Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling

Rakesh Kumar†, Victor Zyuban‡, Dean M. Tullsen†

†Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

‡IBM TJ Watson Research Center
1101 Kitchawan Road
Yorktown Heights, NY 10598

Abstract

This paper examines the area, power, performance, and design issues for the on-chip interconnects on a chip multiprocessor, attempting to present a comprehensive view of a class of interconnect architectures. It shows that the design choices for the interconnect have significant effect on the rest of the chip, potentially consuming a significant fraction of the real estate and power budget. This research shows that designs that treat interconnect as an entity that can be independently architected and optimized would not arrive at the best multi-core design. Several examples are presented showing the need for careful co-design. For instance, increasing interconnect bandwidth requires area that then constrains the number of cores or cache sizes, and does not necessarily increase performance. Also, shared level-2 caches become significantly less attractive when the overhead of the resulting crossbar is accounted for. A hierarchical bus structure is examined which negates some of the performance costs of the assumed baseline architecture.

1 Introduction

High-performance processor architectures are moving towards designs that feature multiple processing cores on a single chip. These designs have the potential to provide higher peak throughput, easier design scalability, and greater performance/power ratios than monolithic designs. At least two dual-core architectures [15, 16] are already on the market.

However, in spite of the growing trend to put multiple cores on the die, a deep understanding is lacking in the literature of the design space of the interconnection framework, and particularly how it interacts with the rest of the multi-core architecture. For a given number of cores, the "best" interconnection architecture in a given chip multiprocessing environment depends on a myriad of factors, including performance objectives, power/area budget, bandwidth requirements, technology, and even the system software. This paper attempts to present a comprehensive analysis of the design issues for a class of chip multiprocessor interconnection architectures.

While interconnects are relatively well understood for connecting chips, multi-chip modules, and board-level nodes, connecting cores on the same chip presents a distinctly different problem. This is because power, area, latency, and bandwidth are all first-order design constraints for on-chip interconnects. Secondly, the design choices for the cores, caches, and interconnect interact to a much greater degree. For example, an aggressive interconnect design consumes power and area resources that then constrains the number, size, and design of the cores and caches. Thus, unlike a conventional multiprocessor, performance is not necessarily maximized by the highest bandwidth interconnect available. Of course, the converse is also true, that the number and type of cores (as well as on-chip memory) also dictate requirements on the interconnect. In fact, increasing the number of cores places conflicting demands on the interconnect – requiring higher bandwidth while decreasing available real estate.

In this paper, we model the implementation of several interconnection mechanisms and topologies, allowing us to quantify their area, power, and latency overheads. We highlight the various tradeoffs between performance and power, and between performance and area. We study the sensitivity of these overheads to technology, pipeline depth, number of cores, and onchip memory sizes. We also present one novel interconnection architecture which effectively exploits some of the behaviors identified by this research – a hierarchical bus structure that reduces local communication overheads, at the expense of cross-chip latencies. All of this work is done in the context of important commercial workloads.

This study has three primary goals: (1) to study on-chip interconnect architectures with detailed and accurate models that go well beyond prior published work in this area, (2) to use those models to explore the design space of interconnect architectures, including crossbars, point-to-point connections, bus architectures, and various combinations of those technologies at different widths, and (3) to demonstrate that neither the core/cache architectures nor the interconnect architecture can be derived independently, but that the best design is a result of careful co-design of each of these elements.

The results of this research deliver several strong messages for architects of future multi-core systems. First, the interconnect is a critical design element of a multi-core architecture. On an 8-core processor, for example, the interconnect, even under conservative assumptions can consume the power equivalent of one core, take the area equivalent of three cores,

A more detailed version of this paper appears as [20]

0-7695-2270-X/05/$20.00 (C) 2005 IEEE

and add delay that accounts for over half the L2 access latency. Second, co-design of the cores, caches and interconnect is critical to achieving the best architecture. We show, for example, one configuration where decreasing the bandwidth of the interconnect increases performance, due to the complex interplay of performance and area on a chip multiprocessor. As another example, when interconnect overheads are taken into account, sharing L2 cache among multiple cores is significantly less attractive than when these factors are ignored. Even with four-way sharing of L2 caches, the interconnect can be as much as a quarter of the chip area. In fact, we show repeatedly in this research that design decisions made ignoring the impact of the interconnect are often the opposite of the decision indicated when these factors are properly accounted for.

The rest of this paper is organized as follows. Section 3 discusses the various interconnection mechanisms. Sections 4 and 5 discuss the methodological details for this study. Sections 6 and 7 present overhead analysis for various interconnection mechanisms. Section 8 discusses scaling of these overheads as a function of technology and pipeline depth. An example interconnection optimization is presented in Section 9. Section 10 concludes.

2 Related Work

There have been several proposals and implementations of high-performance chip multiprocessor architectures [5, 12, 15, 16]. The proposed interconnect for Piranha [5] was an intra-chip switch. Cores in Hydra [12] are connected to the L2 cache through a crossbar. In both cases, the L2 cache is fully shared. IBM Power4 [15] has two cores sharing a triply-banked L2 cache. Connection is through a crossbar-like structure called the CIU (core-interface unit).

In this paper, we consider bus-based and crossbar-based interconnections. There have been recent proposals for packet-based on-chip interconnection networks [13, 7, 25]. Packet-based networks structure the top level wires on a chip and facilitate modular design. Modularity results in enhanced control over electrical parameters and hence can result in higher performance or reduced power consumption. These interconnections can be highly effective in particular environments where most communication is local, explicit core-to-core communication. However, the cost of distant communication is high. Due to their scalability, these architectures are attractive for a large number of cores. The crossover point where these architectures become superior to the more conventional interconnects studied in this paper is not clear, and is left for future work.

There is a large body of related work evaluating tradeoffs between bus-based and scalable shared memory multiprocessors, in the context of conventional (multiple-chip) multiprocessors. Some earlier implementations of the interconnection networks for multiprocessors have been described in [10, 31, 23, 27, 26, 11, 2, 22, 3]. However, on-chip interconnects have different sets of tradeoffs and design issues. Thus, the conclusions of those prior studies must be re-evaluated in the context of on-chip multiprocessors with on-chip interconnects.

3 Interconnection Mechanisms

In this paper, we study three interconnection mechanisms that may serve particular roles in our interconnect hierarchy – a shared bus fabric (SBF) that provides a shared connection to various modules that can source and sink coherence traffic, a point-to-point link (P2P link) that connects two SBFs in a system with multiple SBFs, and a crossbar interconnection system. Many different modules may be connected to these fabrics, which use them in different ways. But from the perspective of the core, an L2 miss goes out over the SBF to be serviced by higher levels of the memory hierarchy, another L2 on the same SBF, or possibly an L2 on another SBF connected to this one by a P2P link. If the core shares L2 cache with another core, there is a crossbar between the cores/L1 caches and the shared L2 banks. Our initial discussion of the SBF in this section assumes private L2 caches.

The results in this paper are derived from a detailed model of a complex system, which are described in the next few sections. The casual reader may want to skim Sections 3 through 5 and get to the results in Section 6 more quickly.

3.1 Shared Bus Fabric

A Shared Bus Fabric is a high speed link needed to communicate data between processors, caches, IO, and memory within a CMP system in a coherent fashion. It is the on-chip equivalent of the system bus for snoop-based shared memory multiprocessors [10, 31, 23]. We model a MESI-like snoopy write-invalidate protocol with write-back L2s for this study [4, 15]. Therefore, the SBF needs to support several coherence transactions (request, snoop, response, data transfer, invalidates, etc.) as well as arbitrate access to the corresponding buses. Due to large transfer distances on the chip and high wire delays, all buses must be pipelined, and therefore unidirectional. Thus, these buses appear in pairs; typically, a request traverses from the requester to the end of one bus, where it is queued up to be re-routed (possibly after some computation) across a broadcast bus that every node will eventually see, regardless of their position on the bus and distance from the origin. In the following discussion a bidirectional bus is really a combination of two unidirectional pipelined buses.

We are assuming, for this discussion, all cores have private L1 and L2 caches, and that the shared bus fabric connects the L2 caches (along with other units on the chip and off-chip links) to satisfy memory requests and maintain coherence. Below we describe a typical transaction on the fabric.

3.1.1 Typical transaction on the SBF

A load that misses in the L2 cache will enter the shared bus fabric to be serviced. First, the requester (in this case, one of the cores) will signal the *central address arbiter* that it has a request. Upon being granted access, it sends the request over

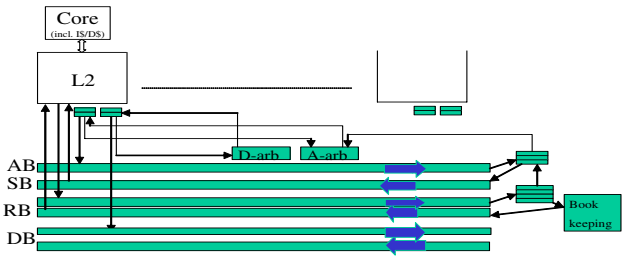

Figure 1. The assumed shared bus fabric.

an *address bus* (AB in Figure 1). Requests are taken off the end of the address bus and placed in a snoop queue, awaiting access to the *snoop bus* (SB). Transactions placed on the snoop bus cause each snooping node to place a response on the *response bus* (RB). Logic and queues at the end of the response bus collect these responses and generate a broadcast message that goes back over the response bus identifying the action each involved party should take (e.g., source the data, change coherence state). Finally, the data is sent over a bidirectional *data bus* (DB) to the original requester. If there are multiple SBFs (e.g., connected by a P2P link), the address request will be broadcast to the other SBFs via that link, and a combined response from the remote SBF returned to the local one, to be merged with the local responses.

Note that the above transactions are quite standard for any shared memory multiprocessor implementing a snoopy write-invalidate coherence protocol [4].

3.1.2 Elements of the SBF

The composition of the SBF allows it to support all the coherence transactions mentioned above. We now discuss the primary buses, queues and logic that would typically be required for supporting these transactions. Figure 1 illustrates a typical SBF. Details of the modeled design, only some of which we have room to describe here, are based heavily on the shared bus fabric in the Power 5 multi-core architecture [16].

Each requester on the SBF interfaces with it via *request* and *data queues*. It takes at least one cycle to communicate information about the occupancy of the request queue to the requester. The request queue must then have at least two entries to maintain the throughput of one request every cycle. Similarly, all the units that can source data need to have data queues of at least two entries. Requesters connected to the SBF include cores, L2 and L3 caches, IO devices, memory controllers, and non-cacheable instruction units.

All requesters interface to the fabric through an arbiter for the address bus. The minimum latency through the arbiter depends on (1) the physical distance from the central arbiter to the most distant unit, and (2) the levels of arbitration. Caches are typically given higher priority than other units, so arbitration can take multiple levels based on priority. Distance is determined by the actual floorplan. Since the address bus is pipelined, the arbiter must account for the location of a requester on the bus in determining what cycle access is granted. Overhead of the arbiter includes control signals to/from the requesters, arbitration logic and some latches.

After receiving a grant from the central arbiter, the requester unit puts the address on the *address bus*. Each address request goes over the address bus and is then copied into multiple queues, corresponding to outgoing P2P links (discussed later) and to off-chip links. Being a broadcast bus, the address bus spans the width of the chip. There is also a local *snoop queue* that queues up the requests and participates in the arbitration for the local *snoop bus*. Every queue in the fabric incurs at least one bus cycle of delay. The minimum size of each queue in the interconnect (there are typically queues associated with each bus) depends on the delay required for the arbiter to stall further address requests if the corresponding bus gets stalled. Thus it depends on the distance and communication protocol to the device or queue responsible for generating requests that are sinked in the queue, and the latency of requests already in transit on the bus. We therefore compute queue size based on floorplan and distance.

The snoop bus can be shared, for example by off-chip links and other SBFs, so it also must be accessed via an arbiter, with associated delay and area overhead. Since the snoop queue is at one end of the address bus, the snoop bus must run in the opposite direction of the address bus, as shown in Figure 1. Each module connected to the snoop bus snoops the requests. Snooping involves comparing the request address with the address range allocated to that module (e.g., memory controllers) or checking the directory (tag array) for caches.

A response is generated after a predefined number of cycles by each snooper, and goes out over the *response bus*. The delay can be significant, because it can involve tag-array lookups by the caches, and we must account for possible conflicts with other accesses to the tag arrays. Logic at one end of the bidirectional response bus collects all responses and broadcasts a message to all nodes, directing their response to the access. This may involve sourcing the data, invalidating, changing coherence state, etc. Some responders can initiate a data transfer on a read request simultaneously with generating the snoop response, when the requested data is in appropriate coherence state. The responses are collected in queues. All units that can source data to the fabric need to be equipped with a data queue. A central arbiter interfacing with the data queues is needed to grant one of the sources access to the bus at a time.

Bidirectional data buses source data. They support two different data streams, one in either direction. Data bandwidth requirements are typically high.

It should be noted that designs are possible with fewer buses, and the various types of transactions multiplexed onto the same bus. However, that would require higher bandwidth (e.g., wider) buses to support the same level of traffic at the same performance, so the overheads are unlikely to change significantly. We assume for the purpose of this paper that only the above queues, logic and buses form a part of the SBF and

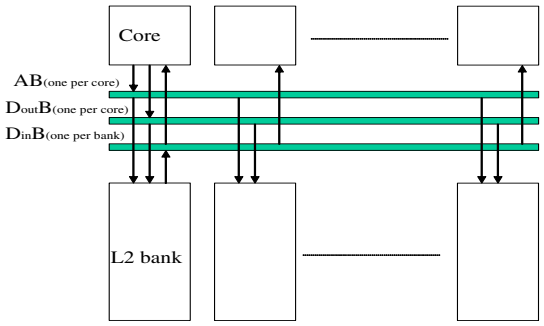

Figure 2. A typical crossbar

contribute to the interconnection latency, power, and area overheads.

3.2 P2P Links

If there are multiple SBFs in the system, the connection between the SBFs is accomplished using P2P links. Multiple SBFs might be required to increase bandwidth, decrease signal latencies, or to ease floorplanning (all connections to a single SBF must be on a line). For example, if a processor has 16 cores as shown in Figure 3, it becomes impossible to maintain die aspect ratio close to 1 unless there are two SBFs each supporting 8 cores.

Each P2P link should be capable of transferring all kinds of transactions (request/response/data) in both directions. Each P2P link is terminated with multiple queues at each end. There needs to be a queue and an arbiter for each kind of transaction described above.

3.3 Crossbar Interconnection System

The previous section assumed private L2 caches, with communication and coherence only occurring on L2 misses. However, if our architecture allows two or more cores to share L2 cache banks, a high bandwidth connection is required between the cores and the cache banks. This is typically accomplished by using a crossbar. It allows multiple core ports to launch operations to the L2 subsystem in the same cycle. Likewise, multiple L2 banks are able to return data or send invalidates to the various core ports in the same cycle.

The crossbar interconnection system consists of crossbar links and crossbar interface logic. A crossbar consists of address lines going from each core to all the banks (required for loads, stores, prefetches, TLB misses), data lines going from each core to the banks (required for writebacks) and data lines going from every bank to the cores (required for data reload as well as invalidate addresses). A typical implementation, shown in Figure 2, consists of one address bus per core from which all the banks feed. Each bank has one outgoing data bus from which all the cores feed. Similarly, corresponding to each write port of a core is an outgoing data bus that feeds all the banks.

Crossbar interface logic presents a simplified interface to the instruction fetch unit and the Load Store Unit in the cores. It typically consists of a load queue corresponding to each core sharing the L2. The load queue sends a request to the L2 bank appropriate to the request, where it is enqueued in a bank load queue (BLQ) (one per core for each bank to avoid conflict between cores accessing the same bank). The BLQs must arbitrate for the L2 tags and arrays, both among the BLQs, as well as with the snoop queue, the writeback queue, and the data reload queue — all of which may be trying to access the L2 at the same time. After L2 access (on a load request), the data goes through the reload queue, one per bank, and over the data bus back to the core. The above description of the crossbar interface logic is based on the crossbar implementation (also called core interface unit) in Power4 [15] and Power5 [16].

Note that even when the caches (or cache banks) are shared, an SBF is required to maintain coherence between various units in the CMP system.

4 Modeling Area, Power, and Latency

Both wires and logic contribute to interconnect overhead. This section describes our methodology for computing various overheads for 65nm technology. The scaling of overheads with technology as well as other design parameters is discussed in Section 8.

4.1 Wiring Area Overhead

We address the area overheads of wires and logic separately.

The latency, area, and power overhead of a metal wire depends on the metal layer used for this wire. Technology that we consider facilitates 10 layers of metal, 4 layers in 1X plane and 2 layers in the higher planes (2X, 4X and 8X) [17]. The 1X metal layers are typically used for macro-level wiring [17]. Wiring tracks in higher layers of metal are very expensive and only used for time-critical signals running over a considerable distance (several millimeters of wire).

We evaluate crossbar implementations for 1X, 2X and 4X metal planes where both data and address lines use the same metal plane. For our SBF evaluations, the address bus, snoop bus, and control signals always use the 8X plane. Response buses preferably use the 8X plane, but can use the 4X plane. Data buses can be placed in the 4X plane (as they have more relaxed latency considerations). All buses for P2P links are routed in the 8X plane.

The area occupied by a bus is determined by the number of wires times the effective pitch of the wires times the length. We account for the case where the area of one bus (partially) subsumes the area of some other bus in a different plane. When buses are wired without logic underneath, repeaters and latches are placed under the buses without incurring any additional area overhead. However, when interconnection buses are routed over array structures (e.g. cache arrays, directories etc.), we account for the fact that the sub-arrays (or memory macros) have to be shifted to make space for the placement of

Metal Plane	Pitch (μm)	Signal Wiring Pitch (μm)	Repeater Spacing (mm)	Repeater Width(μm)	Latch Spacing (mm)	Latch Height (μm)	Channel Leakage per repeater (uA)	Gate Leakage per repeater(uA)
1X	0.2	0.5	0.4	0.4	1.5	120	10	2
2X	0.4	1.0	0.8	0.8	3.0	60	20	4
4X	0.8	2.0	1.6	1.6	5.0	30	40	8
8X	1.6	4.0	3.2	3.2	8.0	15	80	10

Table 1. Design parameters for wires in different metal planes

wire repeaters and latches. In this case the minimal repeater and latch spacing is an essential parameter determining the area overhead. We believe that 4X and 8X wires can be routed over memory arrays. However, in section 7, we also evaluate routing 1X and 2X over memory, even though we believe it to be technologically more difficult.

Table 1 shows the signal wiring pitch for wires in different metal planes for 65nm. These pitch values are estimated by conforming to the considerations mentioned in [29].

The table also shows the minimum spacing for repeaters and latches as well as their heights for computing the corresponding area overheads. We model the height of the repeater macro to be 15 μm. The height of the latch macro given in the table includes the overhead of the local clock buffer and local clock wiring, but excludes the overhead of rebuffering the latch output which is counted separately. The values in Table 1 are for a bus frequency of 2.5 GHz and a bus voltage of 1.1 V. Analysis for different bus frequencies can be found in Section 8.

4.2 Logic Area Overhead

Area overhead due to interconnection-related logic comes primarily from queues. Queues are assumed to be implemented using latches. We estimate the area of a 1-bit latch used for implementing the queues to be $115\ \mu m^2$ for 65nm technology [30]. This size includes the local clock driver and the area overhead of local clock distribution. We also estimated that there is 30% overhead in area due to logic needed to maintain the queues (such as head, tail pointers, queue bypass, overflow signaling, request/grant logic, etc.) [6].

The interconnect architecture can typically be designed such that buses run over interconnection-related logic. The area taken up due to wiring is usually big enough that it (almost) subsumes the area taken up by the logic.

Because queues overwhelmingly dominate the logic area, we ignore the area (but not latency) of multiplexors and arbiters. It should be noted that the assumed overheads can be reduced by implementing queues using custom arrays instead of latches.

4.3 Power

Power overhead comes from wires, repeaters, and latches. For calculating dynamic dissipation in the wires, we optimistically estimate the capacitance per unit length of wire (for all planes) to be 0.2 pF/mm [14]. Repeater capacitance is assumed to be 30% of the wire capacitance [1]. The dynamic power per latch is estimated to be 0.05 mW per latch for 2.5 GHz at 65 nm [30]. This includes the power of the local clock buffer and the local clock distribution, but does not include rebuffering that typically follows latches.

Total dynamic power of a bus would depend on utilization of the bus as well as the efficacy of clock gating. We assume that in 30% of the unused cycles the latches will be gated off, to be consistent with clock gating efficiencies typically quoted for high-end microprocessors [6]. Even though the cycles of inactivity are easier to predict in the fabric than in the core, the physical distance between latches of the neighboring clock stages is much larger in the fabric, which complicates the timing of the clock gate signals.

Repeater leakage is computed using the parameters given in Table 1. For latches, we estimate channel leakage to be 20uA per bit in all planes (again not counting the repeaters following a latch). Gate leakage for a latch is estimated to be 2uA per bit in all planes [1]. For computing dynamic and leakage power in the queues, we use the same assumptions as for the wiring latches.

4.4 Latency

The latency of a signal traveling through the interconnect is primarily due to wire latencies, wait time in the queues for access to a bus, arbitration latencies, and latching that is required between stages of interconnection logic. Latency of wires is determined by the spacing of latches. Spacing between latches for wires is given in Table 1.

Arbitration can take place in multiple stages (where each stage involves arbitration among the same priority units) and latching needs to be done between every two stages. For 65 nm technology, we estimate that no more than four units can be arbitrated in a cycle. The latency of arbitration also comes from the travel of control between a central arbiter and the interfaces corresponding to request/data queues. Other than arbiters, every time a transaction has to be queued, there is at least a bus cycle of delay — additional delays depend on the utilization of the outbound bus.

5 Modeling multi-core architectures

For this study, we consider a stripped version of out-of-order Power4-like cores [15]. We determine the area taken up by such a core at 65nm to be $10mm^2$. The area and power determination methodology is similar to the one presented in [19]. The power taken up by the core is determined to be 10W, including leakage.

For calculating on-chip memory sizes, we use the Power5 cache density, as measured from die photos [16], scaled to 65nm. We determine it to be 1 bit per square micron, or

$0.125MB/mm^2$. For the purpose of this study, we consider L2 caches as the only type of on-chip memory. We do not assume off-chip L3 cache, but in 65nm systems, it is likely that L3 chips would be present as well (the number of L3 chips would be limited, however, due to the large number of pins that every L3 chip would require), but we account for that effect using somewhat optimistic estimates for effective bandwidth and memory latency. Off-chip bandwidth was modeled carefully based on pincount [1] and number of memory channels (Rambus RDRAM interface was assumed).

Our models include the effects of not only L2 banks and memory controllers, but also DMA controllers and Non-cacheable Instruction Units (NCUs) on the die that can generate transactions. NCUs handle instructions like syncs, eieios (enforce inorder execution of I/Os), TLB invalidates, partial read/writes, etc., that are not cached and are directly put on the fabric. Each core has a corresponding NCU. The assumptions about these units are taken from Power 4 and Power 5 [15, 16] designs.

We simulate a MESI-like [24, 15] coherence protocol, and all transactions required by that protocol are faithfully modeled in our simulations. We also model weak consistency [8] for the multiprocessor, so there is no impact on CPI due to the latency of stores and writebacks.

For performance modeling, we use a combination of detailed functional simulation and queuing simulation tools [21]. The functional simulator is used for modeling the memory subsystem as well as the interconnection between modules. It takes instruction traces from a SMP system as input and generates coherence statistics for the modeled memory/interconnect subsystem. The queueing simulator takes as input the modeled subsystem, its latencies, coherence statistics and the inherent CPI of the modeled core assuming perfect L2. It then generates the CPI of the entire system, accounting for real L2 miss rates and real interconnection latencies. Traffic due to syncs, speculation, and MPL (message passing library) effects is accounted for as well. The tools and our interconnection models have been validated against a real, implemented design.

The cache access times are calculated using assumptions similar to those made in CACTI [28]. Memory latency is set to 500 cycles. The average CPI of the modeled core over all the workloads that we use, assuming perfect L2, is measured to be 2.65. Core frequency is assumed to be 5GHz for the 65nm studies. Buses as well as the L2 are assumed to be clocked at half the CPU speed.

5.1 Workload

All our performance evaluations have been done using commercial workloads, including TPC-C, TPC-W, TPC-H, Notesbench and others further described in [21]. The server workloads used represent such market segments as on-line transaction processing (OLTP), business intelligence, enterprise resource planning, web serving, and collaborative groupware. These applications are large and function rich; they use a large number of operating system services and access large databases. These characteristics make the instruction and data working sets large. These workloads are also inherently multiuser and multitasking, with frequent read-write sharing.

We use PowerPC instruction and data reference traces of the workloads running under AIX. The traces are taken in a non-intrusive manner by attaching a hardware monitor to a processor [9, 21]. This enables the traces to be gathered while the system is fully loaded with the normal number of users, and captures the full effects of multitasking, data sharing, interrupts, etc. These traces contain even DMA instructions and non-cacheable accesses.

6 Shared Bus Fabric Overheads, Performance, and Design Considerations

This section examines the various overheads of the shared bus fabric, and the implications this has for the entire multi-core architecture. We examine floorplans for several design points, and characterize the impact on the overall design and performance of the processor of the area, power, and latency overheads. This section demonstrates that the overheads of the SBF can be quite significant. It also illustrates the tension between the desire to have more cores, more cache, and more interconnect bandwidth, and how that plays out in total performance.

In this section, we assume private L2 caches and that all the L2s (along with NCUs, memory controllers, and IO Devices) are connected using a shared bus fabric. We consider architectures with 4, 8, and 16 cores. Total die area is assumed to be constant at $400mm^2$ due to yield considerations. Hence, the amount of L2 per core decreases with increasing number of cores. For 4, 8 and 16 cores, we evaluate multiple floorplans and choose those that maximized cache size per core while maintaining a die aspect ratio close to 1. In the default case, we consider the width of the address, snoop, response and data buses of the SBF to be 7, 12, 8, 38 (in each direction) bytes respectively — these widths are determined such that no more than 0.15 requests get queued up, on average, for the 8 core case. We also evaluate the effect of varying bandwidths. We can lay out 4 or 8 cores with a single SBF, but for 16 cores, we need two SBFs connected by a P2P link. In that case, we model two half-width SBFs and a 76 byte wide P2P link. Figure 3 shows the floorplans arrived at for the three cases. The amount of L2 cache per core is 8MB, 3MB and 0.5MB for 4, 8 and 16 core processors respectively. It must be mentioned that the 16-core configuration is somewhat unrealistic for this technology as it would result in inordinately high power consumption. However, we present the results here for completeness reasons.

Wires are slow and hence cannot be clocked at very high speeds without inserting an inordinately large number of latches. For our evaluations, the SBF buses are cycled at half the core frequency.

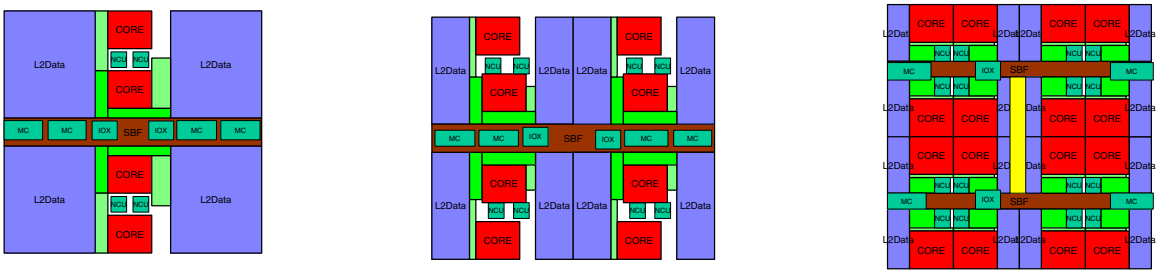

Figure 3. Floorplans for 4, 8 and 16 core processors

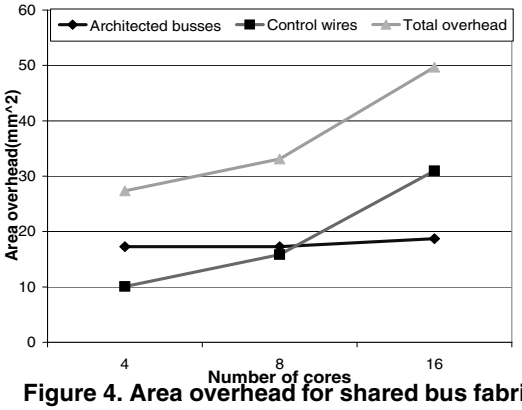

Figure 4. Area overhead for shared bus fabric.

6.1 Area

The area consumed by the shared bus fabric comes from wiring and interconnection-related logic, as described in Section 4. Wiring overhead depends on the architected buses and the control wires that are required for flow control and arbitration. Control wires are needed for each multiplexor connected to the buses and signals to and from every arbiter. Flow control wires are needed from each queue to the units that control traffic to the queue.

Since data buses run in the 4X plane, the area taken up by (76-byte) data buses is calculated as $2\mu m \times 76 \times 8 \times length$, which results in $24.32mm^2$. Address (7 bytes), snoop (12 bytes), response buses (8 bytes) as well as control wires (at least 198 of them for the stated control architecture for 8 cores) that run in the 8X plane can be routed over the data buses. In this case, the area overhead of the data buses is subsumed by that of the remaining buses. For 16 cores, the P2P links do not result in direct area overhead because they can be routed over L2 caches. However, reduction in the cache density (due to bus latches and repeaters) does result in an area overhead.

Figure 5 shows the wiring area overhead for various processors. The graph shows the area overhead due to architected wires, control wires, and the total. We see that area overhead due to interconections in a CMP environment can be significant. For the assumed die area of $400mm^2$, area overhead for the interconnect with 16 cores is 13%. Area overhead for 8 cores and 4 cores is 8.7% and 7.2% of the die area, respectively. Considering that each core is $10mm^2$, the area taken up by the SBF is sufficient to place 3-5 extra cores, or 4-6 MB of extra cache.

Number of cores	4	8	16
Number of data queue latches	28672	45056	92160
Number of request queue latches	3136	4928	8512
Number of snoop queue latches	336	336	896
Number of latches for response bus queue	1680	1680	6720
Total number of latches	33824	52000	108288
Area(in mm^2)	5.6	8.6	17.94

Table 2. Logic overhead

The graph also shows that area overhead increases quickly with the number of cores. This result assumes constant width architected buses, even when the number of cores is increased. If the effective bandwidth per core is kept constant, overhead would increase even faster.

The overhead due to control wires is high. Control takes up at least 37% of SBF area for 4 cores and at least 62.8% of the SBF area for 16 cores. This is because the number of control wires grows linearly with the number of connected units, in addition to the linear growth in the average length of the wires. Reducing SBF bandwidth does not reduce the control area overhead, thus it constrains how much area can be regained with narrower buses. Note that this argues against very lightweight (small, low performance) cores on this type of chip multiprocessor, because the lightweight core does not amortize the incremental cost to the interconnect of adding each core.

Interconnect area due to logic is primarily due to the various queues, as described in Section 4. Table 2 shows the area overhead due to interconnect-related logic and the corresponding breakdown. The area taken up by interconnection-related logic increases superlinearly with the number of connected units (note that the number of connected units is 14, 22 and 38 respectively for 4, 8 and 16 core processors). When going from 8 to 16 cores, the logic-area overhead jumps because the queues are required to support two SBFs and a P2P link. Note, however, that the logic can typically be placed underneath the SBF wires. Thus, under these assumptions the SBF area is dominated by wires, but only by a small amount.

6.2 Power

The power dissipated by the interconnect is the sum of the power dissipated by wires and the logic. Figure 5 shows a breakdown of the total power dissipation by the interconnect.

The graph shows that total power due to the interconnect can be significant. The interconnect power overhead for the 16-core processor is more than the combined power of two cores. It is equal to the power dissipation of one full core even for

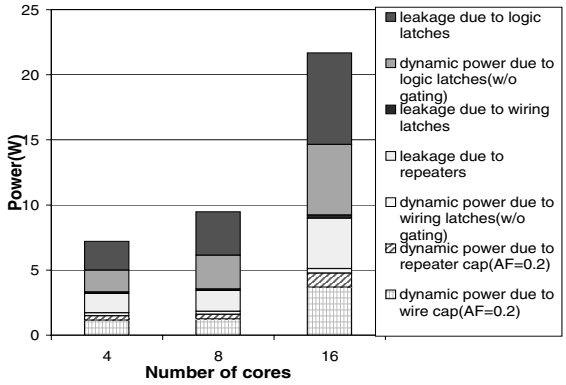

Figure 5. Power overhead for shared bus fabric.

the 8 core processor. Power increases superlinearly with the number of connected units. This is because of the (at least linear) increase in the number of control wires as well as the (at least linear) increase in the number of queuing latches. There is also a considerable increase in the bus traffic with the growing number of cores. Half the power due to wiring is leakage (mostly from repeaters).

Contrary to popular belief, interconnect power is not always dominated by the wires. The power due to logic can be, as in this case, more than the power due to wiring.

6.3 Performance

For the chip multiprocessor consisting of four cores, the end to end latency of the architected buses, except the data buses, would be 6 processor cycles (as three latches would be required for 20mm wires running in the 8X plane). Similarly, the latency of the data buses (routed in the 4X plane) would be 12 processor cycles.

Address arbitration would take 8 processor cycles, but the request transfer between the central address arbiter and the request queues would take 8 processor cycles (two latches each way – we assume central arbiters to be placed around the middle of the chip). Every queue would take at least one bus cycle. Snoop response generation will take at least 20 processor cycles (cache tag access and two latches). Generating a final response will take 4 cycles (two latches). We also estimate that an 8MB cache can be accessed in 46 cycles (includes array access time, time to go through queues, arbitration overhead for getting access to the array, etc.; note that L2 is assumed to be cycled at half the core frequency). Accounting for all arbitration, bus latencies, queues, and the cache access, the minimum (no contention) latency for a load for four cores is 124 cycles. Even under these optimistic assumptions, the interconnect accounts for over half the total latency to the L2 cache.

Figure 6 shows the per-core performance for 4, 8 and 16 core architectures both assuming no interconnection overhead (zero latency interconnection) and with interconnection overheads modeled carefully. Single-thread performance (even assuming no interconnection overhead) goes down as the number

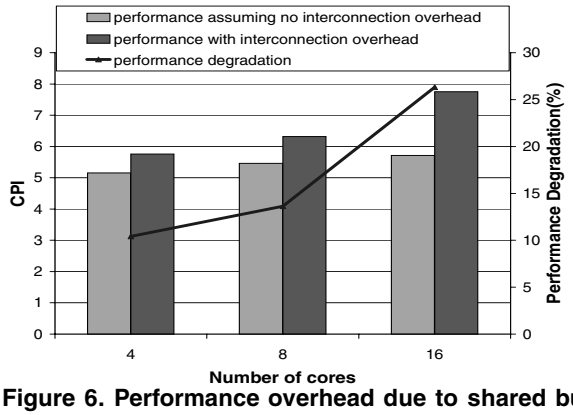

Figure 6. Performance overhead due to shared bus fabric.

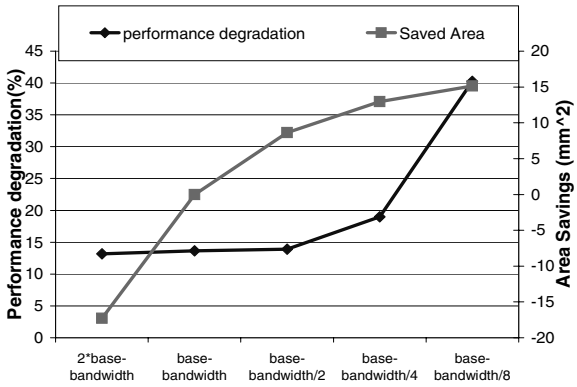

Figure 7. Trading off interconnection bandwidth with area.

of cores increases due to the reduced cache size per core. If interconnect overhead is considered, then the performance decreases much faster. In fact, performance overhead due to interconnection is more than 10% for 4 cores, more than 13% for 8 cores and more than 26% for 16 cores.

In results to this point, we keep bus bandwidth constant. In Figure 7, we show the single-thread performance of a core in the 8 core processor case, when the width of the architected buses is varied by factors of two. The graph also shows the real estate saved compared to the baseline. We see that with wide buses, the area costs are significant, and the incremental performance is minimal. On the other hand, with narrow buses, the area saved by small changes in bandwidth is small, but the performance impact is significant.

Alternatively, we could put that saved area to use. We ran simulations that assume that we put that area back into the caches. We find that over certain ranges, if the bandwidth is reduced by small factors, the performance degradation can be recovered using bigger caches. For example, decreasing the bandwidth by a factor of 2 decreases the performance by 0.57%. But it saves $8.64mm^2$. This can be used to increase the per-core cache size by 135KB. When we ran simulations using new cache sizes, we observed a performance improvement of

0.675%. Thus, *we can decrease bus bandwidth and improve performance* (if only by small margins in this example), because the resulting bigger caches protect the interconnect from a commensurate increase in utilization. On the other hand, when bandwidth is decreased by a factor of 8, performance decreases by 31%, while the area it saves is $15.12mm^2$. The area savings is sufficient to increase per core cache size by only 240KB. The increase in cache size was not sufficient to offset the performance loss in this case. Similarly, when doubling interconnect bandwidth over our baseline configuration, total performance decreased by 1.2% due to the reduced cache sizes.

This demonstrates the importance of co-designing the interconnect and memory hierarchy. It is neither true that the biggest caches nor the widest interconnect give the best performance; designing each of these subsystems independently is unlikely to result in the best design. Similarly, the core itself should be co-architected with the caches and interconnect, but for this study we treat the cores as a constant.

7 Shared Caches and the Crossbar

The previous section presented evaluations with private L1 and L2 caches for each core, but many proposed chip multiprocessors have featured shared L2 caches, connected with crossbars. Shared caches allow the cache space to be partitioned dynamically rather than statically, typically improving overall hit rates. Also, shared data does not have to be duplicated. To fully understand the tradeoffs between private and shared L2 caches, however, we find that it is absolutely critical that we account for the impact of the interconnect.

7.1 Area and power overhead

The crossbar, shown in Figure 2, connects cores (with L1 caches) to the shared L2 banks. The data buses are 32 bytes while the address bus is 5 bytes. Lower bandwidth solutions were found to adversely affect performance and render sharing highly unfruitful. In this section we focus on an 8-core processor with 8 cache banks, giving us the options of 2-way, 4-way, and full (8-way) sharing of cache banks. Crossbar wires can be implemented in the 1X, 2X or 4X plane. For almost 2x reduction in the latency, the wire thickness doubles every time we go to a higher metal plane.

Figure 8 shows the area overhead for implementing different mechanisms of cache sharing. The area overhead is shown for two cases – one where the crossbar runs between cores and L2 and the other where the crossbar can be routed over L2. When the crossbar is placed between the L2 and the cores, interfacing is easy, but all wiring tracks result in area overhead. When the crossbar is routed over L2, area overhead is only due to reduced cache density to accommodate repeaters and latches. However, the implementation is relatively complex as vertical wires are needed to interface the core with the L2. We show the results assuming that the L2 density is kept uniform (i.e. even if repeaters/latches are dropped only over the top region of the cache, sub-arrays are displaced even in the other regions to maintain uniform density).

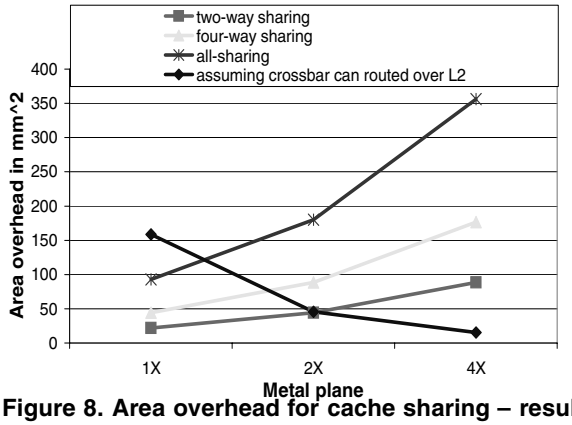

Figure 8. Area overhead for cache sharing – results for crossbar routed over L2 assume uniform cache density.

Cache sharing carries a heavy area overhead. If the total die-area is around $400\ mm^2$, then the area overhead for an acceptable latency (2X) is 11.4% for 2-way sharing, 22.8% for four-way sharing and 46.8% for full sharing (nearly half the chip!). Overhead increases as we go to higher metal layers due to increasing signal pitch values. When we assume that the crossbar can be routed over L2, area overhead is still substantial; however, in that case it improves as we move up in metal layers. At low levels the number of repeater/latches, which must displace cache, is highest.

The point of sharing caches is to get the effect of having more cache space. In this case, the cores can gain significant real cache space by *foregoing sharing*, raising doubts about whether sharing has any benefit. This is seen even more clearly in the next subsection.

The high area overhead again suggests that issues of interconnect/cache/core codesign must be considered. For crossbars sitting between cores and L2, just two-way sharing results in an area overhead equivalent to more than the area of two cores. Four-way sharing results in an area overhead of 4 cores. An 8-way sharing results in an area overhead of 9 cores. If the same area were devoted to caches, one could instead put 2.75 MB, 5.5 MB and 11.6 MB of extra caches, respectively.

Figure 9 shows the corresponding power overhead. A breakdown is also provided for various sources of power dissipation. The graph shows that power overhead due to crossbars is very significant. The overhead can be more than the power taken up by three full cores for a completely shared cache and more than the power of one full core for 4-way sharing. Even for 2-way sharing, power overhead is more than half the power dissipation of a single core. Hence, even if power is the primary constraint, the benefits of the shared caches must be weighed against the possibility of more cores or significantly more cache.

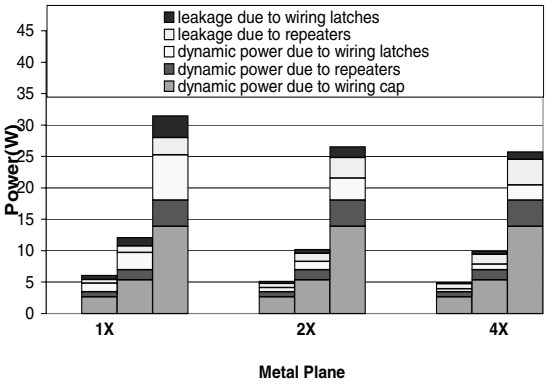

Figure 9. Power overhead for cache sharing (the three bars, left to right, correspond to 2-way, 4-way and full sharing).

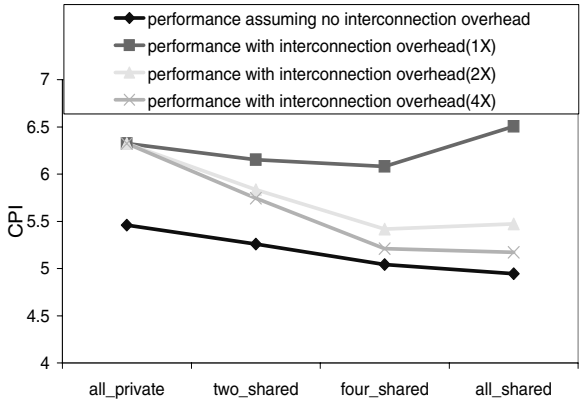

Figure 10. Evaluating cache sharing for a fixed cache size for different crossbar implementations – no area overhead is assumed

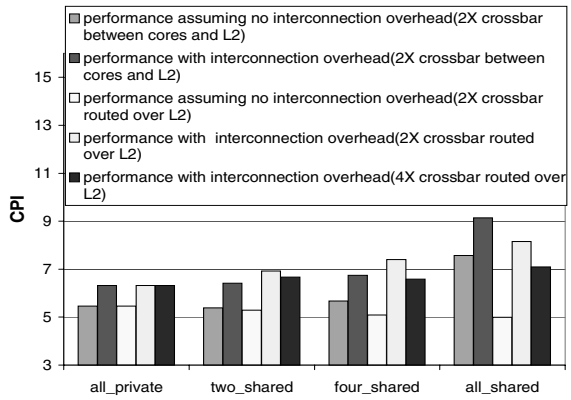

Figure 11. Evaluating cache sharing for a fixed die-area – area overhead taken into account

Leakage is a smaller fraction of the total power for crossbars than for SBFs; however, it is still significant — 18-20% depending on the metal layer.

7.2 Performance

Because of the high area overhead for cache sharing, the total amount of on-chip caches decreases with sharing. We performed our evaluations for the most tightly packed floorplans that we could find for 8-core processors with different levels of sharing. When the crossbar wires are assumed to be routed in the 2X plane between cores and L2, total cache size is 20MB, 14MB and 4MB respectively for 2-way, 4-way and full sharing. When crossbar is assumed to be routed over L2 (and assuming unifor cache density), the total cache size was 22MB for 4X and 18.2MB for 2X. We also conducted experiments assuming no crossbar area overhead to isolate the benefit of sharing. Figure 10 shows results for a fixed on-chip cache size (i.e. assuming no crossbar area overhead – crossbar latency overhead is assumed, however). Figure 11 presents the results for a fixed die area and cache sizes varied accordingly (i.e. taking into account crossbar area overhead).

Figure 10 shows that cache sharing, in general, results in higher performance than just having private caches if interconnection area overheads are not considered. It also shows that crossbar performance is very sensitive to the metal plane used to implement it.

Figure 11, assumes a constant die area and considers interconnection area overhead. It shows that performance, even without considering the interconnection latency overhead (and hence purely the effect of cache sharing), either does not improve or improves only by a slight margin. This is due to reduced size of on-chip caches to accommodate the crossbar. If interconnect latencies are accounted for (higher sharing means longer crossbar latencies), sharing degrades performance even between two cores. Note that in this case, the conclusion reached ignoring interconnect area effects is opposite that reached when those effects are considered.

Note that performance loss due to increased L2 hit latency can be mitigated by using L2 latency hiding techniques, like overlapping of L2 accesses or prefetching. Also, crossbar area overhead can be reduced (and hence performance improved) by implementing caches with non-uniform density. In fact, we observed that two-way and four-way sharing improves performance for the 4X crossbar implementation if the crossbar is routed over memory and the L2 is allowed to have non-uniform density. Sharing might also result in benefit for other workloads with different working set and sharing behavior. Also, the smaller the relative frequency of the bus, the less prohibitive it is to implement L2 with large-scale sharing. For example, if the crossbars are routed over L2 and are clocked at one-fourth the core frequency, we observe that two-way and four-way sharing improves performance on the 4X metal plane.

However, our results definitely show that having shared caches becomes *significantly less desirable* than previously accepted if interconnection overheads are considered. We believe that the conclusion holds, in general, for uniform access time caches and calls for evaluation of caching strategies

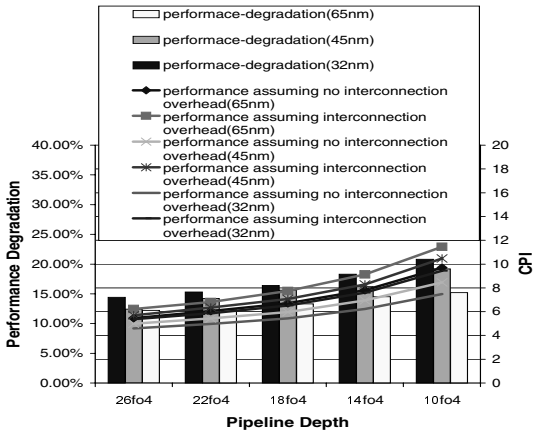

Figure 12. Scaling of interconnection overhead with pipelining and technology

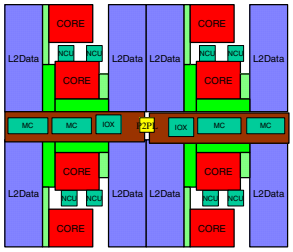

Figure 13. Hierarchical approach (splitting SBFs)

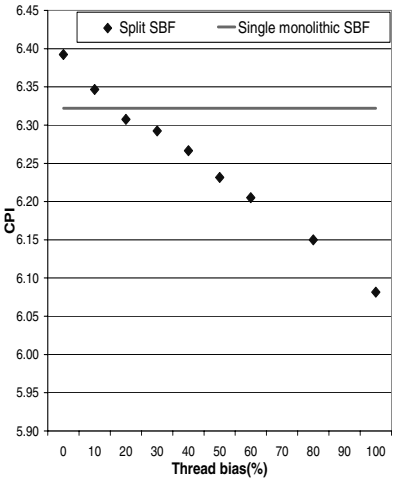

Figure 14. Split vs Monolithic SBF

with careful consideration of interconnect overheads. Further analysis needs to be done for intelligent NUCA (non-uniform cache access) caches [18].

8 Scaling with Technology Parameters

All results to this point in the paper have been carefully parameterized to the 65 nm process. This section extends those results to future technologies, and also considers the effect of deeper pipelining (i.e., faster CPU clocks).

As technology shrinks, the repeater and latch spacings decrease, thereby increasing the latency overhead due to wires. Going to deeper pipelines/faster clocks also decreases repeater and latch spacings, as well as increased logic overhead (in terms of cycles). Technology scaling and deeper pipelines also increase perceived memory and cache access times. We used ITRS scaling trends to compute the latencies for 45nm and 32nm technologies. Figure 12 shows the results for a fixed die area ($400\ mm^2$) for 65nm, 45nm and 32nm. The results are shown for the 8-core case. Each core has a private cache of 3MB, 6MB and 10MB respectively for the three technologies. We assume the base CPI of the core (apart from memory behavior) to remain the same for this study.

Figure 12 shows that with deeper pipelines for the same technology, the interconnection overhead on performance increases. For example at 65 nm, if the pipeline depth is changed from 26FO4 to 10FO4, the interconnection overhead increases by 30%. But for 45 nm, the corresponding change is 55%, and for 32nm, 44.4%. Also, for the same die area, and for the same pipeline depth, interconnection overhead increases with technology. Overhead increase is due to increased latencies, increased arbitration overhead, etc. Note that increased cache size for better technologies leads to higher hit rates as well as reduced traffic on the interconnect, but *it is not sufficient to hide the effect of the increased interconnect latencies*. It must be mentioned, however, that overall performance improves because of higher frequencies.

9 Reducing interconnect latency – an alternate architecture

The intent of this section is to apply one lesson learned from the high volume of data gathered in this research. Our interconnect architectures to this point were highly driven by layout. The SBF spans the width of the chip, allowing us to connect as many units as possible in a straight line across the chip. However, the latency overheads of a long SBF encourage us to consider alternatives. This section describes a more hierarchical approach to interconnects, which can exploit shorter buses with shorter latencies when traffic remains local. We will be considering the 8-core processor again.

The effectiveness of such an approach will depend on the probability that an L2 miss is serviced on a local cache (an L2 connected to the same SBF), rather than a cache on a remote SBF. We will refer to this probability as *"thread bias"*. A workload with high thread bias means that we can identify and map "clusters" of threads that principally communicate with each other on the same SBF.

In this section, we split the single SBF that spanned the chip vertically into two SBFs, with a P2P link between them. Local accesses benefit from decreased distances. Remote accesses suffer because they travel the same distances and see additional queueing and arbitration overheads between interconnects.

Figure 14 shows the performance of the split SBF for various thread bias levels. The SBF is split vertically into two, such that each SBF piece now supports 4 cores, 4 NCUs, 2 memory controllers and 1 IO Device. The X-axis shows the thread bias in terms of the fraction of misses satisfied by an L2 connected to the same SBF. A 25% thread bias means that one out of four L2 misses are satisfied by an L2 connected to the same SBF piece. These results are obtained through statistical simulation by synthetically skewing the traffic pattern.

The figure also shows the system performance for a a single monolithic SBF (the one used in previous sections). As can be

seen, if thread bias is more than 17%, the performance of split SBF can overtake performance of a monolithic SBF. Note that 17% is lower than the statistical probability of locally satisfying an L2 miss assuming uniform distribution (3/7). Hence, the split SBF, in this case, is always a good idea.

10 Summary and Conclusions

This paper presents the results of a detailed modeling of the impact of the interconnection fabric on a hypothetical chip multiprocessor. These results show that the architecture of the interconnect interacts with the design and architecture of the cores and caches to a much greater degree than conventional off-chip interconnections. Thus, any design that hopes to achieve high performance or even energy efficiency needs to be the result of a careful co-design of all three elements.

This study shows several examples of this need for co-design. The interconnect fabric itself is large and power-hungry, consuming resources that would otherwise be available for more cores and caches. The interconnect, even without the sharing of L2 caches, can take the area of three cores and the power of one.

We show examples where decreasing interconnection bandwidth can improve performance, due to the constrained window on total resources. In the same way, large caches can also decrease performance when they constrain the interconnect to too small an area.

We also show that while it is generally believed that shared L2 caches improve cache hit rates, we show that the implications on the interconnect are extreme. For example, sharing four caches among four cores can require a quarter of the chip area just for the crossbar network. When accounting for the area overheads and the latency of the long interconnect, the desirability of shared L2 caches is significantly lower than is assumed if the interconnect is not accounted for.

Acknowledgments

The authors would like to thank the anonymous reviewers as well as Margaret Martonosi, Ravi Nair, Norman Jouppi, Alper Buyuktosunoglu, Pradip Bose, and Eric Tune for their useful comments. This work was supported in part by NSF Grant No. CCR-0311683 and an IBM internship.

References

[1] International Technology Roadmap for Semiconductors 2003, http://public.itrs.net.

[2] Butterfly parallel processor overview. In *BBN Report No 6148,*, Mar. 1986.

[3] A. Agarwal, J. Kubiatowicz, D. Kranz, B.-H. Lim, D. Yeung, G. D'Souza, and M. Parkin. Sparcle: An evolutionary processor design for large-scale multiprocessors. *IEEE Micro*, June 1993.

[4] J. Archibald and J.-L. Baer. Cache coherence protocols: evaluation using a multiprocessor simulation model. *ACM Trans. Comput. Syst.*, 4(4):273–298, 1986.

[5] L. 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 *ISCA-27*, 2000.

[6] J. Clabes, J. Friedrich, M. Sweet, J. DiLullo, S. Chu, D. Plass, J. Dawson, P. Muench, L. Powell, M. Floyd, B. Sinharoy, M. Lee, M. Goulet, J. Wagoner, N. Schwartz, S. Runyon, G. Gorman, P. Restle, R. Kalla, J. McGill, and S. Dodson. Design and implementation of the power5 microprocessor. In *ISSCC*, 2004.

[7] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *DAC-38*, pages 684–689, 2001.

[8] M. Dubois, C. Scheurich, and F. Briggs. Synchronization, coherence, and event ordering in multiprocessors. *IEEE Computer*, 21(2), 1988.

[9] R. J. Eickemeyer, R. E. Johnson, S. R. Kunkel, M. S. Squillante, and S. Liu. Evaluation of multithreaded uniprocessors for commercial application environments. In *ISCA-23*, 1996.

[10] S. J. Frank. Tightly coupled multiprocessor systems speed memory access times. In *Electron*, Jan. 1984.

[11] D. Gajski, D. Kuck, D. Lawrie, and A. Sameh. Cedar - a large scale multiprocessor. In *ICPP*, Aug. 1983.

[12] L. Hammond, B. A. Nayfeh, and K. Olukotun. A single-chip multiprocessor. *IEEE Computer*, 30(9), 1997.

[13] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist. Network on chip: An architecture for billion transistor era. In *IEEE NorChip Conference*, Nov. 2000.

[14] M. Horowitz, R. Ho, and K. Mai. The future of wires. 1999.

[15] IBM. Power4:http://www.research.ibm.com/power4.

[16] IBM. Power5: Presentation at microprocessor forum. 2003.

[17] C. Kaanta, W. Cote, J. Cronin, K. Holland, P. Lee, and T. Wright. Submicron wiring technology with tungsten and planarization. In *Fifth VLSI Multilevel Interconnection Conference*, 1988.

[18] C. Kim, D. Burger, and S. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *ASPLOS*, 2002.

[19] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction. In *MICRO-36*, Dec. 2003.

[20] R. Kumar, V. Zyuban, and D. Tullsen. Exploring interconnections in multi-core architectures. Technical report, University of California, San Diego, 2005.

[21] S. Kunkel, R. Eickemeyer, M. Lipasti, T. Mullins, B. Krafka, H. Rosenberg, S. VanderWiel, P. Vitale, and L. Whitley. A performance methodology for commercial servers. In *IBM Journal of R&D*, Nov. 2000.

[22] D. Lenoski, J. Laudon, K. Gharachorloo, W. Weber, A. Gupta, J. Henessy, M. Horowitz, and M. Lam. The stanford DASH multiprocessor. In *IEEE Computer*, 1992.

[23] T. Lovett and S. Thakkar. The symmetry multiprocessor system. In *ICPP*, Aug. 1988.

[24] M. Papamarcos and J. Patel. A low overhead coherence solution for multiprocessors with private cache memories. In *ISCA-15*, 1988.

[25] L.-S. Peh. Flow control and microarchitectural mechanisms for extending the performance of interconnection networks. PhD Thesis, Stanford University, 2001.

[26] G. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, , and J. Weiss. The IBM Research Parallel Processor prototype (RP3): Introduction and Architecture. In *ICPP*, Aug. 1985.

[27] C. L. Seitz. The cosmic cube. In *Communications of ACM*, 1985.

[28] P. Shivakumar and N. Jouppi. CACTI 3.0: An integrated cache timing, power and area model. In *Technical Report 2001/2, Compaq Computer Corporation*, Aug. 2001.

[29] T. N. Theis. The future of interconnection technology. In *IBM Journal of R&D*, May 2000.

[30] J. Warnock, J. Keaty, J. Petrovick, J. Clabes, C. Kircher, B. Krauter, P. Restle, B. Zoric, and C. Anderson. The circuit and physical design of the Power4 microprocessor. In *IBM Journal of R&D*, Jan. 2002.

[31] A. Wilson. Hierarchical cache/bus architecture for shared memory multiprocessors. In *ISCA-14*, June 1987.