Interconnect-Aware Coherence Protocols for Chip Multiprocessors

Liqun Cheng, Naveen Muralimanohar, Karthik Ramani, Rajeev Balasubramonian, John B. Carter
School of Computing, University of Utah
{*legion,naveen,karthikr,rajeev,retrac*}*@cs.utah.edu* [*]

Abstract

Improvements in semiconductor technology have made it possible to include multiple processor cores on a single die. Chip Multi-Processors (CMP) are an attractive choice for future billion transistor architectures due to their low design complexity, high clock frequency, and high throughput. In a typical CMP architecture, the L2 cache is shared by multiple cores and data coherence is maintained among private L1s. Coherence operations entail frequent communication over global on-chip wires. In future technologies, communication between different L1s will have a significant impact on overall processor performance and power consumption. On-chip wires can be designed to have different latency, bandwidth, and energy properties. Likewise, coherence protocol messages have different latency and bandwidth needs. We propose an interconnect composed of wires with varying latency, bandwidth, and energy characteristics, and advocate intelligently mapping coherence operations to the appropriate wires. In this paper, we present a comprehensive list of techniques that allow coherence protocols to exploit a heterogeneous interconnect and evaluate a subset of these techniques to show their performance and power-efficiency potential. Most of the proposed techniques can be implemented with a minimum complexity overhead.

1. Introduction

One of the greatest bottlenecks to performance in future microprocessors is the high cost of on-chip communication through global wires [19]. Power consumption has also emerged as a first order design metric and wires contribute up to 50% of total chip power in some processors [32]. Most major chip manufacturers have already announced plans [20, 23] for large-scale chip multi-processors (CMPs). Multi-threaded workloads that execute on such processors will experience high on-chip communication latencies and will dissipate significant power in interconnects. In the past, only VLSI and circuit designers were concerned with the layout of interconnects for a given architecture. However, with *communication* emerging as a larger power and performance constraint than *computation*, it may become necessary to understand and leverage the properties of the interconnect at a higher level. Exposing wire properties to architects enables them to find creative ways to exploit these properties. This paper presents a number of techniques by which coherence traffic within a CMP can be mapped intelligently to different wire implementations with minor increases in complexity. Such an approach can not only improve performance, but also reduce power dissipation.

In a typical CMP, the L2 cache and lower levels of the memory hierarchy are shared by multiple cores [24, 41]. Sharing the L2 cache allows high cache utilization and avoids duplicating cache hardware resources. L1 caches are typically not shared as such an organization entails high communication latencies for every load and store. There are two major mechanisms used to ensure coherence among L1s in a chip multiprocessor. The first option employs a bus connecting all of the L1s and a snoopy bus-based coherence protocol. In this design, every L1 cache miss results in a coherence message being broadcast on the global coherence bus and other L1 caches are responsible for maintaining valid state for their blocks and responding to misses when necessary. The second approach employs a centralized directory in the L2 cache that tracks sharing information for all cache lines in the L2. In such a directory-based protocol, every L1 cache miss is sent to the L2 cache, where further actions are taken based on that block's directory state. Many studies [2, 10, 21, 26, 30] have characterized the high frequency of cache misses in parallel workloads and the high impact these misses have on total execution time. On a cache miss, a variety of protocol actions are initiated, such as request messages, invalidation messages, intervention messages, data block writebacks, data block transfers, etc. Each of these messages involves on-chip communication with latencies that are projected to grow to tens of cycles in future billion transistor architectures [3].

Simple wire design strategies can greatly influence a wire's properties. For example, by tuning wire width and spacing, we can design wires with varying latency and bandwidth properties. Similarly, by tuning repeater size and

[*] This work was supported in part by NSF grant CCF-0430063 and by Silicon Graphics Inc.

spacing, we can design wires with varying latency and energy properties. To take advantage of VLSI techniques and better match the interconnect design to communication requirements, a heterogeneous interconnect can be employed, where every link consists of wires that are optimized for either latency, energy, or bandwidth. In this study, we explore optimizations that are enabled when such a heterogeneous interconnect is employed for coherence traffic. For example, when employing a directory-based protocol, on a cache write miss, the requesting processor may have to wait for data from the home node (a two hop transaction) and for acknowledgments from other sharers of the block (a three hop transaction). Since the acknowledgments are on the critical path and have low bandwidth needs, they can be mapped to wires optimized for delay, while the data block transfer is not on the critical path and can be mapped to wires that are optimized for low power.

The paper is organized as follows. We discuss related work in Section 2. Section 3 reviews techniques that enable different wire implementations and the design of a heterogeneous interconnect. Section 4 describes the proposed innovations that map coherence messages to different on-chip wires. Section 5 quantitatively evaluates these ideas and we conclude in Section 6.

2. Related Work

To the best of our knowledge, only three other bodies of work have attempted to exploit different types of interconnects at the microarchitecture level. Beckmann and Wood [8, 9] propose speeding up access to large L2 caches by introducing transmission lines between the cache controller and individual banks. Nelson *et al.* [39] propose using optical interconnects to reduce inter-cluster latencies in a clustered architecture where clusters are widely-spaced in an effort to alleviate power density.

A recent paper by Balasubramonian *et al.* [5] introduces the concept of a heterogeneous interconnect and applies it for register communication within a clustered architecture. A subset of load/store addresses are sent on low-latency wires to prefetch data out of the L1D cache, while non-critical register values are transmitted on low-power wires. A heterogeneous interconnect similar to the one in [5] has been applied to a different problem domain in this paper. The nature of cache coherence traffic and the optimizations they enable are very different from that of register traffic within a clustered microarchitecture. We have also improved upon the wire modeling methodology in [5] by modeling the latency and power for all the network components including routers and latches. Our power modeling also takes into account the additional overhead incurred due to the heterogeneous network, such as additional buffers within routers.

3. Wire Implementations

We begin with a quick review of factors that influence wire properties. It is well-known that the delay of a wire is a function of its RC time constant (R is resistance and C is capacitance). Resistance per unit length is (approximately) inversely proportional to the width of the wire [19]. Likewise, a fraction of the capacitance per unit length is inversely proportional to the spacing between wires, and a fraction is directly proportional to wire width. These wire properties provide an opportunity to design wires that trade off bandwidth and latency. By allocating more metal area per wire and increasing wire width and spacing, the net effect is a reduction in the RC time constant. This leads to a wire design that has favorable latency properties, but poor bandwidth properties (as fewer wires can be accommodated in a fixed metal area). In certain cases, nearly a three-fold reduction in wire latency can be achieved, at the expense of a four-fold reduction in bandwidth. Further, researchers are actively pursuing transmission line implementations that enable extremely low communication latencies [12, 16]. However, transmission lines also entail significant metal area overheads in addition to logic overheads for sending and receiving [8, 12]. If transmission line implementations become cost-effective at future technologies, they represent another attractive wire design point that can trade off bandwidth for low latency.

Similar trade-offs can be made between latency and power consumed by wires. Global wires are usually composed of multiple smaller segments that are connected with repeaters [4]. The size and spacing of repeaters influences wire delay and power consumed by the wire. When smaller and fewer repeaters are employed, wire delay increases, but power consumption is reduced. The repeater configuration that minimizes delay is typically very different from the repeater configuration that minimizes power consumption. Banerjee *et al.* [6] show that at 50nm technology, a five-fold reduction in power can be achieved at the expense of a two-fold increase in latency.

Thus, by varying properties such as wire width/spacing and repeater size/spacing, we can implement wires with different latency, bandwidth, and power properties. Consider a CMOS process where global inter-core wires are routed on the 8X and 4X metal planes. Note that the primary differences between minimum-width wires in the 8X and 4X planes are their width, height, and spacing. We will refer to these mimimum-width wires as baseline *B-Wires* (either 8X-B-Wires or 4X-B-Wires). In addition to these wires, we will design two more wire types that may be potentially beneficial (summarized in Figure 1). A low-latency *L-Wire* can be designed by increasing the width and spacing of the wire on the 8X plane (by a factor of four). A power-efficient *PW-Wire* is designed by decreasing the number and size of repeaters within minimum-width wires on the

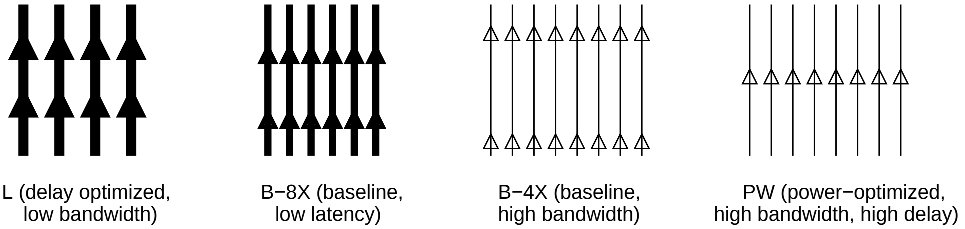


Figure 1. Examples of different wire implementations. Power optimized wires have fewer and smaller repeaters, while bandwidth optimized wires have narrow widths and spacing.

4X plane. While a traditional architecture would employ the entire available metal area for B-Wires (either 4X or 8X), we propose the design of a heterogeneous interconnect, where part of the available metal area is employed for B-Wires, part for L-Wires, and part for PW-Wires. Thus, any data transfer has the option of using one of three sets of wires to effect the communication. A typical composition of a heterogeneous interconnect may be as follows: 256 B-Wires, 512 PW-Wires, 24 L-Wires. In the next section, we will demonstrate how these options can be exploited to improve performance and reduce power consumption. We will also examine the complexity introduced by a heterogeneous interconnect.

4. Optimizing Coherence Traffic

For each cache coherence protocol, there exist a variety of coherence operations with different bandwidth and latency needs. Because of this diversity, there are many opportunities to improve performance and power characteristics by employing a heterogeneous interconnect. The goal of this section is to present a comprehensive listing of such opportunities. We focus on protocol-specific optimizations in Section 4.1 and on protocol-independent techniques in Section 4.2. We discuss the implementation complexity of these techniques in Section 4.3.

4.1 Protocol-Dependent Techniques

We first examine the characteristics of operations in both directory-based and snooping bus-based coherence protocols and how they can map to different sets of wires. In a bus-based protocol, the ability of a cache to directly respond to another cache's request leads to low L1 cache-to-cache miss latencies. L2 cache latencies are relatively high as a processor core has to acquire the bus before sending a request to L2. It is difficult to support a large number of processor cores with a single bus due to the bandwidth and electrical limits of a centralized bus [11]. In a directory-based design [13, 28], each L1 connects to the L2 cache through a point-to-point link. This design has low L2 hit latency and scales better. However, each L1 cache-to-cache miss must be forwarded by the L2 cache, which implies high L1 cache-to-cache latencies. The performance comparison between these two design choices depends on the cache sizes, miss rates, number of outstanding memory requests, working-set sizes, sharing behavior of targeted benchmarks, etc. Since either option may be attractive to chip manufacturers, we will consider both forms of coherence protocols in our study.

Write-Invalidate Directory-Based Protocol

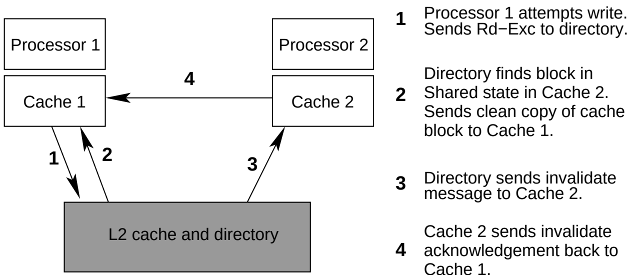
Write-invalidate directory-based protocols have been implemented in existing dual-core CMPs [41] and will likely be used in larger scale CMPs as well. In a directory-based protocol, every cache line has a directory where the states of the block in all L1s are stored. Whenever a request misses in an L1 cache, a coherence message is sent to the directory at the L2 to check the cache line's global state. If there is a clean copy in the L2 and the request is a READ, it is served by the L2 cache. Otherwise, another L1 must hold an exclusive copy and the READ request is forwarded to the exclusive owner, which supplies the data. For a WRITE request, if any other L1 caches hold a copy of the cache line, coherence messages are sent to each of them requesting that they invalidate their copies. The requesting L1 cache acquires the block in exclusive state only after all invalidation messages have been acknowledged.

Hop imbalance is quite common in a directory-based protocol. To exploit this imbalance, we can send critical messages on fast wires to increase performance and send non-critical messages on slow wires to save power. For the sake of this discussion, we assume that the hop latencies of different wires are in the following ratio: L-wire : B-wire : PW-wire :: 1 : 2 : 3

Proposal I: *Read exclusive request for block in shared state*
In this case, the L2 cache's copy is clean, so it provides the data to the requesting L1 and invalidates all shared copies. When the requesting L1 receives the reply message from the L2, it collects invalidation acknowledgment messages from the other L1s before returning the data to the processor core[1]. Figure 2 depicts all generated messages.

The reply message from the L2 requires only one hop, while the invalidation process requires two hops – an example of hop imbalance. Since there is no benefit to receiving

[1] Some coherence protocols may not impose all of these constraints, thereby deviating from a sequentially consistent memory model.



1 Processor 1 attempts write. Sends Rd−Exc to directory.

2 Directory finds block in Shared state in Cache 2. Sends clean copy of cache block to Cache 1.

3 Directory sends invalidate message to Cache 2.

4 Cache 2 sends invalidate acknowledgement back to Cache 1.

Figure 2. Read exclusive request for a shared block in MESI protocol

the cache line early, latencies for each hop can be chosen so as to equalize communication latency for the cache line and the acknowledgment messages. Acknowledgment messages include identifiers so they can be matched against the outstanding request in the L1's MSHR. Since there are only a few outstanding requests in the system, the identifier requires few bits, allowing the acknowledgment to be transferred on a few low-latency L-Wires. Simultaneously, the data block transmission from the L2 can happen on low-power PW-Wires and still finish before the arrival of the acknowledgments. This strategy improves performance (because acknowledgments are often on the critical path) and reduces power consumption (because the data block is now transferred on power-efficient wires). While circuit designers have frequently employed different types of wires within a circuit to reduce power dissipation without extending the critical path, the proposals in this paper represent some of the first attempts to exploit wire properties at the architectural level.

Proposal II: *Read request for block in exclusive state*
In this case, the value in the L2 is likely to be stale and the following protocol actions are taken. The L2 cache sends a speculative data reply to the requesting L1 and forwards the read request as an intervention message to the exclusive owner. If the cache copy in the exclusive owner is clean, an acknowledgment message is sent to the requesting L1, indicating that the speculative data reply from the L2 is valid. If the cache copy is dirty, a response message with the latest data is sent to the requesting L1 and a write-back message is sent to the L2. Since the requesting L1 cannot proceed until it receives a message from the exclusive owner, the speculative data reply from the L2 (a single hop transfer) can be sent on slower PW-Wires. The forwarded request to the exclusive owner is on the critical path, but includes the block address. It is therefore not eligible for transfer on low-bandwidth L-Wires. If the owner's copy is in the exclusive clean state, a short acknowledgment message to the requestor can be sent on L-Wires. If the owner's copy is dirty, the cache block can be sent over B-Wires, while the low priority writeback to the L2 can happen on PW-Wires. With the above mapping, we accelerate the critical path by using faster L-Wires, while also lowering power consumption by sending non-critical data on PW-Wires. The above protocol actions apply even in the case when a read-exclusive request is made for a block in the exclusive state.

Proposal III: *NACK messages*
When the directory state is busy, incoming requests are often NACKed by the home directory, i.e., a negative acknowledgment is sent to the requester rather than buffering the request. Typically the requesting cache controller reissues the request and the request is serialized in the order in which it is actually accepted by the directory. A NACK message can be matched by comparing the request id (MSHR index) rather than the full address, so a NACK is eligible for transfer on low-bandwidth L-Wires. If load at the home directory is low, it will likely be able to serve the request when it arrives again, in which case sending the NACK on fast L-Wires can improve performance. In contrast, when load is high, frequent backoff-and-retry cycles are experienced. In this case, fast NACKs only increase traffic levels without providing any performance benefit. In this case, in order to save power, NACKs can be sent on PW-Wires.

Proposal IV: *Unblock and write control messages*
Some protocols [36] employ unblock and write control messages to reduce implementation complexity. For every read transaction, a processor first sends a request message that changes the L2 cache state into a transient state. After receiving the data reply, it sends an unblock message to change the L2 cache state back to a stable state. Similarly, write control messages are used to implement a 3-phase writeback transaction. A processor first sends a control message to the directory to order the writeback message with other request messages. After receiving the writeback response from the directory, the processor sends the data. This avoids a race condition in which the processor sends the writeback data while a request is being forwarded to it. Sending unblock messages on L-Wires can improve performance by reducing the time cache lines are in busy states. Write control messages (writeback request and writeback grant) are not on the critical path, although they are also eligible for transfer on L-Wires. The choice of sending writeback control messages on L-Wires or PW-Wires represents a power-performance trade-off.

Write-Invalidate Bus-Based Protocol

We next examine techniques that apply to bus-based snooping protocols.

Proposal V: *Signal wires*
In a bus-based system, three wired-OR signals are typically employed to avoid involving the lower/slower memory hierarchy [15]. Two of these signals are responsible for reporting the state of snoop results and the third indicates that the

snoop result is valid. The first signal is asserted when any L1 cache, besides the requester, has a copy of the block. The second signal is asserted if any cache has the block in exclusive state. The third signal is an inhibit signal, asserted until all caches have completed their snoop operations. When the third signal is asserted, the requesting L1 and the L2 can safely examine the other two signals. Since all of these signals are on the critical path, implementing them using low-latency L-Wires can improve performance.

Proposal VI: *Voting wires*
Another design choice is whether to use cache-to-cache transfers if the data is in the shared state in a cache. The Silicon Graphics Challenge [17] and the Sun Enterprise use cache-to-cache transfers only for data in the modified state, in which case there is a single supplier. On the other hand, in the full Illinois MESI protocol, a block can be preferentially retrieved from another cache rather than from memory. However, when multiple caches share a copy, a "voting" mechanism is required to decide which cache will supply the data, and this voting mechanism can benefit from the use of low latency wires.

4.2. Protocol-independent Techniques

Proposal VII: *Narrow Bit-Width Operands for Synchronization Variables*
Synchronization is one of the most important factors in the performance of a parallel application. Synchronization is not only often on the critical path, but it also contributes a large percentage (up to 40%) of coherence misses [30]. Locks and barriers are the two most widely used synchronization constructs. Both of them use small integers to implement mutual exclusion. Locks often toggle the synchronization variable between zero and one, while barriers often linearly increase a barrier variable from zero to the number of processors taking part in the barrier operation. Such data transfers have limited bandwidth needs and can benefit from using L-Wires.

This optimization can be further extended by examining the general problem of cache line compaction. For example, if a cache line is comprised mostly of 0 bits, trivial data compaction algorithms may reduce the bandwidth needs of the cache line, allowing it to be transferred on L-Wires instead of B-Wires. If the wire latency difference between the two wire implementations is greater than the delay of the compaction/de-compaction algorithm, performance improvements are possible.

Proposal VIII: *Assigning Writeback Data to PW-Wires*
Writeback data transfers result from cache replacements or external request/intervention messages. Since writeback messages are rarely on the critical path, assigning them to PW-Wires can save power without incurring significant performance penalties.

Proposal IX: *Assigning Narrow Messages to L-Wires*
Coherence messages that include the data block address or the data block itself are many bytes wide. However, many other messages, such as acknowledgments and NACKs, do not include the address or data block and only contain control information (source/destination, message type, MSHR id, etc.). Such narrow messages can be always assigned to low latency L-Wires to accelerate the critical path.

4.3. Implementation Complexity

4.3.1 Overhead in Heterogeneous Interconnect Implementation

In a conventional multiprocessor interconnect, a subset of wires are employed for addresses, a subset for data, and a subset for control signals. Every bit of communication is mapped to a unique wire. When employing a heterogeneous interconnect, a communication bit can map to multiple wires. For example, data returned by the L2 in response to a read-exclusive request may map to B-Wires or PW-Wires depending on whether there are other sharers for that block (**Proposal I**). Thus, every wire must be associated with a multiplexor and de-multiplexor.

The entire network operates at the same fixed clock frequency, which means that the number of latches within every link is a function of the link latency. Therefore, PW-Wires have to employ additional latches, relative to the baseline B-Wires. Dynamic power per latch at 5GHz and 65nm technology is calculated to be 0.1mW, while leakage power per latch equals 19.8μW [25]. The power per unit length for each wire is computed in the next section. Power overheads due to these latches for different wires are tabulated in Table 1. Latches impose a 2% overhead within B-Wires, but a 13% overhead within PW-Wires.

The proposed model also introduces additional complexity in the routing logic. The base case router employs a cross-bar switch and 8-entry message buffers at each input port. Whenever a message arrives, it is stored in the input buffer and routed to an allocator that locates the output port and transfers the message. In case of a heterogeneous model, three different buffers are required at each port to store L, B, and PW messages separately. In our simulations we employ three 4-entry message buffers for each port. The size of each buffer is proportional to the flit size of the corresponding set of wires. For example, a set of 24 L-Wires employs a 4-entry message buffer with a word size of 24 bits. For power calculations we have also included the fixed additional overhead associated with these small buffers as opposed to a single larger buffer employed in the base case. In our proposed processor model, the dynamic characterization of messages happens only in the processors and intermediate network routers cannot re-assign a message to a different set of wires. While this may have a negative effect on performance in a highly utilized network, we chose to keep the routers simple and not implement such a feature.

Wire Type	Power/Length mW/mm	Latch Power mW/latch	Latch Spacing mm	Total Power/10mm mW/10mm
B-Wire – 8X plane	1.4221	0.119	5.15	14.46
B-Wire – 4X plane	1.5928	0.119	3.4	16.29
L-Wire – 8X plane	0.7860	0.119	9.8	7.80
PW-wire – 4X plane	0.4778	0.119	1.7	5.48

Table 1. Power characteristics of different wire implementations. For calculating the power/length, activity factor α (described in Table 3) is assumed to be 0.15. The above latch spacing values are for a 5GHz network.

For a network employing virtual channel flow control, each set of wires in the heterogeneous network link is treated as a separate physical channel and the same number of virtual channels are maintained per physical channel. Therefore, the heterogeneous network has a larger total number of virtual channels and the routers require more state fields to keep track of these additional virtual channels. To summarize, the additional overhead introduced by the heterogeneous model comes in the form of potentially more latches and greater routing complexity.

4.3.2 Overhead in Decision Process

The decision process in selecting the right set of wires is minimal. For example, in **Proposal I**, an OR function on the directory state for that block is enough to select either B- or PW-Wires. In **Proposal II**, the decision process involves a check to determine if the block is in the exclusive state. To support **Proposal III**, we need a mechanism that tracks the level of congestion in the network (for example, the number of buffered outstanding messages). There is no decision process involved for **Proposals IV**, **V**, **VI** and **VIII**. **Proposals VII** and **IX** require logic to compute the width of an operand, similar to logic used in the PowerPC 603 [18] to determine the latency for integer multiply.

4.3.3 Overhead in Cache Coherence Protocols

Most coherence protocols are already designed to be robust in the face of variable delays for different messages. For protocols relying on message order within a virtual channel, each virtual channel can be made to consist of a set of L-, B-, and PW-message buffers. A multiplexor can be used to activate only one type of message buffer at a time to ensure correctness. For other protocols that are designed to handle message re-ordering within a virtual channel, we propose to employ one dedicated virtual channel for each set of wires to fully exploit the benefits of a heterogeneous interconnect. In all proposed innovations, a data packet is not distributed across different sets of wires. Therefore, different components of an entity do not arrive at different periods of time, thereby eliminating any timing problems. It may be worth considering sending the critical word of a cache line on L-Wires and the rest of the cache line on PW-Wires. Such a proposal may entail non-trivial complexity to handle corner cases and is not discussed further in this paper.

In a snooping bus-based coherence protocol, transactions are serialized by the order in which addresses appear on the bus. None of our proposed innovations for snooping protocols affect the transmission of address bits (address bits are always transmitted on B-Wires), so the transaction serialization model is preserved.

5. Results

5.1. Methodology

5.1.1 Simulator

We simulate a 16-core CMP with the Virtutech Simics full-system functional execution-driven simulator [33] and a timing infrastructure GEMS [34]. GEMS can simulate both in-order and out-of-order processors. In most studies, we use the in-order blocking processor model provided by Simics to drive the detailed memory model (Ruby) for fast simulation. Ruby implements a one-level MOESI directory cache coherence protocol with migratory sharing optimization [14, 40]. All processor cores share a non-inclusive L2 cache, which is organized as a non-uniform cache architecture (NUCA) [22]. Ruby can also be driven by an out-of-order processor module called Opal, and we report the impact of the processor cores on the heterogeneous interconnect in Section 5.3. Opal is a timing-first simulator that implements the performance sensitive aspects of an out of order processor but ultimately relies on Simics to provide functional correctness. We configure Opal to model the processor described in Table 2 and use an aggressive implementation of sequential consistency.

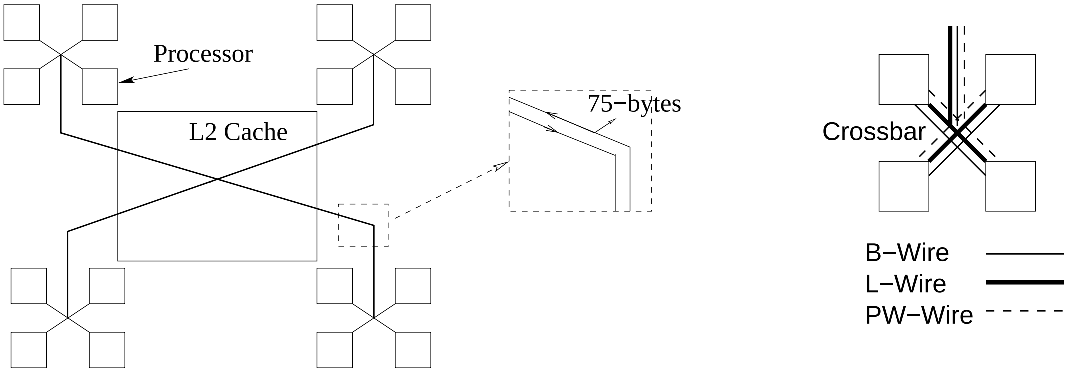
To test our ideas, we employ a workload consisting of all programs from the SPLASH-2 [43] benchmark suite. The programs were run to completion, but all experimental results reported in this paper are for the parallel phases of these applications. We use default input sets for most programs except fft and radix. Since the default working sets of these two programs are too small, we increase the working set of fft to 1M data points and that of radix to 4M keys.

5.1.2 Interconnect Power/Delay/Area Models

This section describes details of the interconnect architecture and the methodology we employ for calculating the area, delay, and power values of the interconnect. We consider 65nm process technology and assume 10 metal layers, 4 layers in 1X plane and 2 layers, in each 2X, 4X, and 8X plane [25]. For most of our study we employ a crossbar based hierarchical interconnect structure to connect the cores and L2 cache (Figure 3(a)), similar to that in SGI's

Parameter	Value	Parameter	Value
number of cores	16	clock frequency	5GHz
pipeline width	4-wide fetch and issue	pipeline stages	11
cache block size	64 Bytes	split L1 I & D cache	128KB, 4-way
shared L2 cache	8MBytes, 4-way, 16-banks non-inclusive NUCA	memory/dir controllers	30 cycles
interconnect link latency	4 cycles (one-way) for the baseline 8X-B-Wires	DRAM latency	400 cycles
memory bank capacity	1 GByte per bank	latency to mem controller	100 cycles

Table 2. System configuration.



b) Links with different sets of wires

a) Hierarchical network topology for 16−core CMP

Figure 3. Interconnect model used for coherence transactions in a sixteen-core CMP.

NUMALink-4 [1]. The effect of other interconnect topologies is discussed in our sensitivity analysis. In the base case, each link in Figure 3(a) consists of (in each direction) 64-bit address wires, 64-byte data wires, and 24-bit control wires. The control signals carry source, destination, signal type, and Miss Status Holding Register (MSHR) id. All wires are fully pipelined. Thus, each link in the interconnect is capable of transferring 75 bytes in each direction. Error Correction Codes (ECC) account for another 13% overhead in addition to the above mentioned wires [38]. All the wires of the base case are routed as B-Wires in the 8X plane.

As shown in Figure 3(b), the proposed heterogeneous model employs additional wire types within each link. In addition to B-Wires, each link includes low-latency, low-bandwidth L-Wires and high-bandwidth, high-latency, power-efficient, PW-Wires. The number of L- and PW-Wires that can be employed is a function of the available metal area and the needs of the coherence protocol. In order to match the metal area with the baseline, each unidirectional link within the heterogeneous model is designed to be made up of 24 L-Wires, 512 PW-Wires, and 256 B-Wires (the base case has 600 B-Wires, not counting ECC). In a cycle, three messages may be sent, one on each of the three sets of wires. The bandwidth, delay, and power calculations for these wires are discussed subsequently.

Table 3 summarizes the different types of wires and their area, delay, and power characteristics. The area overhead of the interconnect can be mainly attributed to repeaters and wires. We use wire width and spacing (based on ITRS projections) to calculate the effective area for minimum-width wires in the 4X and 8X plane. L-Wires are designed to occupy four times the area of minimum-width 8X-B-Wires.

Delay

Our wire model is based on the RC models proposed in [6, 19, 37]. The delay per unit length of a wire with optimally placed repeaters is given by equation (1), where R_{wire} is resistance per unit length of the wire, C_{wire} is capacitance per unit length of the wire, and $FO1$ is the fan-out of one delay:

$$Latency_{wire} = 2.13\sqrt{R_{wire}C_{wire}FO1} \qquad (1)$$

R_{wire} is inversely proportional to wire width, while C_{wire} depends on the following three components: (i) fringing capacitance that accounts for the capacitance between the side wall of the wire and substrate, (ii) parallel plate capacitance between the top and bottom layers of the metal that is directly proportional to the width of the metal, (iii) parallel plate capacitance between the adjacent metal wires that is inversely proportional to the spacing between the wires. The C_{wire} value for the top most metal layer at 65nm technology is given by equation (2) [37].

$$C_{wire} = 0.065 + 0.057W + 0.015/S(fF/\mu) \qquad (2)$$

We derive relative delays for different types of wires by tuning width and spacing in the above equations. A variety of width and spacing values can allow L-Wires to yield a two-fold latency improvement at a four-fold area cost, relative to 8X-B-Wires. In order to reduce power consumption, we

Wire Type	Relative Latency	Relative Area $(wireWidth + spacing)$	Dynamic Power (W/m) α = Switching Factor	Static Power W/m
B-Wire (8X plane)	$1x$	$1x$	2.65α	1.0246
B-Wire (4X plane)	$1.6x$	$0.5x$	2.9α	1.1578
L-Wire (8X plane)	$0.5x$	$4x$	1.46α	0.5670
PW-Wire (4X plane)	$3.2x$	$0.5x$	0.87α	0.3074

Table 3. Area, delay, and power characteristics of different wire implementations.

Component	Energy/transaction (J)
Arbiter	6.43079e-14
Crossbar	5.32285e-12
Buffer read operation	1.23757e-12
Buffer write operation	1.73723e-12

Table 4. Energy consumed (max) by arbiters, buffers, and crossbars for a 32-byte transfer.

selected a wire implementation where the L-Wire's width was twice that of the minimum width and the spacing was six times as much as the minimum spacing for the 8X metal plane.

Power

The total power consumed by a wire is the sum of three components (dynamic, leakage, and short-circuit power). Equations derived by Banerjee and Mehrotra [6] are used to derive the power consumed by L- and B-Wires. These equations take into account optimal repeater size/spacing and wire width/spacing. PW-Wires are designed to have twice the delay of 4X-B-Wires. At 65nm technology, for a delay penalty of 100%, smaller and widely-spaced repeaters enable power reduction by 70% [6].

Routers

Crossbars, buffers, and arbiters are the major contributors for router power [42]:

$$E_{router} = E_{buffer} + E_{crossbar} + E_{arbiter} \qquad (3)$$

The capacitance and energy for each of these components is based on analytical models proposed by Wang *et al.* [42]. We model a 5x5 matrix crossbar that employs a tristate buffer connector. As described in Section 4.3, buffers are modeled for each set of wires with word size corresponding to flit size. Table 3 shows the peak energy consumed by each component of the router for a single 32-byte transaction.

5.2 Results

For our simulations, we restrict ourselves to directory-based protocols. We model the effect of proposals pertaining to such a protocol: **I, III, IV, VIII, IX**. Proposal-II optimizes speculative reply messages in MESI protocols, which are not implemented within GEMS' MOESI protocol. Evaluations involving compaction of cache blocks (Proposal VII) is left as future work.

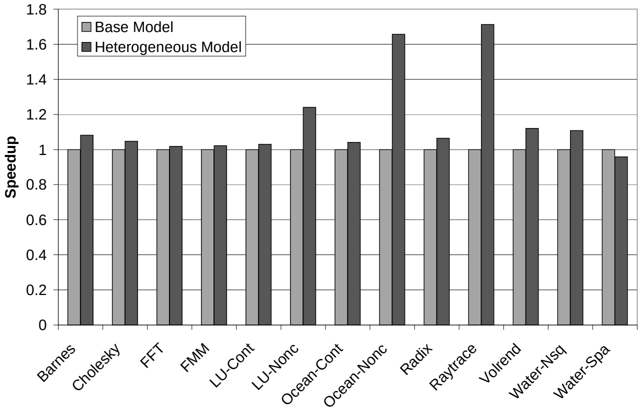


Figure 4. Speedup of heterogeneous interconnect

Figure 4 shows the execution time in cycles for SPLASH2 programs. The first bar shows the performance of the baseline organization that has one interconnect layer of 75 bytes, composed entirely of 8X-B-Wires. The second shows the performance of the heterogeneous interconnect model in which each link consists of 24-bit L-wires, 32-byte B-wires, and 64-byte PW-wires. Programs such as LU-Non-continuous, Ocean-Non-continuous, and Raytracing yield significant improvements in performance. These performance numbers can be analyzed with the help of Figure 5 that shows the distribution of different transfers that happen on the interconnect. Transfers on L-Wires can have a huge impact on performance, provided they are on the program critical path. LU-Non-continuous, Ocean-Non-continuous, Ocean-Continuous, and Raytracing experience the most transfers on L-Wires. But the performance improvement of Ocean-Continuous is very low compared to other benchmarks. This can be attributed to the fact that Ocean-Continuous incurs the most L2 cache misses and is mostly memory bound. The transfers on PW-Wires have a negligible effect on performance for all benchmarks. This is because PW-Wires are employed only for writeback transfers that are always off the critical path. On average, we observe a 11.2% improvement in performance, compared to the baseline, by employing heterogeneity within the network.

Proposals I, III, IV, and IX exploit L-Wires to send small messages within the protocol, and contribute 2.3, 0, 60.3,

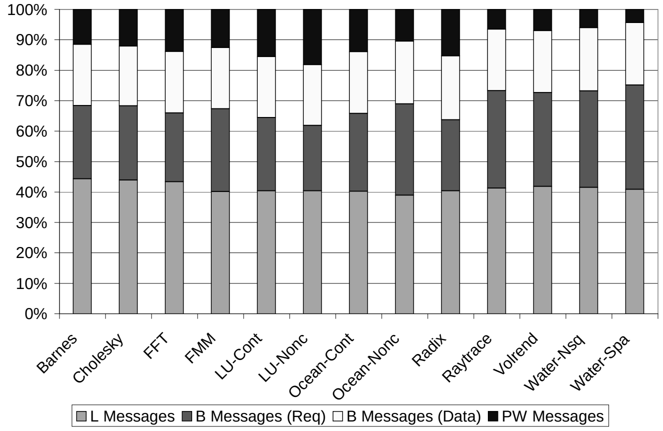


Figure 5. Distribution of messages on the heterogeneous network. B-Wire transfers are classified as Request and Data.

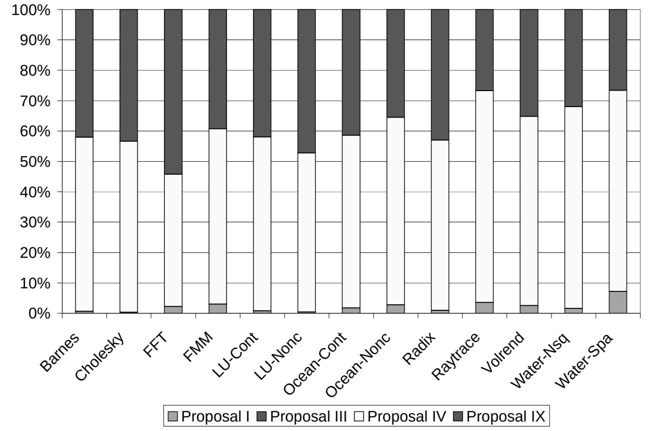


Figure 6. Distribution of L-message transfers across different proposals.

and 37.4 per cent, respectively, to total L-Wire traffic. A per-benchmark breakdown is shown in Figure 6. Proposal-I optimizes the case of a read exclusive request for a block in shared state, which is not very common in the SPLASH2 benchmarks. We expect the impact of Proposal-I to be much higher in commercial workloads where cache-to-cache misses dominate. Proposal-III and Proposal-IV impact NACK, unblocking, and writecontrol messages. These messages are widely used to reduce the implementation complexity of coherence protocols. In GEMS' MOESI protocol, NACK messages are only used to handle the race condition between two write-back messages, which are negligible in our study (causing the zero contribution of Proposal-III). Instead, the protocol implementation heavily relies on unblocking and writecontrol messages to maintain the order between read and write transactions, as discussed in Section 4.1. The frequency of occurrence of NACK, unblocking, and writecontrol messages depends on the protocol implementation, but we expect the sum of these messages to be relatively constant in different protocols and play an important role in L-wire optimizations. Proposal-IX includes all other acknowledgment messages eligible for transfer on L-Wires.

We observed that the combination of proposals I, III, IV, and IX caused a performance improvement more than the sum of improvements from each individual proposal. A parallel benchmark can be divided into a number of phases by synchronization variables (barriers), and the execution time of each phase can be defined as the longest time any thread spends from one barrier to the next. Optimizations applied to a single thread may have no effect if there are other threads on the critical path. However, a different optimization may apply to the threads on the critical path, reduce their execution time, and expose the performance of other threads and the optimizations that apply to them. Since different threads take different data paths, most parallel applications show nontrivial workload imbalance [31]. Therefore, employing one proposal might not speedup all threads on the critical path, but employing all applicable proposals can probably optimize threads on every path, thereby reducing the total barrier to barrier time.

Figure 7 shows the improvement in network energy due to the heterogeneous interconnect model. The first bar shows the reduction in network energy and the second bar shows the improvement in the overall processor $Energy \times Delay^2$ (ED^2) metric. Other metrics in the $E - D$ space can also be computed with data in Figures 7 and 4. To calculate ED^2, we assume that the total power consumption of the chip is 200W, of which the network power accounts for 60W. The energy improvement in the heterogeneous case comes from both L and PW transfers. Many control messages that are sent on B-Wires in the base case are sent on L-Wires in the heterogeneous case. As per Table 3, the energy consumed by an L-Wire is less than the energy consumed by a B-Wire. But due to the small sizes of these messages, the contribution of L-messages to the total energy savings is negligible. Overall, the heterogeneous network results in a 22% saving in network energy and a 30% improvement in ED^2.

5.3 Sensitivity Analysis

In this sub-section, we discuss the impact of processor cores, link bandwidth, routing algorithm, and network topology on the heterogeneous interconnect.

Out-of-order/In-order Processors

To test our ideas with an out-of-order processor, we configure Opal to model the processor described in Table 2 and only report the results of the first 100M instructions in the

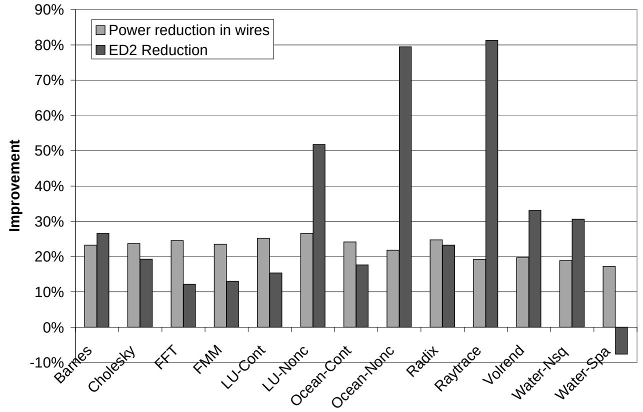


Figure 7. Improvement in link energy and ED^2.

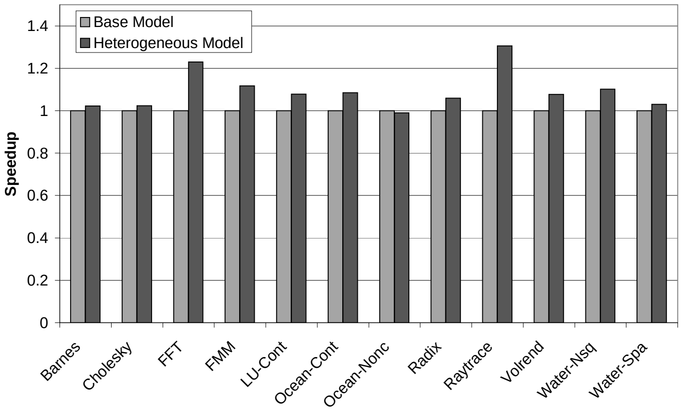


Figure 8. Speedup of heterogeneous interconnect when driven by OoO cores (Opal and Ruby)

parallel sections[2].

Figure 8 shows the performance speedup of the heterogeneous interconnect over the baseline. All benchmarks except Ocean-Noncontinuous demonstrate different degrees of performance improvement, which leads to an average speedup of 9.3%. The average performance improvement is less than what we observe in a system employing in-order cores (11.2%). This can be attributed to the greater tolerance that an out-of-order processor has to long instruction latencies.

Link Bandwidth

The heterogeneous network poses more constraints on the type of messages that can be issued by a processor in a cycle. It is therefore likely to not perform very well in a bandwidth-constrained system. To verify this, we modeled a base case where every link has only 80 8X-B-Wires and a heterogeneous case where every link is composed of 24 L-Wires, 24 8X-B-Wires, and 48 PW-Wires (almost twice the metal area of the new base case). Benchmarks with higher network utilizations suffered significant performance losses. In our experiments raytracing has the maximum messages/cycle ratio and the heterogeneous case suffered a 27% performance loss, compared to the base case (in spite of having twice the metal area). The heterogeneous interconnect performance improvement for Ocean Non-continuous and LU Non-continuous is 12% and 11%, as against 39% and 20% in the high-bandwidth simulations. Overall, the heterogeneous model performed 1.5% worse than the base case.

Routing Algorithm

Our simulations thus far have employed adaptive routing within the network. Adaptive routing alleviates the contention problem by dynamically routing messages based on the network traffic. We found that deterministic routing degraded performance by about 3% for most programs for systems with the baseline and with the heterogeneous network. Raytracing is the only benchmark that incurs a significant performance penalty of 27% for both networks.

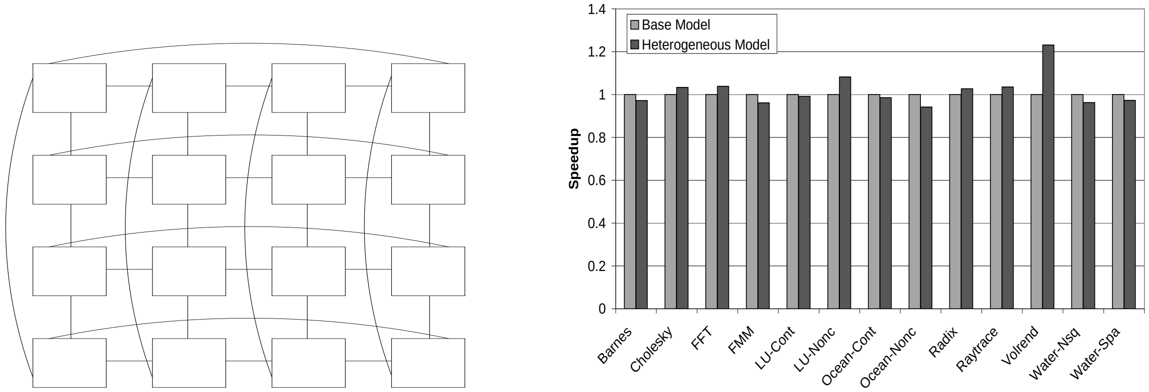
Network Topology

Our default interconnect thus far was a two-level tree based on SGI's NUMALink-4 [1]. To test the sensitivity of our results to the network topology, we also examine a 2D-torus interconnect resembling that in the Alpha 21364 [7]. As shown in Figure 9, each router connects to 4 links that connect to 4 neighbors in the torus, and wraparound links are employed to connect routers on the boundary.

Our proposed mechanisms show much less performance benefit (1.3% on average) in the 2D torus interconnect than in the two-level tree interconnect. The main reason is that our decision process in selecting the right set of wires calculates hop imbalance at the coherence protocol level without considering the physical hops a message takes on the mapped topology. For example, in a 3-hop transaction as shown in Figure 2, the one-hop message may take 4 physical hops while the 2-hop message may also take 4 physical hops. In this case, sending the 2-hop message on the L-Wires and the one-hop message on the PW-Wires will actually lower performance.

This is not a first-order effect in the two-level tree interconnect, where most hops take 4 physical hops. However, the average distance between two processors in the 2D torus interconnect is 2.13 physical hops with a standard deviation of 0.92 hops. In an interconnect with such high standard deviation, calculating hop imbalance based on protocol hops is inaccurate. For future work, we plan to develop a more accurate decision process that considers source id, destination id, and interconnect topology to dynamically compute

[2] Simulating the entire program takes nearly a week and there exist no effective toolkits to find the representative phases for parallel benchmarks. LU-Noncontinuous and Radix were not compatible with the Opal timing module.



(a) 2D Torus topology.

(b) Heterogeneous interconnect speedup.

Figure 9. **Results for the 2D Torus.**

an optimal mapping to wires.

6. Conclusions and Future Work

Coherence traffic in a chip multiprocessor has diverse needs. Some messages can tolerate long latencies, while others are on the program critical path. Further, messages have varied bandwidth demands. On-chip global wires can be designed to optimize latency, bandwidth, or power. We advocate partitioning available metal area across different wire implementations and intelligently mapping data to the set of wires best suited for its communication. This paper presents numerous novel techniques that can exploit a heterogeneous interconnect to simultaneously improve performance and reduce power consumption.

Our evaluation of a subset of the proposed techniques shows that a large fraction of messages have low bandwidth needs and can be transmitted on low latency wires, thereby yielding a performance improvement of 11.2%. At the same time, a 22.5% reduction in interconnect energy is observed by transmitting non-critical data on power-efficient wires. The complexity cost is marginal as the mapping of messages to wires entails simple logic.

For future work, we plan to strengthen our decision process in calculating the hop imbalance based on the topology of the interconnect. We will also evaluate the potential of other techniques listed in this paper. There may be several other applications of heterogeneous interconnects within a CMP. For example, in the *Dynamic Self Invalidation* scheme proposed by Lebeck *et al.* [29], the self-invalidate [27, 29] messages can be effected through power-efficient PW-Wires. In a processor model implementing token coherence, the low-bandwidth token messages [35] are often on the critical path and thus, can be effected on L-Wires. A recent study by Huh *et al.* [21] reduces the frequency of false sharing by employing incoherent data. For cache lines suffering from false sharing, only the sharing states need to be propagated and such messages are a good match for low-bandwidth L-Wires.

References

[1] SGI Altix 3000 Configuration. “http://www.sgi.com/products/servers/altix/configs.html”.

[2] M. E. Acacio, J. Gonzalez, J. M. Garcia, and J. Duato. The Use of Prediction for Accelerating Upgrade Misses in CC-NUMA Multiprocessors. In *Proceedings of PACT-11*, 2002.

[3] V. Agarwal, M. Hrishikesh, S. Keckler, and D. Burger. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. In *Proceedings of ISCA-27*, pages 248–259, June 2000.

[4] H. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley, 1990.

[5] R. Balasubramonian, N. Muralimanohar, K. Ramani, and V. Venkatachalapathy. Microarchitectural Wire Management for Performance and Power in Partitioned Architectures. In *Proceedings of HPCA-11*, February 2005.

[6] K. Banerjee and A. Mehrotra. A Power-optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs. *IEEE Transactions on Electron Devices*, 49(11):2001–2007, November 2002.

[7] P. Bannon. Alpha 21364: A Scalable Single-Chip SMP. October 1998.

[8] B. Beckmann and D. Wood. TLC: Transmission Line Caches. In *Proceedings of MICRO-36*, December 2003.

[9] B. Beckmann and D. Wood. Managing Wire Delay in Large Chip-Multiprocessor Caches. In *Proceedings of MICRO-37*, December 2004.

[10] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood. Multicast Snooping: A New Coherence Method using a Multicast Address Network. *SIGARCH Comput. Archit. News*, pages 294–304, 1999.

[11] F. A. Briggs, M. Cekleov, K. Creta, M. Khare, S. Kulick, A. Kumar, L. P. Looi, C. Natarajan, S. Radhakrishnan, and L. Rankin. Intel 870: A Building Block for Cost-Effective, Scalable Servers. *IEEE Micro*, 22(2):36–47, 2002.

[12] R. Chang, N. Talwalkar, C. Yue, and S. Wong. Near Speed-of-Light Signaling Over On-Chip Electrical Interconnects. *IEEE Journal of Solid-State Circuits*, 38(5):834–838, May 2003.

[13] Corporate Institute of Electrical and Electronics Engineers, Inc. Staff. *IEEE Standard for Scalable Coherent Interface, Science: IEEE Std. 1596-1992*. 1993.

[14] A. Cox and R. Fowler. Adaptive Cache Coherency for Detecting Migratory Shared Data. pages 98–108, May 1993.

[15] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: a Hardware/software Approach*. Morgan Kaufmann Publishers, Inc, 1999.

[16] W. Dally and J. Poulton. *Digital System Engineering*. Cambridge University Press, Cambridge, UK, 1998.

[17] M. Galles and E. Williams. Performance Optimizations, Implementation, and Verification of the SGI Challenge Multiprocessor. In *HICSS (1)*, pages 134–143, 1994.

[18] G. Gerosa and et al. A 2.2 W, 80 MHz Superscalar RISC Microprocessor. *IEEE Journal of Solid-State Circuits*, 29(12):1440–1454, December 1994.

[19] R. Ho, K. Mai, and M. Horowitz. The Future of Wires. *Proceedings of the IEEE*, Vol.89, No.4, April 2001.

[20] P. Hofstee. Power Efficient Processor Architecture and The Cell Processor. In *Proceedings of HPCA-11 (Industrial Session)*, February 2005.

[21] J. Huh, J. Chang, D. Burger, and G. S. Sohi. Coherence Decoupling: Making Use of Incoherence. In *Proceedings of ASPLOS-XI*, pages 97–106, 2004.

[22] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA Substrate for Flexible CMP Cache Sharing. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 31–40, New York, NY, USA, 2005. ACM Press.

[23] P. Kongetira. A 32-Way Multithreaded SPARC Processor. In *Proceedings of Hot Chips 16*, 2004. (`http://www.hotchips.org/archives/`).

[24] K. Krewell. UltraSPARC IV Mirrors Predecessor: Sun Builds Dualcore Chip in 130nm. *Microprocessor Report*, pages 1,5–6, Nov. 2003.

[25] R. Kumar, V. Zyuban, and D. Tullsen. Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads, and Scaling. In *Proceedings of the 32nd ISCA*, June 2005.

[26] A.-C. Lai and B. Falsafi. Memory Sharing Predictor: The Key to a Speculative Coherent DSM. In *Proceedings of ISCA-26*, 1999.

[27] A.-C. Lai and B. Falsafi. Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction. In *Proceedings of ISCA-27*, pages 139–148, 2000.

[28] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of ISCA-24*, pages 241–251, June 1997.

[29] A. R. Lebeck and D. A. Wood. Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors. In *Proceedings of ISCA-22*, pages 48–59, 1995.

[30] K. M. Lepak and M. H. Lipasti. Temporally Silent Stores. In *Proceedings of ASPLOS-X*, pages 30–41, 2002.

[31] J. Li, J. F. Martinez, and M. C. Huang. The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors. In *HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture*, page 14, Washington, DC, USA, 2004. IEEE Computer Society.

[32] N. Magen, A. Kolodny, U. Weiser, and N. Shamir. Interconnect Power Dissipation in a Microprocessor. In *Proceedings of System Level Interconnect Prediction*, February 2004.

[33] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, 35(2):50–58, February 2002.

[34] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood. Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset. *Computer Architecture News*, 2005.

[35] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token Coherence: Decoupling Performance and Correctness. In *Proceedings of ISCA-30*, 2003.

[36] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood. Improving Multiple-CMP Systems Using Token Coherence. In *HPCA*, pages 328–339, 2005.

[37] M. L. Mui, K. Banerjee, and A. Mehrotra. A Global Interconnect Optimization Scheme for Nanometer Scale VLSI With Implications for Latency, Bandwidth, and Power Dissipation. *IEEE Transactions on Electronic Devices*, Vol.51, No.2, February 2004.

[38] S. Mukherjee, J. Emer, and S. Reinhardt. The Soft Error Problem: An Architectural Perspective. In *Proceedings of HPCA-11 (Industrial Session)*, February 2005.

[39] N. Nelson, G. Briggs, M. Haurylau, G. Chen, H. Chen, D. Albonesi, E. Friedman, and P. Fauchet. Alleviating Thermal Constraints while Maintaining Performance Via Silicon-Based On-Chip Optical Interconnects. In *Proceedings of Workshop on Unique Chips and Systems*, March 2005.

[40] P. Stenström, M. Brorsson, and L. Sandberg. An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing. pages 109–118, May 1993.

[41] J. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. POWER4 System Microarchitecture. Technical report, IBM Server Group Whitepaper, October 2001.

[42] H. S. Wang, L. S. Peh, and S. Malik. A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers. In *IEEE Micro, Vol 24, No 1*, January 2003.

[43] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of ISCA-22*, pages 24–36, June 1995.