Microarchitectural Wire Management for Performance and Power in Partitioned Architectures *

Rajeev Balasubramonian, Naveen Muralimanohar, Karthik Ramani, Venkatanand Venkatachalapathy
University of Utah

Abstract

Future high-performance billion-transistor processors are likely to employ partitioned architectures to achieve high clock speeds, high parallelism, low design complexity, and low power. In such architectures, inter-partition communication over global wires has a significant impact on overall processor performance and power consumption. VLSI techniques allow a variety of wire implementations, but these wire properties have previously never been exposed to the microarchitecture. This paper advocates global wire management at the microarchitecture level and proposes a heterogeneous interconnect that is comprised of wires with varying latency, bandwidth, and energy characteristics. We propose and evaluate microarchitectural techniques that can exploit such a heterogeneous interconnect to improve performance and reduce energy consumption. These techniques include a novel cache pipeline design, the identification of narrow bit-width operands, the classification of non-critical data, and the detection of interconnect load imbalance. For a dynamically scheduled partitioned architecture, our results demonstrate that the proposed innovations result in up to 11% reductions in overall processor ED^2, compared to a baseline processor that employs a homogeneous interconnect.

1. Introduction

One of the biggest challenges for computer architects is the design of billion-transistor architectures that yield high parallelism, high clock speeds, low design complexity, and low power. There appears to be a consensus among several research groups [1, 7, 10, 15, 18, 21, 25, 27, 28, 29, 30, 35, 36, 39, 42] that a *partitioned architecture* is the best approach to achieving these design goals.

Partitioned architectures consist of many small and fast computational units connected by a communication fabric. A computational unit is commonly referred to as a *cluster* and is typically comprised of a limited number of ALUs, local register storage, and a buffer for instruction issue. Since a cluster has limited resources and functionality, it enables fast clocks, low power, and low design effort. Abundant transistor budgets allow the incorporation of many clusters on a chip. The instructions of a single program are distributed across the clusters, thereby enabling high parallelism. Since it is impossible to localize all dependent instructions to a single cluster, data is frequently communicated between clusters over the inter-cluster communication fabric. Depending on the workloads, different flavors of partitioned architectures can exploit instruction-level, data-level, and thread-level parallelism (ILP, DLP, and TLP).

As we move to smaller process technologies, logic delays scale down with transistor widths, while wire delays do not scale down at the same rate. To alleviate the high performance penalty of long wire delays at future technologies, most research efforts have concentrated on reducing the number of communications through intelligent instruction and data assignment to clusters. Such an assignment can be accomplished either at compile-time [21, 27, 30, 35, 36, 39, 42] or at run-time [1, 7, 10, 15, 18]. However, in spite of our best efforts, global communication is here to stay. For a dynamically scheduled 4-cluster system (described in Sections 4 and 5), performance degrades by 12% when the inter-cluster latency is doubled. The papers listed above also report similar slowdowns for high-latency interconnects. Thus, irrespective of the implementation, partitioned architectures experience a large number of global data transfers and performance can be severely degraded if the interconnects are not optimized for low delay.

Since global communications happen on long wires with high capacitances, they are responsible for a significant fraction of on-chip power dissipation. Interconnect power is a major problem not only in today's industrial designs, but also in high-performance research prototypes. A recent evaluation by Wang *et al.* [44] demonstrates that the intertile network accounts for 36% of the total energy dissipated in the Raw processor [42]. A recent report by Magen *et al.* [32] also attributes 50% of total chip power in an Intel processor to interconnects. We are clearly moving to an era

where movement of data on a chip can have greater impact on performance and energy than computations involving the data. In other words, future microprocessors are becoming increasingly *communication-bound*.

VLSI techniques enable a variety of different wire implementations. For example, by tuning the wire width and spacing, we can design wires with varying latency and bandwidth properties. Similarly, by tuning repeater size and spacing, we can design wires with varying latency and energy properties. Further, as interconnect technology develops, transmission lines may become feasible, enabling very low latency for very low-bandwidth communication. Data transfers on the on-chip network also have different requirements – some transfers benefit from a low latency network, others benefit from a high bandwidth network, and yet others are latency insensitive. To take advantage of VLSI techniques and to better match interconnect design to communication requirements, we propose a *heterogeneous interconnect*, where every link consists of wires that are optimized for either latency, energy, or bandwidth. We propose novel mechanisms that can take advantage of these interconnect choices to improve performance *and* reduce energy consumption.

To exploit a low-latency, low-bandwidth interconnect, we design a cache pipeline that employs a subset of the address bits to prefetch data out of cache banks. We also take advantage of the fact that a number of data transfers involve narrow bit-width operands that can benefit from a low-bandwidth interconnect. Further, we can see improved performance by diverting bursts of interconnect traffic to high-bandwidth high-latency interconnects. These high-bandwidth interconnects can also be designed to be energy-efficient, enabling significant energy savings in addition to performance improvements.

The paper is organized as follows. Section 2 reviews techniques that enable different wire implementations. Section 3 outlines the design of a heterogeneous interconnect. Section 4 describes the proposed innovations to exploit different on-chip wires and they are evaluated in Section 5. Finally, Section 6 discusses related work and we conclude in Section 7.

2. Wire Implementations with Varying Characteristics

The delay of a wire is a function of the RC time constant (R is resistance and C is capacitance). The resistance per unit length of the wire can be expressed by the following equation [25]:

$$R_{wire} = \frac{\rho}{(thickness - barrier)(width - 2\ barrier)} \tag{1}$$

$Thickness$ and $width$ represent the geometrical dimensions of the wire cross-section, $barrier$ represents the thin barrier layer around the wire to prevent copper from diffusing into surrounding oxide, and ρ is the material resistivity. The capacitance per unit length can be modeled by four parallel-plate capacitors for each side of the wire and a constant for fringing capacitance [25]:

$$C_{wire} = \epsilon_0(2K\epsilon_{horiz}\frac{thickness}{spacing} + 2\epsilon_{vert}\frac{width}{layerspacing})$$

$$+fringe(\epsilon_{horiz}, \epsilon_{vert}) \tag{2}$$

The potentially different relative dielectrics for the vertical and horizontal capacitors are represented by ϵ_{horiz} and ϵ_{vert}, K accounts for Miller-effect coupling capacitances, $spacing$ represents the gap between adjacent wires on the same metal layer, and $layerspacing$ represents the gap between adjacent metal layers. We now examine the techniques that enable wires with varying properties.

Wire Width and Spacing

As can be seen from Equation (1), increasing the width of the wire can significantly decrease resistivity, while also resulting in a modest increase in capacitance per unit length (Equation (2)). Similarly, increasing the spacing between adjacent wires results in a drop in C_{wire}. By allocating more metal area per wire and increasing the wire width and spacing, the overall effect is that the product of R_{wire} and C_{wire} decreases, resulting in lower wire delays. The primary difference between wires in the different types of metal layers in modern processors is the wire width and spacing (in addition to the thickness). Ho *et al.* [25] report that a 10mm unbuffered wire at 180nm technology has delays of 57 FO4s, 23 FO4s, and 6 FO4s on local, semi-global, and global wires. Thus, wire width and spacing are powerful parameters that can vary the latency by at least a factor of 10. However, wide wires are more suited for low-bandwidth traffic such as for clock and power distribution. If global communication involves the transfer of 64-bit data between clusters, employing 64 wide wires can have enormous area overheads. For a given metal area, the wider the wire, the fewer the number of wires that can be accommodated (see Figure 1). Hence, optimizing a wire for low delay by designing wide wires has a negative impact on bandwidth.

Repeater Size and Spacing

The resistance and capacitance of a wire are both linear functions of the wire length. Hence, the delay of a wire, that depends on the product of wire resistance and capacitance, is a quadratic function of wire length. A simple technique to overcome this quadratic dependence is to break the wire into multiple smaller segments and connect them with repeaters [5]. As a result, wire delay becomes a
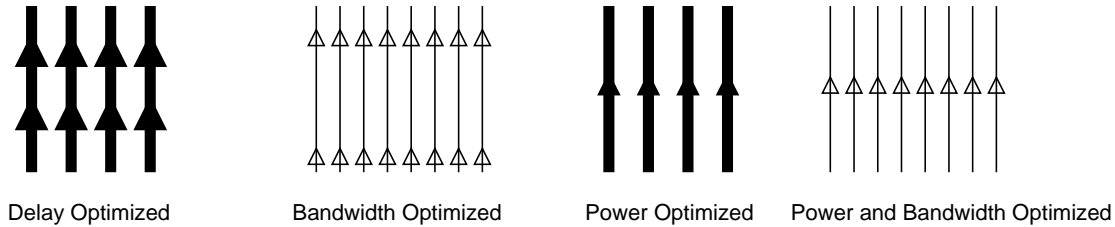
Figure 1. Examples of different wire implementations. Energy optimized wires have fewer and smaller repeaters, while bandwidth optimized wires have narrow widths and spacing.

linear function of wire length and depends on the number of segments, the wire delay across each segment, and the logic delay across each repeater. Overall wire delay can be minimized by selecting optimal repeater sizes and spacing between repeaters [5] and this technique is commonly employed in modern-day processors. However, these repeaters have high overheads associated with them. Contacts have to be cut from the metal layer to the silicon substrate every time a logic element is introduced in the middle of a wire. The contacts and the transistors not only impose area overheads and routing constraints, but also impose high capacitive loads on the wires. Banerjee *et al.* [8, 9] report that sub-100nm designs will have over a million repeaters and that optimally sized repeaters are approximately 450 times the minimum sized inverter at that technology point.

Energy in the interconnect can be reduced by employing repeaters that are smaller than the optimally-sized repeaters and by increasing the spacing between successive repeaters (see Figure 1). This increases overall wire delay. Recently, Banerjee *et al.* [8] developed a methodology to estimate repeater size and spacing that minimizes power consumption for a fixed wire delay. They show that at 50nm technology, it is possible to design a repeater configuration such that the wire has twice the delay and $1/5^{th}$ the energy of a wire that is delay-optimal. Thus, repeater size and spacing are parameters that can dramatically influence interconnect power and performance.

Transmission Lines

In future technologies, other promising wire implementations may become feasible, such as transmission lines [16, 19]. In a transmission line, the wire delay is determined by the time taken to detect a voltage ripple on the wire. This delay is determined by the LC time constant and the velocity of the ripple, which is a function of the speed of light in the dielectric surrounding the interconnect. A transmission line, therefore, enables very low wire latencies. For a wire to operate as a transmission line, it must have very high width, thickness, horizontal and vertical spacing, and signal frequency. There are other implementation issues as well, such as the design of signal modulation and sensing

circuits, reference planes above and below the metal layer, and shielding power and ground lines adjacent to each transmission line [12].

Because of the large area requirements and other associated costs, transmission lines have been sparsely used in modern processors, usually as single wires for clock distribution [33, 45, 46]. They have also been shown to work in other test CMOS chips [16, 20]. As we move to higher clock frequencies and increasing metal layers, transmission line implementations may become more practical and cost-effective. However, owing to the high area requirements per wire, transmission lines are likely to be feasible only for very low bandwidth communication. Thus, a transmission line represents another interesting wire implementation that trades off bandwidth for extremely low latencies.

3. Heterogeneous Interconnects

From the above discussion, it is clear that a large number of different wire implementations are possible, either by varying properties such as wire width/spacing and repeater size/spacing, or by employing transmission lines. Typically, inter-cluster global interconnects are designed to minimize delay for the transfer of 64-bit data and the associated tags (typically fewer than eight bits). Wire widths are chosen such that 72 wires can be accommodated in the available metal area and repeaters are sized and spaced to optimize delay. We refer to these wires as *B-Wires*. In addition to this base 72-bit interconnect, there are at least three other interesting wire implementations that the architecture can benefit from:

- *P-Wires:* Wires that are power-optimal. The wires have longer delays as they employ small repeater sizes and wide repeater spacing.

- *W-Wires:* Wires that are bandwidth-optimal. The wires have minimum width and spacing and have longer delays.

- *L-Wires:* Wires that are latency-optimal. These wires operate as transmission lines or employ very wide

width and spacing and have low bandwidth (potentially, a network with fewer than 20 bits).

To limit the range of possibilities, *P-Wires* and *W-Wires* can be combined to form a single wire implementation with minimum width and spacing and with small repeater sizes and wide repeater spacing. Such wires have poor delay characteristics, but allow low power and high bandwidth (referred to as *PW-Wires*).

These options can be incorporated into the inter-cluster global interconnect in a variety of ways. In this evaluation, we limit ourselves to the following topology. Every link on the network offers the same degree of heterogeneity. For example, every link may consist of 72 *B-Wires*, 144 *PW-Wires*, and 18 *L-Wires*. Thus, for any data transfer, the microarchitecture can dynamically choose to effect the transfer on either *B-Wires*, *PW-Wires*, or *L-Wires*. Such an implementation choice may entail additional complexity within the send buffers. To reduce this complexity, one can design a topology where some links consist entirely of *PW-Wires*, while others consist entirely of *B-Wires*. Such a topology has lower design complexity, but affords less flexibility to the microarchitecture. In this paper, we adopt the former implementation since we are evaluating the potential of a heterogeneous interconnect. We argue that the additional logic to route data to one of three possible interconnects only requires one-to-three demultiplexers and compared to a base processor that already has multiple interconnects, the overhead is likely to be negligible. We assume a model where in any cycle, data can be dynamically assigned to one of the available wire implementations based on the program's needs. The mechanisms that enable this dynamic decision-making are discussed in the next section.

Different wire implementations may or may not be accommodated on a single metal layer. There are no technological barriers to having wires with different width/spacing or repeater size/spacing on a single metal layer. However, aspect ratio guidelines necessitate that wire width can not be reduced beyond a certain limit. If the width and spacing for *W-Wires* or *PW-Wires* is lower than this limit, they may have to be implemented on a lower metal layer that has smaller thickness. Further, current implementations of transmission lines are rather cumbersome and entail huge overheads. Unless these implementations become more cost-effective, the use of transmission lines will involve additional metal layers. The International Technology Roadmap for Semiconductors [3] projects that the number of metal layers will increase in future generations. Evaluations of this nature help identify the most promising ways to exploit such a resource.

4. Exploiting Heterogeneous Interconnects

In this section, we describe the partitioned architecture model that serves as an evaluation platform for this study and the proposed innovations that can take advantage of a heterogeneous interconnect.

The Baseline Partitioned Architecture

Instruction assignment to clusters in a partitioned architecture may happen at compile-time [11, 21, 27, 30, 35, 36, 39], or at run-time [1, 7, 10, 15, 18]. There are advantages to either approach – static techniques entail lower hardware overheads and have access to more information on program dataflow, while dynamic techniques are more reactive to events such as branch mispredicts, cache misses, network congestion, etc. Our evaluations employ a dynamically scheduled partitioned architecture. We expect that our proposals can be applied even to statically scheduled architectures.

Our partitioned architecture model dispatches a large window of in-flight instructions from a single-threaded application. We adopt a centralized cache implementation because earlier studies have shown that a centralized cache offers nearly as much performance as a distributed cache while enabling low implementation complexity [6, 23, 38]. The assignment of instructions to clusters happens through a state-of-the-art dynamic instruction steering heuristic [7, 15, 43] that takes the following information into account: data dependences, cluster load imbalance, criticality of operands, and proximity to the data cache. While dispatching an instruction, the steering algorithm assigns weights to each cluster to determine the cluster that is most likely to minimize communication and issue-related stalls. Weights are assigned to a cluster if it produces input operands for the instruction and if it has many empty issue queue entries. Additional weights are assigned to a cluster if it is the producer of the input operand that is predicted to be on the critical path for the instruction's execution. For loads, more weights are assigned to clusters that are closest to the data cache. The steering algorithm assigns the instruction to the cluster that has the most weights. If that cluster has no free register and issue queue resources, the instruction is assigned to the nearest cluster with available resources.

Results produced within a cluster are bypassed to consumers in that cluster in the same cycle, while communicating the result to consumers in other clusters takes additional cycles. In order to effect the transfer of data between clusters, the instruction decode and rename stage inserts a "copy instruction" [15] in the producing cluster that places the value on the inter-cluster network as soon as the value is made available. Each cluster has a scheduler for the inter-cluster network that is similar in organization to the issue queue and that has an issue bandwidth that matches the maximum number of transfers possible on each link of the network. Similar to the instruction wake-up process in conventional dynamic superscalars, the register tags for the operand are sent on the network ahead of the data so that the
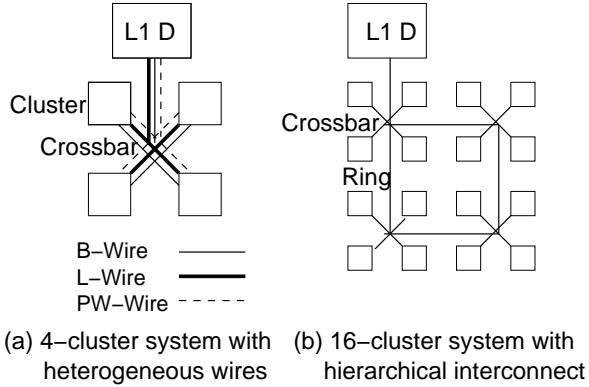
Figure 2. (a) A partitioned architecture model with 4 clusters and a heterogeneous interconnect comprised of *B-*, *L-*, and *PW-Wires*. (b) A 16-cluster system with a hierarchical interconnect. Sets of four clusters are connected with a crossbar and the crossbars are connected in a ring topology.

dependent instruction can be woken up and can consume the value as soon as it arrives.

For most of our experiments, we assume a processor model that has four clusters. These four clusters and the centralized data cache are connected through a crossbar network, as shown in Figure 2 (a). All links contain a uni-directional interconnect in each direction. The processor model in Figure 2 (a) adopts a heterogeneous interconnect where every link in the network is comprised of *B-Wires*, *PW-Wires*, and *L-Wires*. Note that every data transfer has the option to use any one of these sets of wires. Our evaluations show the effects of using interconnects that employ different combinations of these sets of wires. For all processor organizations, the bandwidth requirements to the cache are much higher than bandwidth requirements to the clusters since more than one third of all instructions are loads or stores. Hence, the links going in and out of the cache are assumed to have twice as much area and twice as many wires as the links going in and out of a cluster. If multiple transfers compete for a link in a cycle, one transfer is effected in that cycle, while the others are buffered. We assume unbounded buffers at each node of the network. An earlier study [37] has shown that these buffers typically require a modest number of entries. We also examine aggressive processor models with 16 clusters. For a 16-cluster system, we adopt a hierarchical topology similar to the one proposed by Aggarwal and Franklin [2]. As shown in Figure 2 (b), a set of four clusters is connected through a crossbar, allowing low-latency communication to neighboring clusters. The crossbars are connected with a ring topology. Similar to the 4-cluster system, every link in the network is comprised of

wires with different properties.

Accelerating Cache Access

First, we examine how low-latency low-bandwidth *L-Wires* can be exploited to improve performance. *L-Wires* are designed by either employing very large wire widths and spacing or by implementing transmission lines. Because of the area overhead, for the sake of this discussion, we assume that 18 *L-Wires* occupy the same metal area as 72 *B-Wires*.

Consider the behavior of the cache pipeline in the baseline processor. When a cluster executes a load instruction, it computes the effective address and communicates it to the centralized load/store queue (LSQ) and cache. The load/store queue waits until it receives addresses of stores prior to the load in program order, guarantees that there is no memory dependence, and then initiates the cache access. The cost of communication to the cache influences load latency in two ways – (i) it delays the arrival of load addresses at the LSQ, (ii) it delays the arrival of store addresses at the LSQ, thereby delaying the resolution of memory dependences.

To accelerate cache access, we propose the following novel technique. A subset of the address bits are transmitted on low-latency *L-Wires* to prefetch data out of the L1 cache and hide the high communication cost of transmitting the entire address. After the cluster computes the effective address, the least significant (LS) bits of the address are transmitted on the low-latency *L-Wires*, while the most significant (MS) bits are transmitted on *B-Wires*. The same happens for store addresses. Thus, the LSQ quickly receives the LS bits for loads and stores, while the MS bits take much longer. The early arrival of the partial addresses allows the following optimizations.

The LSQ can effect a partial comparison of load and store addresses with the available LS bits. If the LS bits of the load do not match the LS bits of any earlier store, the load is guaranteed to not have any memory dependence conflicts and it can begin cache access. If the LS bits of the load match the LS bits of an earlier store, it has to wait for the MS bits to arrive before determining if there is a true dependence. A large number of false dependences can also increase contention for the LSQ ports. Fortunately, we found that false dependences were encountered for fewer than 9% of all loads when employing eight LS bits for the partial address comparison.

To effect an L1 data cache access, the least significant bits of the effective address are used to index into the data and tag RAM arrays and read out a relevant set of cache blocks. The most significant bits of the effective address are used to index into the TLB and the resulting translation is then compared with the tags to select the appropriate data block and forward it to the cluster. Since the accesses to the cache RAM arrays do not require the most significant bits, the accesses can be initiated as soon as the least significant

bits of the address arrive on *L-Wires* (provided the *L-Wires* transmit enough bits to determine the set index).

Similarly, a few bits of the virtual page number can be included in the transfer on the *L-Wires*. This allows TLB access to proceed in parallel with RAM array look-up. The modifications to enable indexing with partial address information are more significant for a CAM structure than a RAM structure. Hence, a highly-associative TLB design may be more amenable to this modified pipeline than a fully-associative one. When the rest of the effective address arrives, tag comparison selects the correct translation from a small subset of candidate translations.

Thus, the transfer of partial address bits on *L-Wires* enables data to be prefetched out of L1 cache and TLB banks and hide the RAM access latency, which is the biggest component in cache access time. If the cache RAM access has completed by the time the entire address arrives, only an additional cycle is spent to detect the correct TLB translation and effect the tag comparison before returning data to the cluster. This overlap of effective address transfer with cache RAM and TLB access can result in a reduction in effective load latency if the latency difference between *L-Wires* and *B-Wires* is significant.

It must be noted that the proposed pipeline works well and yields speedups even if the processor implements some form of memory dependence speculation. The partial address can proceed straight to the L1 cache and prefetch data out of cache banks without going through partial address comparisons in the LSQ if it is predicted to not have memory dependences. To allow cache and TLB index bits to fit in a narrow low-bandwidth interconnect, it might be necessary to make the cache and TLB highly set-associative. For example, 18 *L-Wires* can accommodate 6 bits of tag to identify the instruction in the LSQ, 8 index bits for the L1 data cache, and 4 index bits for the TLB. For the assumed cache and TLB sizes, this corresponds to an associativity of 4 and 8 for the cache and TLB, respectively. If the associativity is reduced, we may need a few more *L-Wires*.

Narrow Bit-Width Operands

An interconnect composed of *L-Wires* can also be employed for results that can be encoded by a few bits. 18 *L-Wires* can accommodate eight bits of register tag and ten bits of data. We employ the simplest form of data compaction here – integer results between 0 and 1023 are eligible for transfer on *L-Wires*. The hardware required to detect narrow bit-width data can be easily implemented – the PowerPC 603 [22] has hardware to detect the number of leading zeros that is then used to determine the latency for integer multiply. A special case in the transfer of narrow bit-width data is the communication of a branch mispredict back to the front-end. This only involves the branch ID that can be easily accommodated on *L-Wires*, thereby reducing the branch mispredict penalty.

Other forms of data compaction might also be possible, but is not explored here. For example, Yang *et al.* [47] identify that the eight most frequent values in SPEC95-Int programs account for roughly 50% of all data cache accesses and can be easily encoded by a few bits.

In order to schedule a wake-up operation at the consuming cluster, the register tags are sent before the data itself. For a narrow bit-width operand, the tags have to be sent on *L-Wires*. Hence, the pipeline requires advance knowledge of whether the result can be expressed in 10 bits. For our evaluations, we make the optimistic assumption that this information is available early in the pipeline. A realistic implementation would require inspection of the instruction's input operands or a simple predictor. We confirmed that a predictor with 8K 2-bit saturating counters, that predicts the occurrence of a narrow bit-width result when the 2-bit counter value is three, is able to identify 95% of all narrow bit-width results. With such a high-confidence predictor, only 2% of all results predicted to be narrow have bit widths greater than 10.

Exploiting PW-Wires

Next, we examine how *PW-Wires* can be employed to not only reduce contention in other wires, but also reduce energy consumption. Our objective here is to identify those data transfers that can tolerate the higher latency of these wires or to identify situations when the cost of contention on *B-Wires* offsets its wire latency advantage. If a data transfer has the choice of using either *B-Wires* or *PW-Wires*, the following three criteria dictate when a transfer can be effected on the high bandwidth, low energy, high latency *PW-Wires*:

- If the input operands are already ready in a remote register file at the time an instruction is dispatched, the operands are transferred to the instruction's cluster on *PW-Wires*. The rationale here is that there is usually a long gap between instruction dispatch and issue and the long communication latency for the ready input operand can be tolerated.

- Store data is assigned to *PW-Wires*. This can slow the program down only if the store is holding up the commit process or if there is a waiting dependent load. Both are fairly rare cases and we noticed a minimal performance impact from adopting this policy.

- We keep track of the amount of traffic injected into either interconnect in the past N cycles (N=5 in our simulations). If the difference between the traffic in each interconnect exceeds a certain pre-specified threshold (10 in our simulations), subsequent data transfers are steered to the less congested interconnect.

Thus, by steering non-critical data towards the high-bandwidth energy-efficient interconnect, we are likely to see little performance degradation and by steering data away from the congested interconnect, we can potentially

Fetch queue size	64	Branch predictor	comb. of bimodal and 2-level
Bimodal predictor size	16K	Level 1 predictor	16K entries, history 12
Level 2 predictor	16K entries	BTB size	16K sets, 2-way
Branch mispredict penalty	at least 12 cycles	Fetch width	8 (across up to 2 basic blocks)
Issue queue size	15 per cluster (int and fp, each)	Register file size	32 per cluster (int and fp, each)
Integer ALUs/mult-div	1/1 per cluster	FP ALUs/mult-div	1/1 per cluster
L1 I-cache	32KB 2-way	Memory latency	300 cycles for the first block
L1 D-cache	32KB 4-way set-associative,	L2 unified cache	8MB 8-way, 30 cycles
	6 cycles, 4-way word-interleaved	I and D TLB	128 entries, 8KB page size

Table 1. Simplescalar simulator parameters.

Wire Implementation	Relative delay	Crossbar latency	Ring hop latency	Relative leakage	Relative dynamic
W-Wires	1.0			1.00	1.00
PW-Wires	1.2	3 cycles	6 cycles	0.30	0.30
B-Wires	0.8	2 cycles	4 cycles	0.55	0.58
L-Wires	0.3	1 cycle	2 cycles	0.79	0.84

Table 2. Wire delay and relative energy parameters for each RC-based wire.

see performance improvements. Most importantly, we can observe large savings in interconnect energy.

5. Results

5.1. Methodology

Our simulator is based on Simplescalar-3.0 [14] for the Alpha AXP ISA. Separate issue queues and physical register files for integer and floating-point streams are modeled for each cluster. Contention on the interconnects and for memory hierarchy resources (ports, banks, buffers, etc.) are modeled in detail. We assume that each cluster has 32 registers (int and fp, each), 15 issue queue entries (int and fp, each), and one functional unit of each kind. While we use a large ROB size of 480, in-flight instruction windows are typically much smaller as dispatch gets stalled as soon as the processor runs out of physical registers or issue queue entries. Our evaluations show results for processor models with four and sixteen clusters. Important simulation parameters are listed in Table 1.

We use 23 of the 26 SPEC-2k programs with reference inputs as a benchmark set [1]. Each program was simulated for 100 million instructions over simulation windows identified by the Simpoint toolkit [40]. Detailed simulation was carried out for one million instructions to warm up various processor structures before taking measurements.

5.2. Latency and Energy Estimates

It can be shown that the delay of a wire with optimal repeater placement is directly proportional to $\sqrt{RC}$ [8, 24,

25, 34]. When the width of a wire is increased by a factor X, resistance decreases by a factor X, while capacitance increases slightly. We start by assuming that *W-Wires* have the minimum allowed width and spacing for the selected metal layer. We then design a *PW-Wire* by reducing the size and number of repeaters. According to the methodology proposed by Banerjee and Mehrotra [8], roughly 70% of interconnect energy can be saved at 45nm technology while incurring a 20% delay penalty. We design *B-Wires* such that each wire has twice as much metal area as a *PW-Wire* and its delay is lower by a factor of 1.5. We were able to meet the delay constraint by keeping the width the same as a *W-Wire* and only increasing wire spacing. This strategy also helps us reduce the power consumed in *B-Wires*. Finally, *L-Wires* were designed by increasing the width and spacing of *W-Wires* by a factor of 8. Based on the analysis of Banerjee *et al.* [8, 34], we compute that at 45nm technology, $R_L = 0.125R_W$, $C_L = 0.8C_W$, giving us the result that $Delay_L = 0.3Delay_W = 0.25Delay_{PW}$. If we instead implement *L-Wires* as transmission lines, the improvement in wire delay will be much more. Chang *et al.* [16] report that at 180nm technology, a transmission line is faster than an RC-based repeated wire of the same width by a factor of 4/3. This gap may widen at future technologies. For the purposes of our evaluation, we restrict ourselves to RC-based models, but point out that performance and energy improvements can be higher if transmission lines become a cost-effective option.

We assume that communication with neighbors through the crossbar takes three cycles for *PW-Wires*. Based on the relative latency estimates above, *B-Wires* and *L-Wires* are 1.5 times and 4 times faster than *PW-Wires*, corresponding to inter-cluster communication latencies of two cycles and one cycle, respectively. When examining a 16-cluster

---

[1] *Sixtrack*, *Facerec*, and *Perlbmk* were not compatible with our simulation infrastructure.
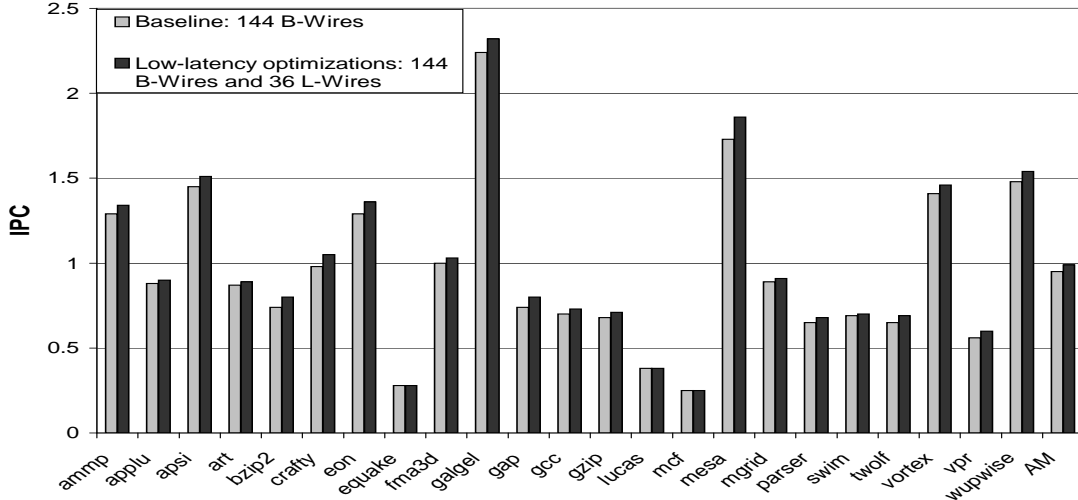
Figure 3. IPCs for the baseline 4-cluster partitioned architecture employing one layer of *B-Wires* and for a partitioned architecture employing one layer of *B-Wires* and one layer of *L-Wires*. The *L-Wires* transmit narrow bit-width data, branch mispredict signals, and LS bits of load/store addresses.

system with a hierarchical interconnect, the latency for a hop on the ring interconnect for *PW-Wires*, *B-Wires*, and *L-Wires* are assumed to be 6, 4, and 2 cycles. The various wire parameters are summarized in Table 2. We assume that all transfers are fully pipelined.

Our energy estimates for 45nm are derived from the analysis described by Banerjee *et al.* [8, 34]. Relative dynamic and leakage energy values are listed in Table 2. Different wire width and spacing cause energy differences in *W-*, *B-*, and *L-Wires*, while smaller and fewer repeaters cause a 70% energy decrease between *W-Wires* and *PW-Wires*. Chang *et al.* [16] report a factor of three reduction in energy consumption by employing transmission line technology. Thus, low-bandwidth transfers effected on *L-Wires* can not only improve performance, but also reduce energy consumption. For our evaluations, we restrict ourselves to RC-based *L-Wires*.

Our analysis does not model the power consumed within the schedulers for the inter-cluster network. Heterogeneity will likely result in negligible power overhead in the schedulers while comparing networks with equal issue bandwidth.

5.3. Behavior of L-Wires

We first examine how *L-Wires* enable the optimizations described in Section 4. Figure 3 shows IPCs for SPEC2k programs for two 4-cluster systems. The first is our baseline organization that has only one interconnect layer comprised entirely of *B-Wires*. Each link to a cluster can transfer 72 bits of data and tag in each direction, while the link to the

data cache can transfer 144 bits in each direction. In the second 4-cluster system shown in Figure 3, the baseline interconnect is augmented with another metal layer that is comprised entirely of *L-Wires*. Each link to a cluster can transfer 18 bits of data and tag in each direction and the link to the cache can transfer 36 bits in each direction. The *L-Wires* are employed to send the LS bits of a load or store effective address, for the transfer of narrow bit-width data, and for the transfer of branch mispredict signals. We see that overall performance improves by only 4.2%, while comparing the arithmetic mean (AM) of IPCs[2]. We observed that the novel cache pipeline, the transfer of narrow bit-width operands, and the transfer of branch mispredict signals, contributed equally to the performance improvement. In this particular processor model, the transfer on *L-Wires* can save at most a single cycle, yielding a modest performance improvement. Considering that the proposed pipeline entails non-trivial complexity to determine operand bit-widths and compare multiple tags at the LSQ, we believe that the performance improvement is likely not worth the design effort. However, as listed below, there may be other scenarios where *L-Wires* can yield significant benefits.

If future technology points are more wire constrained, the latency gap between *B-Wires* and *L-Wires* widens. If we assume latencies that are twice as much as those listed in Table 2, the performance improvement by adding an interconnect layer comprised of *L-Wires* is 7.1%. As transistor budgets increase, high-performance processors may em-

---

[2]The AM of IPCs represents a workload where every program executes for an equal number of cycles [26].

Model	Description of each link	Relative Metal Area	IPC	Relative interconnect dyn-energy	Relative interconnect lkg-energy	Relative Processor Energy (10%)	Relative ED^2 (10%)	Relative ED^2 (20%)
$Model-I$	144 *B-Wires*	1.0	0.95	100	100	100	100	100
$Model-II$	288 *PW-Wires*	1.0	0.92	52	112	97	103.4	100.2
$Model-III$	144 *PW-Wires*, 36 *L-Wires*	1.5	0.96	61	90	97	95.0	92.1
$Model-IV$	288 *B-Wires*	2.0	0.98	99	194	103	96.6	99.2
$Model-V$	144 *B-Wires*, 288 *PW-Wires*	2.0	0.97	83	204	102	97.8	99.6
$Model-VI$	288 *PW-Wires*, 36 *L-Wires*	2.0	0.97	61	141	99	94.4	93.0
$Model-VII$	144 *B-Wires*, 36 *L-Wires*	2.0	0.99	105	130	101	93.3	94.5
$Model-VIII$	432 *B-Wires*	3.0	0.99	99	289	106	97.2	102.4
$Model-IX$	288 *B-Wires*, 36 *L-Wires*	3.0	1.01	105	222	104	92.0	95.5
$Model-X$	144 *B-Wires*, 288 *PW-Wires*, 36 *L-Wires*	3.0	1.00	82	233	103	92.7	95.1

Table 3. Heterogeneous interconnect energy and performance for 4-cluster systems. All values (except IPC) are normalized with respect to $Model-I$. ED^2 is computed by multiplying total processor energy by square of executed cycles. 10% and 20% refer to the contribution of interconnect energy to total processor energy in $Model-I$.

ploy as many as 16 clusters. Such an aggressive architecture can not only improve thread-level parallelism (TLP), but it can also improve single-thread performance for high-ILP programs. For our base processor model with a single interconnect layer comprised of *B-Wires*, the improvement in single-thread IPC by moving from 4 to 16 clusters for the 23 SPEC-2k programs is 17%. Again, the single-thread performance improvement in moving from 4 to 16 clusters likely does not warrant the complexity increase. However, if processors are going to employ many computational units for TLP extraction, the complexity entailed in allowing a single thread to span across 16 clusters may be tolerable. For such a wire-delay-constrained 16-cluster system, the performance improvement by adding a metal layer with *L-Wires* is 7.4%. As our subsequent tables shall show, there are other processor and interconnect models where the addition of an *L-Wire* interconnect layer can improve performance by more than 10%. It is possible that the novel cache pipeline may yield higher benefits for ISAs with fewer registers that may have more loads and stores. Only 14% of all register traffic on the inter-cluster network are comprised of integers between 0 and 1023. More complex encoding schemes might be required to take additional benefit of *L-Wires*. It is also possible that there are other mechanisms to exploit low-latency low-bandwidth wires that may be more complexity-effective. For example, such wires can be employed to fetch critical words from the L2 or L3.

5.4. Heterogeneous Interconnect Choices

The above evaluation shows performance improvements by the addition of a metal layer comprised entirely of *L-Wires*. This helps gauge the potential of *L-Wires* to reduce the cost of long wire latencies, but is not a fair compari-

son because of the difference in the number of metal layers. In this subsection, we try to evaluate the best use of available metal area. We start by evaluating processor models that only have enough metal area per link to each cluster to accommodate either 144 *B-Wires*, or 288 *PW-Wires*, or 36 *L-Wires* (the link to the data cache has twice this metal area). Our base processor ($Model-I$) that effects one transfer in and out of each cluster on *B-Wires* is an example of such a processor model. We then examine processors that have twice and thrice as much metal area, allowing more interesting combinations of heterogeneous wires. Table 3 summarizes the performance and energy characteristics of interesting heterogeneous interconnect organizations for a system with four clusters. All values in Table 3 except IPC are normalized with respect to the values for $Model-I$. ED^2 is computed by taking the product of total processor energy and the square of the number of cycles to execute 100M instructions. Total processor energy assumes that interconnect energy accounts for 10% of total chip energy in $Model-I$ and that leakage and dynamic energy are in the ratio 3:7 for $Model-I$. Table 3 also shows ED^2 when assuming that interconnect energy accounts for 20% of total chip energy.

We first examine processor models that employ as much metal area as $Model-I$. The only alternative interconnect choice that makes sense is one that employs 288 *PW-Wires* for each link to a cluster ($Model-II$). We also evaluate a heterogeneous interconnect that consumes 1.5 times as much metal area as $Model-I$ by employing 144 *PW-Wires* and 36 *L-Wires* to each cluster ($Model-III$). From Table 3, we observe that only employing slower *PW-Wires* ($Model-II$) degrades IPC and increases ED^2, in spite of the increased bandwidth. $Model-III$ with *PW-Wires* and *L-Wires* allows a combination of high performance and

Model	Description of each link	IPC	Relative Processor Energy (20%)	Relative ED^2 (20%)
$Model - I$	144 *B-Wires*	1.11	100	100
$Model - II$	288 *PW-Wires*	1.05	94	105.3
$Model - III$	144 *PW-Wires*, 36 *L-Wires*	1.11	94	93.6
$Model - IV$	288 *B-Wires*	1.18	105	93.1
$Model - V$	144 *B-Wires*, 288 *PW-Wires*	1.15	104	96.5
$Model - VI$	288 *PW-Wires*, 36 *L-Wires*	1.13	97	93.2
$Model - VII$	144 *B-Wires*, 36 *L-Wires*	1.19	102	88.7
$Model - VIII$	432 *B-Wires*	1.19	111	96.2
$Model - IX$	288 *B-Wires*, 36 *L-Wires*	1.22	107	88.7
$Model - X$	144 *B-Wires*, 288 *PW-Wires*, 36 *L-Wires*	1.19	106	91.9

Table 4. Heterogeneous interconnect energy and performance for 16-cluster systems where interconnect energy contributes 20% of total processor energy in $Model - I$. **All values (except IPC) are normalized with respect to** $Model - I$.

low energy. Most transfers happen on *PW-Wires*, resulting in 30% savings in interconnect energy dissipation, while *L-Wires* enable the optimizations described in Section 4 and boost performance back up to that with the baseline interconnect ($Model - I$). Thus, in terms of overall processor ED^2, the heterogeneous interconnect allows a 5% improvement, although at an area cost.

Next, we evaluate processor models that have twice as much metal area per link as $Model - I$. $Model - IV$ accommodates 288 *B-Wires* in each link to a cluster, while $Model - V$ represents a heterogeneous interconnect that employs 144 *B-Wires* and 288 *PW-Wires*. In $Model - V$, data is assigned to *PW-Wires* according to the criteria discussed in Section 4. The higher latency of *PW-Wires* causes only a slight performance degradation of 1% compared to $Model - IV$. This is partly because our criteria are effective at identifying latency insensitive data transfers and partly because *PW-Wires* reduce overall contention by 14%. 36% of all data transfers happen on energy-efficient wires, leading to energy savings when compared with $Model - IV$. $Model - VI$ improves on energy and ED^2 by sending all of its traffic on *PW-Wires* and using *L-Wires* to offset the performance penalty. Finally, $Model - VII$ represents the high-performance option in this class, by employing *B-Wires* and *L-Wires*, yielding a decrease in ED^2 in spite of an increase in overall energy consumption. Thus, the interconnects with the best ED^2 employ combinations of different wires and not a homogeneous set of wires.

Finally, we evaluate interesting designs that have enough area per link to a cluster to accommodate 432 *B-Wires* ($Model - VIII$). The high-performance option in this class ($Model - IX$) employs 288 *B-Wires* and 36 *L-Wires*, while the low-power option ($Model - X$) accommodates, *B-*, *PW-*, and *L-* wires. While there is little performance benefit to be derived from having thrice as much metal area

as $Model - I$, it is interesting to note that heterogeneous interconnects continue to yield the best ED^2 values.

We repeated our evaluation on a 16-cluster system that is likely to be more sensitive to interconnect design choices. Table 4 summarizes the IPC, processor energy, and ED^2 values while assuming that interconnect energy accounts for 20% of total processor energy. Up to 11% reductions in ED^2 can be observed by employing heterogeneous interconnects.

In summary, our results indicate that heterogeneous wires have the potential to improve performance and energy characteristics, as compared to a baseline approach that employs homogeneous wires. We see overall processor ED^2 reductions of up to 8% for 4-cluster systems and 11% for 16-cluster systems by employing energy-efficient and low-latency wires. As previously discussed, the improvements can be higher in specific processor models or if transmission line technology becomes feasible.

There are clearly some non-trivial costs associated with the implementation of a heterogeneous interconnect, such as pipeline modifications, demultiplexing in the send buffers, logic to identify narrow bit-widths and network load imbalance, etc. The above results demonstrate the high potential of such an approach, necessitating a more careful examination of whether these overheads are tolerable.

6. Related Work

Here, we mention other related work that hasn't already been cited in context. Austin and Sohi [4] propose mechanisms to overlap cache indexing with effective address calculation. These mechanisms differ from our proposed cache pipeline in the following two major aspects: (i) they serve to hide the cost of deep pipelines and arithmetic computations, not wire delays, (ii) they employ prediction techniques.

A recent study by Citron [17] examines entropy within data being transmitted on wires and identifies opportunities for compression. The author suggests that if most traffic can be compressed, the number of wires can scale down, allowing each wire to be fatter. Unlike our proposal, the author employs a single interconnect to transfer all data and not a hybrid interconnect with different latency/bandwidth/power characteristics. A study by Loh [31] exploits narrow bitwidths to execute multiple instructions on a single 64-bit datapath. Performance improves because the effective issue width increases in some cycles. Brooks and Martonosi [13] show that in a 64-bit architecture, roughly 50% of all integer ALU operations in SPEC95-Int have both operands with bit-widths less than 16 bits. In their study, this property was exploited to reduce power consumption in integer ALUs.

The recent paper by Beckmann and Wood [12] on Transmission Line Caches is the only study that exploits low latency transmission lines at the microarchitectural level. Taylor *et al.* [41] define the inter-cluster communication fabric as a *Scalar Operand Network* and provide a detailed analysis of the properties of such a network and the effect of these properties on ILP extraction. Wang *et al.* [44] examine power consumed within on-chip interconnects, with a focus on the design of router microarchitectures. No prior architectural work has examined trade-offs in wire characteristics and the design of microarchitectures to exploit a variety of wire implementations. Thus, to the best of our knowledge, this is the first proposal of wire management at the microarchitectural level.

7. Conclusions and Future Work

The design of the inter-cluster interconnect has a significant impact on overall processor energy and performance. A single wire implementation is unable to simultaneously meet the high bandwidth, low-latency, and low-energy requirements of such an interconnect. A heterogeneous interconnect that consists of wires with different properties can better meet the varying demands of inter-cluster traffic.

The paper makes three key contributions:

- We show that a low-latency low-bandwidth network can be effectively used to hide wire latencies and improve performance.

- We show that a high-bandwidth low-energy network and an instruction assignment heuristic are effective at reducing contention cycles and total processor energy.

- We carry out a comprehensive evaluation of different combinations of heterogeneous interconnects and show that by selecting the right combination of wires, total processor ED^2 can be reduced by up to 11%, compared to a baseline processor with homogeneous interconnects.

We therefore make a case for microarchitectural wire management in future communication-bound processors. As future work, we plan to explore other applications of heterogeneous interconnects, such as in the transfer of data between different levels of the memory hierarchy.

References

[1] A. Aggarwal and M. Franklin. An Empirical Study of the Scalability Aspects of Instruction Distribution Algorithms for Clustered Processors. In *Proceedings of ISPASS*, 2001.

[2] A. Aggarwal and M. Franklin. Hierarchical Interconnects for On-Chip Clustering. In *Proceedings of IPDPS*, April 2002.

[3] S. I. Association. International Technology Roadmap for Semiconductors 2001. url:http://public.itrs.net/Files/2001ITRS/Home.htm.

[4] T. M. Austin and G. Sohi. Zero-Cycle Loads: Microarchitecture Support for Reducing Load Latency. In *Proceedings of MICRO-28*, November 1995.

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

[6] R. Balasubramanian. Cluster Prefetch: Tolerating On-Chip Wire Delays in Clustered Microarchitectures. In *Proceedings of ICS-18*, June 2004.

[7] R. Balasubramanian, S. Dwarkadas, and D. Albonesi. Dynamically Managing the Communication-Parallelism Trade-Off in Future Clustered Processors. In *Proceedings of ISCA-30*, pages 275–286, June 2003.

[8] 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.

[9] K. Banerjee, A. Mehrotra, A. Sangiovanni-Vincentelli, and C. Hu. On Thermal Effects in Deep Submicron VLSI Interconnects. In *Proceedings of the Design Automation Conference*, pages 885–891, 1999.

[10] A. Baniasadi and A. Moshovos. Instruction Distribution Heuristics for Quad-Cluster, Dynamically-Scheduled, Superscalar Processors. In *Proceedings of MICRO-33*, pages 337–347, December 2000.

[11] R. Barua, W. Lee, S. Amarasinghe, and A. Agarwal. Maps: A Compiler-Managed Memory System for Raw Machines. In *Proceedings of ISCA-26*, May 1999.

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

[13] D. Brooks and M. Martonosi. Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance. In *Proceedings of HPCA-5*, January 1999.

[14] D. Burger and T. Austin. The Simplescalar Toolset, Version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[15] R. Canal, J. M. Parcerisa, and A. Gonzalez. Dynamic Cluster Assignment Mechanisms. In *Proceedings of HPCA-6*, pages 132–142, January 2000.

[16] 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.

[17] D. Citron. Exploiting Low Entropy to Reduce Wire Delay. *IEEE Computer Architecture Letters*, vol.2, January 2004.

[18] J. Collins and D. Tullsen. Clustered Multithreaded Architectures – Pursuing Both IPC and Cycle Time. In *Proceedings of the 18th IPDPS*, April 2004.

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

[20] A. Deutsch. Electrical Characteristics of Interconnections for High-Performance Systems. *Proceedings of the IEEE*, 86(2):315–355, Feb 1998.

[21] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic. The Multicluster Architecture: Reducing Cycle Time through Partitioning. In *Proceedings of MICRO-30*, pages 149–159, December 1997.

[22] 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.

[23] E. Gibert, J. Sanchez, and A. Gonzalez. Flexible Compiler-Managed L0 Buffers for Clustered VLIW Processors. In *Proceedings of MICRO-36*, December 2003.

[24] R. Ho, J. Gainsley, and R. Drost. Long Wires and Asynchronous Control. In *Proceedings of ASYNC-10*, April 2004.

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

[26] L. John. More on Finding a Single Number to Indicate Overall Performance of a Benchmark Suite. *ACM Computer Architecture News*, 32(1), March 2004.

[27] U. Kapasi, W. Dally, S. Rixner, J. Owens, and B. Khailany. The Imagine Stream Processor. In *Proceedings of ICCD*, September 2002.

[28] S. Keckler and W. Dally. Processor Coupling: Integrating Compile Time and Runtime Scheduling for Parallelism. In *Proceedings of ISCA-19*, pages 202–213, May 1992.

[29] R. Kessler. The Alpha 21264 Microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[30] H.-S. Kim and J. Smith. An Instruction Set and Microarchitecture for Instruction Level Distributed Processing. In *Proceedings of ISCA-29*, May 2002.

[31] G. Loh. Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth. In *Proceedings of MICRO-35*, November 2002.

[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] M. Minzuno, K. Anjo, Y. Sume, M. Fukaishi, H. Wakabayashi, T. Mogami, T. Horiuchi, and M. Yamashina. Clock Distribution Networks with On-Chip Transmission Lines. In *Proceedings of the IEEE International Interconnect Technology Conference*, pages 3–5, 2000.

[34] 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.

[35] R. Nagarajan, K. Sankaralingam, D. Burger, and S. Keckler. A Design Space Evaluation of Grid Processor Architectures. In *Proceedings of MICRO-34*, pages 40–51, December 2001.

[36] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K.-Y. Chang. The Case for a Single-Chip Multiprocessor. In *Proceedings of ASPLOS-VII*, October 1996.

[37] J.-M. Parcerisa, J. Sahuquillo, A. Gonzalez, and J. Duato. Efficient Interconnects for Clustered Microarchitectures. In *Proceedings of PACT*, September 2002.

[38] P. Racunas and Y. Patt. Partitioned First-Level Cache Design for Clustered Microarchitectures. In *Proceedings of ICS-17*, June 2003.

[39] J. Sanchez and A. Gonzalez. Modulo Scheduling for a Fully-Distributed Clustered VLIW Architecture. In *Proceedings of MICRO-33*, pages 124–133, December 2000.

[40] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of ASPLOS-X*, October 2002.

[41] M. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures. In *Proceedings of HPCA-9*, February 2003.

[42] M. Taylor, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, J. Kim, J. Psota, A. Raraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *Proceedings of ISCA-31*, June 2004.

[43] E. Tune, D. Liang, D. Tullsen, and B. Calder. Dynamic Prediction of Critical Path Instructions. In *Proceedings of HPCA-7*, pages 185–196, January 2001.

[44] H.-S. Wang, L.-S. Peh, and S. Malik. Power-Driven Design of Router Microarchitectures in On-Chip Networks. In *Proceedings of MICRO-36*, December 2003.

[45] 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. *IBM Journal of Research and Development*, 46(1):27–51, Jan 2002.

[46] T. Xanthopoulos, D. Bailey, A. Gangwar, M. Gowan, A. Jain, and B. Prewitt. The Design and Analysis of the Clock Distribution Network for a 1.2GHz Alpha Microprocessor. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 402–403, 2001.

[47] J. Yang, Y. Zhang, and R. Gupta. Frequent Value Compression in Data Caches. In *Proceedings of MICRO-33*, pages 258–265, December 2000.