics of global interconnects to be improved considerably. Furthermore, as deduced from the wiring and tiling information given in Table 1, more than 6,600 global wires cross each edge of the tile in a tiled chip for all process generations, assuming only the top two metal layers are used. As noted in [22], having such a large number of "pins" crossing the four edges of a tile (over 26,000 in this case) allows wiring resources to be traded off for improved network performance. For instance, this allows for very wide network channels to be implemented over which data can be sent broadside, as opposed having the data to be serialized over narrow channels.

Clearly, the architecture of processor microsystems is tending to be more communication-aware than it has been in the past. Greater emphasis is now being placed on the communication needs of the architecture and how on-chip networks can be designed to meet those needs efficiently.

III. On-Chip Microsystem Communication

In this section, the on-chip communication needs of processor microsystems are surveyed for several classes of architectures. We introduce a new classification scheme to distinguish between the various processor microsystem architectures. A previous classification presented in [23] is based mainly on the architecture's communication subsystem for operand transport. Here, the architecture of the processor microsystem is classified based on how it is partitioned, both logically and physically. For the various processor architecture classes presented, the additional communication paths needed between partitions at various pipeline stages are identified.

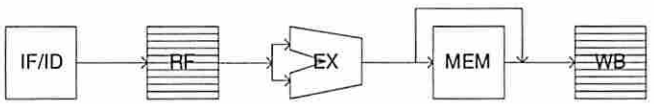


Fig. 6. Five basic pipeline stages of a monolithic processor chip.

Figure 5 gives an overview of various processor architecture classes in which some form of partitioning is used. Architectures can be partitioned only logically, only physically, or with some combination of both logical and physical partitioning. Physical partitioning can occur at the granularity of processor cores, at the granularity of functional unit clusters, or at both levels. Partitioning may or may not be exposed to the compiler; however, in this taxonomy, we assume the compiler can be made aware of logical partitioning (i.e., always compiler-visible). In the case of combined logical and physical partitioning, a single logical partition or multiple logical partitions in the form of fine- or coarse-grained thread(s) can be associated to each physical partition.

Although processors may have different pipeline structures, all must implement the following basic functions: fetch and decode instructions (IF/ID), fetch operands from the register file (RF), issue and execute instructions (EX), access data memory (MEM), and write back results to the register file (WB). More deeply pipelined architectures can be reduced down to these five basic stages, shown in Figure 6 for a monolithic (non-partitioned) processor chip. Using this template, the necessary on-chip data communication paths beyond those supported by a traditional pipeline can be observed for the various partitioned architectures. Section III-E summarizes these findings in Table II for several recent commercial and proposed processor microsystems.

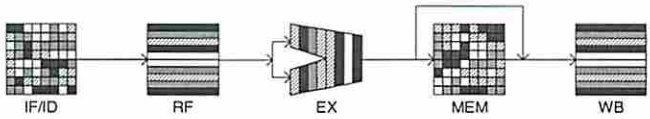


Fig. 7. An architecture in which only logically partitioning is used. Four logical work units or threads are shown with the four different shadings.

A. Logical Partitioning

Let us consider architectures which are logically partitioned. These aim to exploit thread-level parallelism of applications. A thread logical work unit may comprise an entire sequential program or it may comprise one of a number of instruction sequences composing a sequential or parallel program. Each thread has an associated state which defines its execution progress, i.e., the point reached within its instruction sequence as given by the program counter (PC) and its data values as captured by datapath registers and memory. Coarser grained TLP may be exposed by the operating system (OS) or the user as a consequence of parallelism existing at the program-level. This can occur for a collection of applications that run independent processes simultaneously in a multiprogrammed workload environment or for a parallel program that is decomposed into independent processes which may need to synchronize at various points during execution. Finer grained TLP may be exposed within a single program (e.g., with thread-level speculation on loops or instruction traces) by a parallelizing compiler, the hardware architecture or a combination of both. In any case, logically partitioning the architecture into separate thread logical work units enables multiple threads to run simultaneously.

An architecture that is *only logically partitioned* enables multiple threads to be overlapped in time over a set of shared resources. This is shown for a physically non-partitioned five-stage pipeline in Figure 7, where four different threads are distinguished from one another by their different shadings. This is illustrative of physically non-partitioned architectures which use fine-grained multithreading (FMT) or the more popular simultaneous multithreading (SMT) [24]. If needed, threads communicate data values to each other through the data memory in the MEM stage, usually through the first level data cache (L1). Hence, no extra communication paths beyond those normally supplied by the pipeline are required. However, without any physical partitioning, the monolithic control structure used to dispatch and execute the threads across the chip will result in wire delays and design/verification complexities at the implementation technology level which, ultimately, will limit the cost and performance scalability of the architecture.

An example of an SMT architecture that is logically partitioned is the Intel Pentium 4 [25]. The pipeline stages of the Pentium 4 are fully shared by two threads, including a centralized instruction dispatcher. However, the register alias table, translation look-aside buffer, PC, stack pointer, and other resources which capture the state of the threads are dedicated (partitioned) to each thread to help reduce complexity and improve performance.

B. Physical Partitioning

Now we consider architectures which are physically partitioned. Physically partitioned architectures decompose complex monolithic global structures into simpler localized structures that are distributed across the chip. This aims to exploit the resource scaling properties of CMOS implementation technology while not exacerbating wiring delay and design/verification complexity. Physically partitioning the architecture into separate physical work units allows instructions to execute on resources that are geographically close in proximity, thus within acceptable wire delays. However, since physical work units span much less chip area than their monolithic counterparts, their designs tend to be much simpler. Typically, each is designed to exploit only a modest amount of instruction-level parallelism, e.g., single- or dual-issue, in-order instruction execution over each physical work unit.

An architecture that is *only physically partitioned* allows instructions of a single thread to be overlapped in space over distinct sets of dedicated resources or *clusters*. If a cluster consists of all the functional units needed by the basic five-stage pipeline, the cluster may be considered a *processor core*. An example of a physically partitioned, logically non-partitioned five-stage pipelined architecture is shown in Figure 8 with four different clusters. In this figure, each cluster has associated to it a register file and a set of execution units (denoted by a single ALU symbol); other functional units along the pipelined datapath such as the instruction fetch/decode and data memory units are shared by instructions. Thus, this cluster is not considered a processor core. Since only a single thread is executed over all the clusters, this architecture enables ILP to be exploited beyond the modest amount provided by each cluster alone. ILP can be exploited exclusively by the hardware architecture without explicit compiler intervention, as is done in the Alpha 21264 [26], 21364 [9], and other multiple-issue superscalar architectures, or by a combination of hardware architecture and compiler support, as is done in the U.T. Austin Grid processor [27] and EPIC architectures.

Results of instructions can be communicated between clusters several ways beyond that which is normally provided by the pipeline's forwarding/bypassing paths and MEM stage. To express this, we denote the possible intercluster uni- or bi-directional data communication paths between stage A of a local cluster and stage B of a remote cluster by <A→B> and <A↔B>, respectively, where A and B are substituted with IF/ID, RF, EX or MEM. The communication paths of the WB stage are regarded as the same as the RF stage since both stages access the same resource: the register file. Therefore, for simplicity of expression, we use only the RF designation in specifying communication paths, i.e., no explicit communication path is represented for the WB stage (no distinction is made for out-of-order instruction commit, if allowed).

If resources only in the EX stage are partitioned, a communication path should exist between them to allow forwarding of ALU results from producer clusters to consumer clusters: <EX↔EX>, as shown in Figure 8. This is in addition to augmenting the existing pipeline communication paths

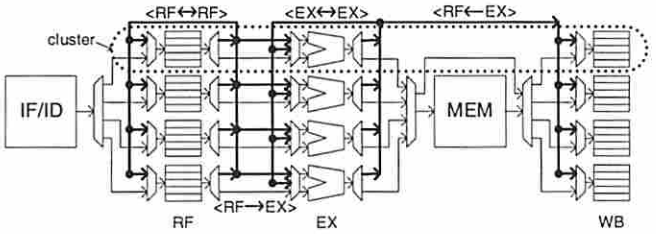


Fig. 8. An architecture in which only physical partitioning is used. Four physical work units or clusters are shown (resources of a cluster are grouped). Possible intercluster communication paths are shown in bold.

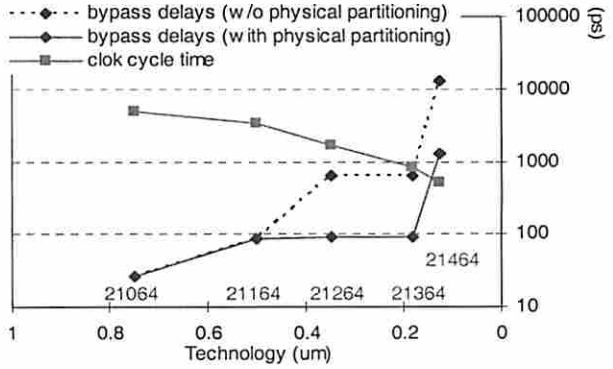


Fig. 9. Estimated bypass delays of Alpha microprocessor generations with and without physical partitioning, assuming non-repeated wires.

to allow communication to occur between shared resources and partitioned ones, e.g. between the shared RF stage and partitioned EX stage, and between the EX stage and the shared MEM stage. By partitioning resources in the RF stage as well, more new communication paths are possible, also shown in Figure 8: <RF↔RF> and <RF↔EX>. These paths can be affected by the way register files are partitioned. One way is to replicate the register files in the clusters. Replicating the register files reduces complexity, but requires communication on every register update to propagate the result to all register files (all need to have the same system image). Another way is to distribute the register files. This would require communication on a subset of the updates to the register files—that is, the minimum subset needed to maintain coherency between the register files. Distributed register files may result in less communication, but they may necessitate some synchronizing data structure (e.g., shared global register file) which adds design complexity. In Section III-C below, a few other possibilities for physically partitioned architectures are discussed in the context of combined logical partitioning.

An example of an architecture that is physically partitioned is the Alpha 21264 processor core. This core is also used in the Alpha 21364 processor chip. This single-threaded architecture has two clusters each consisting of a replicated register file and two integer execution units. The compiler is oblivious to the physical partitioning, thus the partitions are managed by hardware mechanisms (i.e., the issue queue). Intercluster communication is allowed only between the register files, i.e., <RF↔RF> is allowed, but <EX↔EX> and <RF↔EX> are not supported. One cycle is needed to propagate updates between the register files of the two clusters.

Figure 9 illustrates the value of physically partitioning the microarchitecture. Shown is the clock cycle time of various Alpha architecture generations relative to the estimated bypass delay needed to propagate results from the EX stage both for physical partitioning and without it. This data was obtained using the bypass delay model in [28]. For each successive architecture and technology generation, the bypass delays increase for the non-partitioned architecture, whereas they remain almost constant for all but the last generation of the partitioned architecture. The reason for this is the span of the bypass paths becomes larger for each architectural generation due to more execution units, physical registers, and read/write ports being implemented. For instance, the 21264 and 21364 have 1.7 and 4 times more execution units and registers, respectively, than the 21164. The 21464[29][6] has 2, 3.2, and 2 times more execution units, registers, and number of register ports, respectively, than the 21264 and 21364. Hence, a non-partitioned 21264 and 21364 would need ≈650ps or one clock cycle to bypass its results, whereas a non-partitioned Alpha 21464 would require 25 clock cycles to bypass its results. As shown, partitioning the RF and EX stages and grouping those resources into clusters mitigates this bypass delay problem. The Alpha 21464 combines logical partitioning with physical partitioning to allow four SMT threads per cluster. If repeated wires were used, the additional logic complexity needed to support this form of logical partitioning would likely prevent bypass delays from surpassing a single clock cycle.

C. Logical and Physical Partitioning

Architectures that are partitioned both logically and physically, like the Alpha 21464, are more capable of exploiting the TLP in applications as well as the scaling properties of the implementation technology. This is true as long as the logical span (i.e., logic complexity in kλ's) needed to implement the combined logical and physical work units remains below the critical wire-limited bound. It is for this reason that these partitioned work units typically are simpler in design than their monolithic counterparts, though more complex than only physically partitioned work units.

As indicated by the taxonomy in Figure 5, there are several ways in which architectures can be logically and physically partitioned. At one extreme, architectures can be physically partitioned into multiple clusters and logically partitioned into multiple finer-grained threads such that each cluster executes a single thread or multiple threads. We refer to such architectures as multiclustered processors (MCPs). At another extreme, architectures can be physically partitioned into multiple processor cores and logically partitioned into multiple coarser-grained threads possibly consisting of multiple finer-grained threads such that each processor core executes a single thread or multiple threads. These are commonly referred to as chip

[6]This processor architecture project was canceled before any chips were ever built.

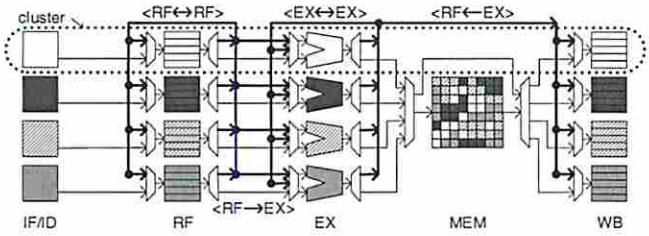


Fig. 10. A multiclustered processor chip consisting of four clusters and one thread per cluster. Possible intercluster communication paths shown in bold.

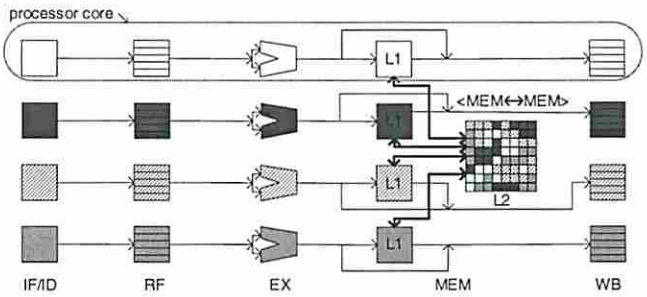


Fig. 11. A chip multiprocessor consisting of four processor cores each of which executes one of four threads. Intercluster communication paths are shown in bold from L1 cache to L2 cache.

multiprocessors (CMPs). A combination of these two extremes is an architecture that is physically partitioned into multiple processor cores each of which is composed of multiple clusters that execute one or more threads. We refer to these as hybrid multiclustered chip multiprocessors or, simply, hybrid processors (Hybrids). The communication needs of each of these architectures are briefly examined below through examples.

Multiclustered Processors

One example of a multiclustered processor is the Trace processor [30]. This architecture implements both register file structures mentioned in the previous section and takes on a pipeline structure similar to that given in Figure 8. However, different threads can be executed over the clusters similar to the shading shown in Figure 10. The architecture's replicated register files contain global registers that are visible to remote clusters. The distributed register files contain local registers that are invisible to remote clusters, as allocated by the compiler. Hence, the only intercluster communication path is through <RF↔EX>, which occurs only if the produced register value maps to the global register file.

Figure 10 illustrates further partitioning that allows each thread to work more independently in terms of instruction dispatching. The possible data communication paths are the same as discussed for Figure 8. The Multiscalar processor [31] is an example of this architecture. It allows <RF↔RF> between the distributed register files by sending and receiving register values through a queue, which is recorded in register control bit masks. As another example of this architecture, the register values produced in the EX stage in the M-Machine's MAP processor [32] can be directly written into the register files of remote clusters through <RF←EX>. In the Superthreaded processor [33], clusters have private register files that are distributed and global registers that are shared but synchronized through memory. Hence, a virtual <RF↔RF> communication path exists through the normal pipeline MEM stage as is done in SMT architectures.

More TLP can be exploited by allowing multiple threads to execute on each cluster. This idea was used in the M-Machine's MAP processor which supports cycle-by-cycle multithreading (FMT). The Alpha 21464 was designed to allow simultaneous issuing of four threads (i.e., 4-way SMT) on each cluster. For this architecture class, the same intercluster communication paths as in Figure 10 are possible, i.e., <RF↔RF>, <RF↔EX> and <EX↔EX> and the pipeline MEM stage. The different intercluster MEM stage accesses must be aggregated to and distributed from memory in some efficient way, possibly by augmenting the pipeline communication paths between the dedicated EX resources and shared MEM resource and between the MEM resource and the dedicated WB register file resources. Intracluster communication between threads in the same cluster can be through the data memory in the MEM stage as is done in the SMT architecture.

Chip Multiprocessors

Figure 11 illustrates a chip multiprocessor architecture in which partitions at the granularity of reduced processor cores with L1 instruction and data caches are replicated across the chip. Each processor core executes a different thread. Each core has access to a second level cache (L2) or higher to allow threads to synchronize or otherwise communicate with one another through memory. Hence, interprocessor communication occurs only through additional <MEM↔MEM> paths between dedicated L1s via the shared L2. Examples of processor chip designs that have adopted this architecture style include the IBM POWER4 [10], Compaq Piranha [2], and Stanford Hydra [34].

Similar to multiclustered architectures, more TLP can be exploited in chip multiprocessors by allowing multiple threads to execute on each processor core. IBM has adopted this for its POWER5 processor chip due out later this year [35]. Its design has two processors on the chip each capable of dynamically switching between executing a single thread in superscalar mode or multiple threads in SMT mode.

Hybrid Processors

Figure 12 illustrates one example of a hybrid processor chip consisting of two processors that share an L2 cache. In the figure, each processor is composed of two clusters, and each cluster executes a different thread. An alternative design would allow only a single thread per multiclustered processor, as in the Sun MAJC [36]. Yet another design alternative is to allow multiple threads to execute on each cluster within each processor consisting of multiple such clusters.[7] In order to keep the logic complexity of work units below the critical wire-limited bound, the logical span implemented over a

[7]We know of no hybrid designs that allow this as of yet, however these may emerge in the near future if the required logic complexity remains below the critical wire-limited bound.

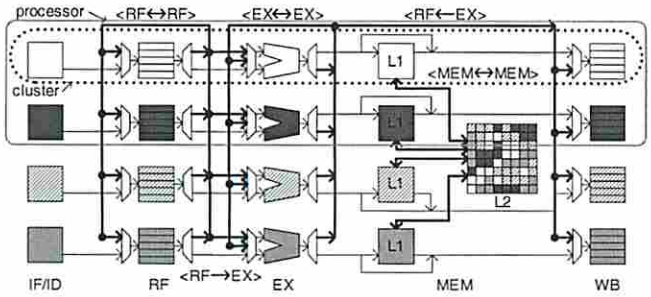


Fig. 12. A hybrid processor chip consisting of two processor cores each composed of two clusters in which one of four threads is executed. Possible intercluster communication paths are shown in bold.

physical distance of a "tile" is bounded typically by cluster granularity, not by the processor core. Hence, a processor core in a hybrid architecture may be composed of several neighboring tiles, where each tile contains all the functional units composing a cluster.

As with mulitclustered architectures, intercluster communication can occur through the same paths as in Figure 10, i.e., <RF↔RF>, <RF↔EX> and <EX↔EX>, except that MEM stage communication must go through <MEM↔MEM> paths between L1s via L2. Likewise, similar to chip multiprocessors, interprocessor communication occurs primarily through <MEM↔MEM> paths. However, the hierarchical structure of hybrid architectures makes it possible for both intercluster and interprocessor communication to use any of these paths, albeit less efficiently for some types of data. The advantage of this structure is that it enables the architecture to adapt and scale more efficiently to a given workload.

An example of a hybrid architecture that exploits such flexibility is the MIT Raw processor [8]. Its sixteen tiles can, in concept, compose anywhere from one to sixteen different processors. A single thread or multiple threads of a single program can execute over all the clusters comprising a single processor through compiler intervention (operation-operand matching [23]). Alternatively, multiple programs can run simultaneously on distinct sets of clusters comprising different processors under the control of the compiler and operating system. This capability enables the architecture to adapt to the type of parallelism presented by the application workload, whether mainly ILP, TLP, or a combination of both.

Another example is the Multiplex processor [37]. It also is reconfigurable according to the characteristics of the code being executed. If the code has less explicit TLP, the architecture is configured as multiple clusters and mainly exploits ILP or fine-grained TLP. Communication between work units occurs through <RF↔RF> to reduce the communication overhead. If the code has more explicit TLP, the architecture is configured as multiple processors, allowing infrequent interprocessor communication through <MEM↔MEM>. Reconfiguration between these two architecture modes occurs when the code characteristics change.

D. On-Chip Networks in Current Partitioned Architectures

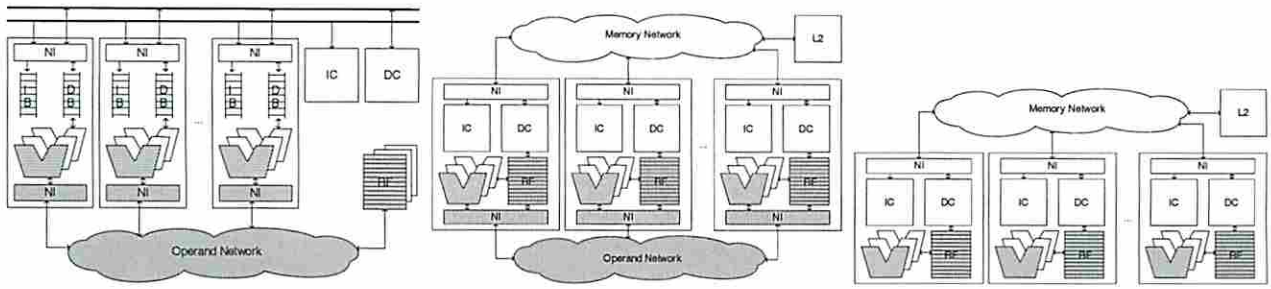
To obtain the best performance from these partitioned architectures, not only must the user, OS, hardware and/or compiler balance the parallelism in the workload over the clusters and/or processors, but also the interconnection subsystem must support efficient communication between these work units. Currently, partitioned processor architectures employ traditional networks such as buses, rings, meshes and crossbars. However, it has yet to be determined whether these are the best topologies for future on-chip processor microsystems.

In general, the communication paths used to transfer data between work units can be classified in terms of the type of data that is transmitted: register operands or cache memory blocks. Multiclustered architectures communicate register operands through <RF↔RF>, <RF↔EX>, <EX↔EX>, and the augmented MEM stage communication paths. Chip multiprocessor architectures communicate cache blocks through explicit <MEM↔MEM> communication paths between L1s via L2 cache (or some lower memory resource in the hierarchy). Hybrid architectures use a combination of both to communicate register operands and cache memory blocks. Thus, we can think of <RF↔RF>, <RF↔EX>, and <EX↔EX> communication paths as being supplied by an *operand network*, and <MEM↔MEM> communication paths as being supplied by a *memory network*. Either or both of these networks can be designed as a general-purpose packet-switched on-chip network.

Figure 13 illustrates the use of operand and memory on-chip networks for the various partitioned architecture classes. Represented in the figure are the main resources over which data is communicated and their groupings.[8] The size of data transfer in operand networks is relatively small; the payload is equivalent to the size of a register value (typically $\leq$ 64 bits for integer data). The traffic generation rate, however, is relatively high. It depends on the degree of dependence between instructions executing in different clusters. It is expected to be higher when a single thread executes over multiple clusters (i.e., exploiting only ILP) or if thread-level speculation or loop-level parallelism is used to increase the number of threads being executed, thus exploiting finer-grained TLP. Coarser-grained threads are likely to communicate mostly using memory through the MEM pipeline stage, and only at synchronization points. The size of data transfer in memory networks, however, is relatively large; the payload is equivalent to the size of a cache block (typically between 32 and 128 bytes). Traffic is generated on the event of an L1 cache miss, thus the traffic generation rate is relatively low but possibly bursty compared to operand network traffic.

The latency of operand networks is likely to affect microsystem performance more than memory networks due to the finer granularity of threads running on the system. Coarser-grained threads tend to synchronize less frequently than finer-grained threads, causing fewer communication events. With an abundance of such threads supplied by an application, the communication latency can be effectively hidden by the physical work unit switching between threads. If only a single thread is executed by each work unit, the memory network latency becomes as important as operand network latency.

[8]For completion, also shown are some resources for communicating instructions, but we focus on data communication.



(a) Multiclustered Processor (MPC) (b) Hybrid Processor (Hybrid) (c) Chip Multiprocessor (CMP)

Fig. 13. Two general classes of on-chip networks: register networks and memory networks. In this figure IC, IB, DC, DB, RF and NI stand for instruction cache, instruction buffer, data cache, data buffer, register file, and network interface, respectively. The two instances of NI for each cluster in part (a) and (b) could be two separate network interfaces or one-in-the-same. For the multiclustered processor, a generic bus structure is shown as the augmented MEM stage communication path, but this can be replaced by a general-purpose memory network as in the hybrid and chip multiprocessor architectures.

High network latency causes additional pipeline stalls to be suffered or requires an increased number of pipeline stages for communication. Several approaches have been proposed to deal with on-chip network latency. One approach is to reduce the number of intercluster communications, thus the number of times this latency is experienced. The Multicluster Architecture [38] does this by efficiently partitioning the code to have dependent instructions map to the same cluster. Another technique tries to hide network latency by using value prediction [39]. Neither of these approaches, however, attempt to optimize the design of the on-chip network architecture to achieve low latency communication.

The bandwidth of the network also affects performance as well as the implementation complexity. For instance, as the bandwidth of the operand network increases, the number of register file ports may also need to increase as well as the complexity of the intracluster forwarding/bypass logic. In the Alpha 21264 processor core, updated register values are broadcast to the remote cluster (there are only two clusters) since the register files are duplicated. Sufficient bandwidth needs to be supplied by the operand network to support this. The Multiscalar and Superthreaded processor chips implement a unidirectional ring as the operand network since traffic flows only in one direction from producers to consumers. The Grid and Raw processor chips implement a mesh network, while the Trace processor and the M-Machine use a bus and crossbar operand network, respectively. The Hydra, Alpha 21264, Trace, and Grid processor chips interconnect L1 caches to the shared L2 cache using a bus memory network. IBM POWER4, Compaq Piranha, and M-Machine's MAP use a crossbar as the memory network. The Sun MAJC implements a bus or a crossbar, and the Raw processor implements a separate mesh as the memory network.

E. Summary

Table II summarizes the various types and degrees of partitioning applied to recently proposed or implemented processor architectures. The SMT architectures would likely benefit from an on-chip switched network to replace the dedicated links used for communicating global control signals and bypassing data across the chip even though no explicit operand or memory network is required for the monolithic microsystem. Likewise, the multiclustered architectures that support intercluster operand transport using dedicated links (which offer severely limited scalability of only two clusters) similarly could benefit from a switched network. Some multiclustered and CMP architectures implement operand and memory networks using buses and crossbars, allowing as many as four clusters and eight processors. However, in addition to having their own scaling limitations, these centralize interconnecting structures are inconsistent with the underlying design objective behind partitioning the architecture—that of localizing communication within groups of high-affinity functional blocks that span a small geographical area. Chip-crossing wiring delays may be suffered with each access to these networks. In contrast, switched networks like rings and meshes not only localize communication and increase scalability of tiled microsystems, they also better enable modular design of on-chip operand and memory networks. The Raw processor, for instance, currently implements 16 mesh-connected tiles but scales to 64 tiles in 100nm technology, and supports glueless connection of up to 64 Raw chips connected in a rectangular mesh topology, allowing Raw macrosystems of 1K to 4K total tiles [8], [23].

IV. Future Research Directions

Chip-crossing wire delays and logic design/verification complexity will continue to play an increasingly important role in the way future processor microsystems are designed. With integration densities soon to surpass the billion transistor mark, the way in which power consumption, device defects and failures are dealt with will also be important factors which define the architecture of these microsystems. Given the criticality of these issues, a number of interesting research directions are being pursued.

One interesting line of research is the exploration of ways to partition other monolithic architectural structures to further reduce the frequency of chip-crossing communication events. The L2 cache shown in Figures 11–13, for instance, is currently implemented as a monolithic or banked structure that resides at the periphery of the partitioned array of clusters

TABLE II

SUMMARY OF PARTITIONED PROCESSOR ARCHITECTURES IMPLEMENTED OR PROPOSED BY INDUSTRY OR ACADEMIA.

Processor chip	Logical partitioning		Physical parititioning			Arch. category	On-chip network	
	# threads	thread size	# proc.	# clusters	partitioned stages		operand	memory
Intel P4[25]	2	coarse-grain	1	—	—	SMT	—	—
Alpha 21264[26]	1	coarse-grain	1	2	integer RF, EX	MCP†	ded. link	bus
Alpha 21364[9]	1	coarse-grain	1	2	integer RF, EX	MCP†	ded. link	bus
Grid proc.[27]	1	coarse-grain	1	64‡	EX, partial RF	MCP†	mesh	bus/mesh
Multiscalar proc.[31]	4‡	loop/subroutine	1	4‡	IF/ID, RF, EX	MCP	unidir. ring	crossbar
Trace proc.[30]	4‡	trace	1	4‡	RF, EX	MCP	bus	bus
Superthreaded proc.[33]	2‡	loop	1	2‡	IF/ID, RF, EX	MCP	unidir. ring	bus
Alpha 21464[29]	4	coarse-grain	1	2	integer RF, EX	MCP	ded. link	bus
MAP[32]	16	loop/coarse-grain	1	4	IF/ID, RF, EX	MCP	crossbar	crossbar
IBM POWER4[10]	2	coarse-grain	2	—	all	CMP	—	crossbar
Compaq Piranha[2]	8	coarse-grain	8	—	all	CMP	—	crossbar
Hydra[34]	4	coarse-grain	4	—	all	CMP	—	bus
IBM POWER5[35]	4	coarse-grain	2	—	all	CMP	—	crossbar
Sun Niagara[40]	32	coarse-grain	8	—	all	CMP	—	crossbar
Sun MAJC[36]	2††	coarse-grain	2††	4	RF, EX (cluster)	Hybrid	ded. link	bus/crossbar
Raw proc.[8]	≤ 16	coarse-grain	16 total tiles		all	Hybrid	mesh	mesh
Multiplex[37]	4	loop/coarse-grain	4 proc. or clusters		all	Hybrid	unidir. ring	bus

† Only physical partitioning at cluster-level with single thread

‡ The minimum number of simulated threads or clusters

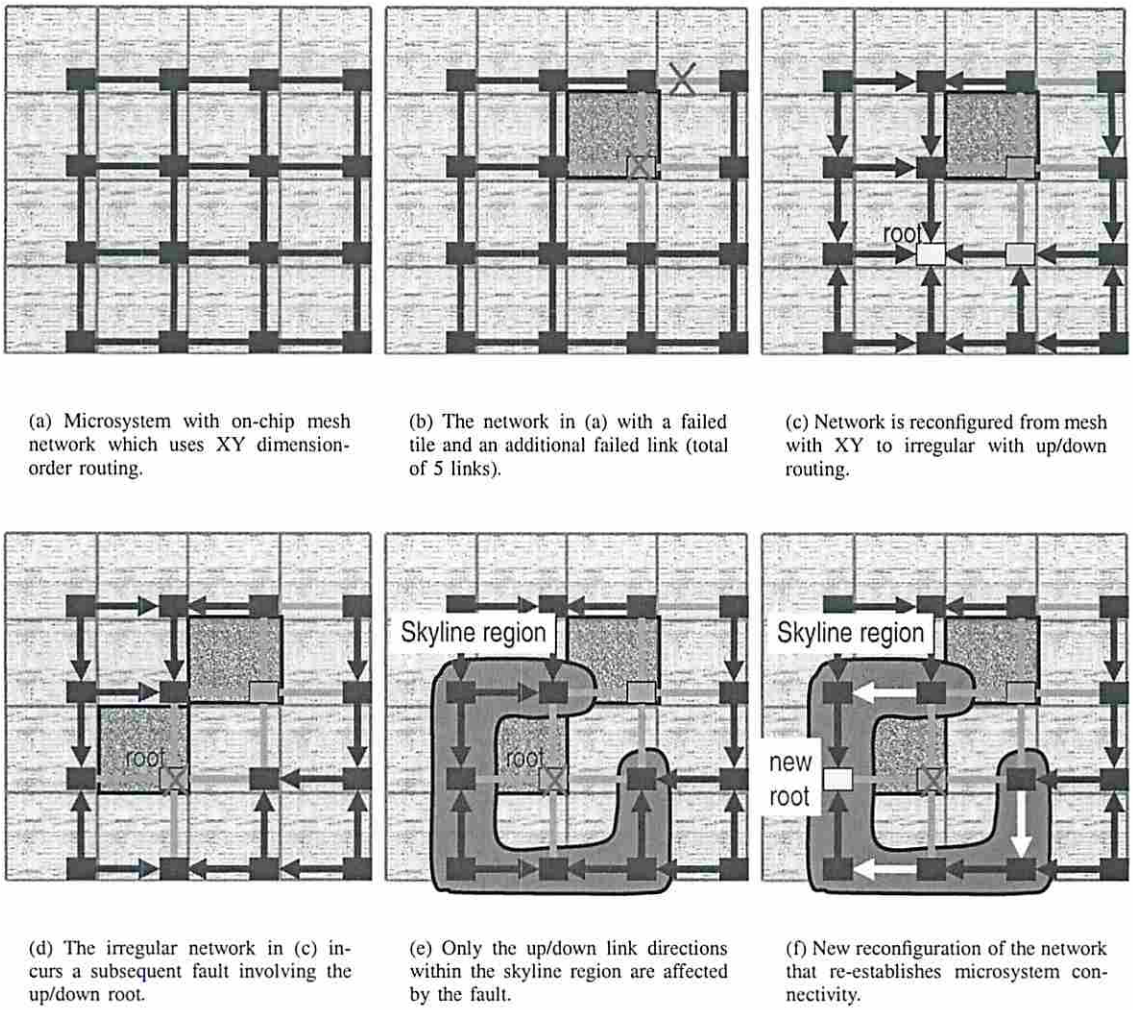
†† According to [36] more than two processors are supported as well (MAJC-5200 has two processors).

or processors, or it resides off-chip in the form of DRAM. Communication of memory blocks to and from L2 or DRAM has uniformly high latency as chip-crossing wiring delay is encountered on each access, in addition to memory look-up time. Such large, uniform-access cache and memory architectures are undesirable in future implementation technologies for reasons discussed in Section II-B. Partitioning and distributing the L2 cache banks across the tiles in a tiled design is a preferred alternative. In [41], it is estimated that an on-chip 16MB L2 cache built in a 50nm process could have a local bank access time of only 4 cycles while the farthest remote bank might take as many as 47 cycles. This motivates researchers to actively explore the design space of adaptive, dynamic non-uniform cache architectures (NUCAs) that scale with technology yet still exploit the locality referencing behavior of applications.

Another interesting research topic is the development of optimized on-chip network architectures for achieving low communication latency, resource cost, and power consumption. While packet-switched networks are advantageous for many reasons [22], the current ones implemented (i.e., rings and meshes) are not always the best choice. The on-chip network should provide sufficient bandwidth to handle the communication requirements of the applications running on the microsystem while not over-committing resources that will be inefficiently used. Optimizing the required resources is particularly important for on-chip networks as interfaces, switches and links may consume a large portion of chip area and power if not optimized. For example, the Raw processor allocates nearly 50% of its chip area to the on-chip network, and the integrated on-chip macro router and links of the Alpha 21364 consumes about 20% of the chip's power [42]. Research is underway to address these optimization issues. In [7], on-chip network architectures based on express cubes are explored which show a reduction in network power of up to $\approx$45% with no degradation in network performance and even improved latency-throughput in some cases. In [19], [43], design methodologies and algorithms are developed for constructing deadlock-free network topologies and routing functions that optimize certain design objectives, such as minimizing resource cost and communication energy. The generated networks exploit communication behavior extracted from a target application workload and, thus, are application-specific on-chip networks. For some applications, up to 60% fewer switch and link resources are needed as compared to meshes and tori, with only negligible performance degradation as compared to a crossbar [19]; in [43], $\approx$52% energy savings have been observed as compared to ad-hoc network designs.

Yet another very interesting research direction currently being pursued in the area of on-chip networked microsystems (among many others) is dynamic reconfiguration. Techniques for reconfiguring the microsystem's architecture in response to changes in code characteristics from ILP mode to TLP mode and from multiclustered to chip multiprocessor and various hybrid forms are already being implemented in current processors (see Section III-C *Hybrid Processors*). This will continue in the future. The polymorphic paradigm espoused by the TRIPS architecture [44], for example, points to future possible directions in this area.

In addition to its ability to reconfigure dynamically in response to application parallelism, the architecture of future microsystems should also be capable of reconfiguring in response to changes in communication behavior and/or microsystem state. Doing so would increase the microsystem's performance and dependability, enabling the on-chip network to adapt to changing communication demands as well as to survive defect, error, and fault scenarios that would otherwise cause it to perform poorly, or even fail. Section II-C describes the benefit that modular design has in enabling defects and faults to be more easily isolated and tolerated. In tile-based designs, deadlock-free dynamic reconfiguration techniques such as those described in [45], [46], [47] may be applicable, enabling failed tiles to be removed from the rest of



(a) Microsystem with on-chip mesh network which uses XY dimension-order routing.

(b) The network in (a) with a failed tile and an additional failed link (total of 5 links).

(c) Network is reconfigured from mesh with XY to irregular with up/down routing.

(d) The irregular network in (c) incurs a subsequent fault involving the up/down root.

(e) Only the up/down link directions within the skyline region are affected by the fault.

(f) New reconfiguration of the network that re-establishes microsystem connectivity.

Fig. 14. Illustration of on-chip network reconfiguration in response to defects or faults which may occur at different times during a chip's life cycle.

the functioning microsystem efficiently. Figure 14 illustrates an example of this for a 16 tile processor microsystem.

In Figure 14(a), the tiles are initially interconnected using a rectangular mesh network which routes packets in XY dimension order to avoid deadlock. In Figure 14(b), one of the tiles has become faulty, which disables its router and its four links to routers in neighboring tiles. In addition, a link fault occurs in the east-west link of the tile in the upper righthand corner of the chip. In response to these faults, the network is reconfigured in Figure 14(c) to an irregular topology on which packets route in an up*/down* fashion, traversing zero or more links in the up direction followed by zero or more links in the down direction to reach their destinations in a deadlock-free manner. The processes defined in [45], [46], [47] enable this transformation of the routing function to be done dynamically and in a deadlock-free manner. Subsequent faults can be handled similarly, causing the network to be reconfigured into a different irregular topology that reconnects all functioning tiles and communication links, as shown in Figures 14(d)–(f). By demarcating a "skyline" region, as described in [48], it is possible to confine the reconfiguration process to only a subset of the network resources to improve reconfiguration response time. Just how effective these proposed techniques are when applied to on-chip networks has yet to be fully investigated.

These and many other approaches for improving cost, performance, and fault resilience of billion transistor, highly parallel and partitioned, on-chip networked microsystems appear to be promising areas of future research.

V. Conclusion

Trends in application, implementation technology, and process architecture are highlighted in this work. A taxonomy of partitioned processor microsystem architectures is presented, and the communication needs of various classes of these architectures is also briefly surveyed. While this is not an exhaustive study, a clear conclusion is that interconnect delay and integrity issues have risen to the point of criticality and are now first-class citizens in a microsystem's architecture.

There is a trend toward designing on-chip networked microsystems derived from logically and physically partitioning the processor architecture. Partitioning the architecture enables the parallelism offered by growing application workloads

to be well exploited. It also enables the scaling properties of the underlying implementation technology to continue to provide increasing performance and not be encumbered by chip-crossing wire delay, which no longer is a negligible factor. A consequence of partitioning the architecture is that an interconnecting subsystem must be provided that enables efficient communication between the various partitions. The resulting on-chip network should conform to a modular, switch-based design approach to ease design/verification complexity as well as to allow upward scalability both at the microsystem level and, if possible, at the macrosystem level.

While the above stated conclusion seems evident, the definitive solutions to the various partitioning and on-chip network issues mentioned in this paper remain to be discovered. Several ongoing and future research directions show promise while many others have yet to be identified. A future publication may provide some answers [49], but it seems likely that an active debate on the subject will continue for many years to come, or at least until the end of the CMOS technology era.

ACKNOWLEDGMENT

We thank a couple of members from the *SMART* Interconnects Group, Wai Hong Ho and Bilal Zafar, who reviewed an earlier draft of this work.

REFERENCES

[1] *The International Technology Roadmap for Semiconductors: 2003 Update.* Semiconductor Industry Association, http://public.itrs.net/home.htm, 2003.

[2] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000, pp. 282–293.

[3] L. A. Barroso, J. Dean, and U. Holzle, "Web Serach for a Planet: The Google Cluster Architecture," *IEEE Micro*, pp. 22–28, March-April 2003.

[4] R. Ho, K. W. Mai, and M. A. Horowitz, "The Future of Wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, April 2001.

[5] V. Zyuban and P. Kogge, "Optimization of High-performance Superscalar architectures for energy efficiency," in *Proceedings of International symposium on Low power electronics and design*, July 2000, pp. 84–89.

[6] T. Mudge, "Power: A first-class architectural design constraint," *IEEE Computer*, vol. 34, no. 4, pp. 52–58, April 2001.

[7] H. Wang, L.-S. Peh, and S. Malik, "Power-driven Design of Router Microarchitectures in On-chip Networks," in *Proceedings of the 36^{th} Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, December 2003, pp. 105–116.

[8] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, March/April 2002.

[9] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The Alpha 21364 Network Architecture," in *Symposium on High Performance Interconnects (HOT Interconnects 9)*. IEEE Computer Society Press, August 2001, pp. 113–117.

[10] J. M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5–26, January 2002.

[11] L. Benini and G. D. Micheli, "Networks on Chip: A New SoC Paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–80, January 2002.

[12] V. Raghunathan, M. B. Srivastava, and R. K. Gupta, "A Survey of Techniques for Energy Efficient On-Chip Communication," in *Proceedings of Design Automation Conference*, June 2003.

[13] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. M. Rabaey, and A. L. Sangiovanni-Vincentelli, "Addressing the system-on-a-chip interconnect woes through communication-based design," in *Proceedings of Design Automation Conference*, June 2001, pp. 667–672.

[14] J. Liang, S. Swaminathan, and R. Tessier, "aSoC: A scalable, singlechip communications architecture," in *International Conference on Parallel Architectures and Compilation Techniques*, October 2000, pp. 37–46.

[15] N. Swaminathan and R. Mahapatra, "Communication Architecture Synthesis of Packet-Switched Network-on-Chip, Tech. Rep. TR-CS-2002-08-0, 2002.

[16] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," in *Proceedings of the Design Automation and Test in Europe*, March 2000, pp. 250–256.

[17] J.-M. Parcerisa, J. Sahuquillo, A. Gonzalez, and J. Duato, "Efficient Interconnects for Clustered Microarchitectures," in *Proceedings of 2002 International Conference on Parallel Architectures and Compilation Techniques*, September 2002.

[18] A. Aggarwal and M. Franklin, "Hierarchical Interconnects for On-chip Clustering," in *Proceedings of International Parallel and Distributed Processing Symposium*, April 2002.

[19] W. H. Ho and T. M. Pinkston, "A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns," in *Proceedings of the 9^{th} International Symposium on High-Performance Computer Architecture*. IEEE Computer Society Press, February 2003, pp. 377–388.

[20] T. M. P. *(Panel Moderator)*, "What Will Have The Greatest Impact In 2010: Processor, Memory, or Interconnnect Architecture?" in *Proceedings of the 8^{th} Int'l Symp. on High Performance Computer Architecture*. www.usc.edu/dept/ceng/pinkston/presentations/statistic.html, February 2002.

[21] W. J. Dally, "Interconnect limited VLSI architecture," in *Proceedings of International Interconnect Technology Conference*, May 1999, pp. 15–17.

[22] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Proceedings of the Design Automation Conference (DAC)*. ACM, June 2001, pp. 684–689.

[23] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures," in *Proceedings of the 9^{th} International Symposium on High-Performance Computer Architecture*. IEEE Computer Society Press, February 2003, pp. 341–353.

[24] D. M. Tullsen, S. Eggers, and H. M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," in *Proceedings of the 22rd International Symposium on Computer Architecture*, June 1995, pp. 392–403.

[25] D. Koufaty and D. T. Marr, "Hyperthreading Technology in the Netburst Microarchitecture," *IEEE Micro*, vol. 23, no. 2, pp. 56–65, March/April 2003.

[26] R. E. Kessler, "The Alpha 21264 microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, March/April 1999.

[27] R. Nagarajan, K. Sankaralingam, D. Burger, and S. W. Keckler, "A design space evaluation of grid processor architectures," in *Proceedings of the 34th Annual International Symposium on Microarchitecture*, December 2001, pp. 40–51.

[28] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Quantifying the Complexity of Superscalar Processors," University of Wisconsin, Madison, Tech. Rep. CS-TR-1996-1328, November 1996.

[29] K. Diefendorff, "Compaq Chooses SMT for Alpha," *Microprocessor Report*, vol. 13, no. 16, pp. 5–11, December 1999.

[30] J. E. Smith and S. Vajapeyam, "Trace Processors: Moving to Fourth-Generation Microarchitectures," *IEEE Computer*, vol. 30, no. 9, pp. 68–74, September 1997.

[31] G. S. Sohi, S. E. Breach, and T. N. Vijaykumar, "Multiscalar Processors," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995, pp. 414–425.

[32] M. Fillo, S. W. Keckler, W. J. Dally, N. P. Carter, A. Chang, Y. Gurevich, and W. S. Lee, "The M-Machine Multicomputer," in *Proceedings of the 28th International Symposium Microarchitecture*, December 1995, pp. 146–156.

[33] J.-Y. Tsai, J. Huang, C. Amlo, D. J. Lilja, and P.-C. Yew, "The Superthreaded Processor Architecture," *IEEE Transactions on Computers*, vol. 48, no. 9, pp. 881–902, September 1999.

[34] L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, and K. Olukotun, "The Stanford Hydra CMP," *IEEE Micro*, vol. 20, no. 2, pp. 71–84, February 2000.

[35] P. N. Glaskowsky, "IBM rasies curtain on Power5," *Microprocessor Report*, vol. 17, no. 10, pp. 13–14, October 2003.

[36] "MAJC architecture tutorial," *white paper, Sun microsystems*, September 1999.

[37] S. W. Kim, C.-L. Ooi, I. Park, R. Eigenmann, B. Falsafi, and T. N. Vijaykumar, "Multiplex: unifying conventional and speculative thread-level parallelism on a chip multiprocessor," in *Proceedings of International Conference on Supercomputing*, June 2001, pp. 368–380.

[38] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. G. Vranesic, "The Multicluster Architecture: Reducing Cycle Time Through Partitioning," in *Proceedings of International Symposium on Microarchitecture*, December 1997, pp. 149–159.

[39] J.-M. Parcerisa and A. Gonzalez, "Reducing wire delay penalty through value prediction," in *Proceedings of International Symposium on Microarchitecture*, December 2000, pp. 317–326.

[40] K. Krewell, "Sun's Niagara Pours on the Cores," *Microprocessor Report*, pp. 1–3, September 2004.

[41] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002, pp. 211–222.

[42] L. Shang, L.-S. Peh, and N. K. Jha, "Power-Efficient Interconnection Networks: Dynamic Voltage Scaling with Links," *Computer Architecture Letters*, vol. 1, no. 2, pp. 1–4, May 2002.

[43] J. Hu and R. Marculescu, "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures," in *Proceedings of Design, Automation and Test in Europe Conference*. Munich, Germany, March 2003.

[44] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," in *Proceedings of the International Symposium on Computer Architecture*, June 2003, pp. 422–433.

[45] T. M. Pinkston, R. Pang, and J. Duato, "Deadlock-free Dynamic Reconfiguration Schemes for Increased Network Dependability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 8, pp. 780–794, August 2003.

[46] O. Lysne, T. M. Pinkston, and J. Duato, "A Methodology for Developing Dynamic Network Reconfiguration Processes," in *Proceedings of the International Conference on Parallel Processing (ICPP'03)*. IEEE Press, October 2003, pp. 77–86.

[47] J. Duato, O. Lysne, R. Pang, and T. M. Pinkston, "A theory for deadlock-free dynamic network reconfiguration," *USC Technical Report CENG-2003-06*, vol. (available at www.usc.edu/dept/ceng/pinkston/SMART.html), pp. pages 1–30, 2003.

[48] O. Lysne and J. Duato, "Fast Dynamic Reconfiguration in Irregular Networks," in *The 2000 International Conference on Parallel Processing*. IEEE Computer Society, August 2000, pp. 449–458.

[49] L.-S. Peh and T. M. Pinkston, "Special Issue on On-Chip Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, February 2005.

Timothy Mark Pinkston completed his B.S.E.E. degree from The Ohio State University in 1985 and his M.S. and Ph.D. degrees in electrical engineering from Stanford University in 1986 and 1993, respectively. Prior to joining the University of Southern California (USC) in 1993, he was a Member of Technical Staff at Bell Laboratories, a Hughes Doctoral Fellow at Hughes Research Laboratory, and a visiting researcher at IBM T. J. Watson Research Laboratory. Presently, Dr. Pinkston is a Professor and Director of the Computer Engineering Division of the EE-Systems Department at the University of Southern California, and he heads the *SMART* Interconnects Group. His current research interests include the development of deadlock-free adaptive routing techniques, dynamic reconfiguration techniques, and on-chip network and router architectures for achieving high-performance communication in microprocessor and parallel computer systems—scalable parallel processor and cluster computing systems. Dr. Pinkston has authored over seventy refereed technical papers and has received numerous awards, including the Zumberge Fellow Award, the National Science Foundation Research Initiation Award, and the National Science Foundation Career Award. Dr. Pinkston is a member of the ACM and a Senior Member of the IEEE. He has also been a member of the Program Committee for several major conferences (ISCA, HPCA, ICPP, IPPS/IPDPS, ICDCS, SC, CS&I, CAC, PCRCW, OC, MPPOI, LEOS, WOCS, and WON), the Program Chair for HiPC'03, the Program Vice-chair for EuroPar'03 and ICPADS'04, the Program Co-chair for MPPOI'97, the Tutorials Chair for ISCA'04, the Workshops Chair for ICPP'01, and the Finance Chair for Cluster 2001. He recently concluded two 2-year terms as an Associate Editor for the IEEE Transactions on Parallel and Distributed Systems (TPDS) and is currently serving as a Guest Editor of TPDS for a Special Issue on On-Chip Networks (due out February, 2005).

Jeonghee Shin received her BS and MS degrees in computer engineering from Pusan National University in 1999 and 2001, respectively. She is currently pursuing the PhD degree in computer engineering at the University of Southern California. Her research interests include the design of high-performance on-chip interconnection networks for chip multiprocessors, the development of network simulators, performance analysis, and parallel processing.