Microarchitectural Techniques to Reduce Interconnect Power in Clustered Processors

Karthik Ramani[†], Naveen Muralimanohar[†], Rajeev Balasubramonian [‡]
[†] Department of Electrical and Computer Engineering
[‡] School of Computing
University of Utah

Abstract

The paper presents a preliminary evaluation of novel techniques that address a growing problem – power dissipation in on-chip interconnects. Recent studies have shown that around 50% of the dynamic power consumption in modern processors is within on-chip interconnects. The contribution of interconnect power to total chip power is expected to be higher in future communication-bound billion-transistor architectures. In this paper, we propose the design of a heterogeneous interconnect, where some wires are optimized for low latency and others are optimized for low power. We show that a large fraction of on-chip communications are latency insensitive. Effecting these non-critical transfers on low-power long-latency interconnects can result in significant power savings without unduly affecting performance. Two primary techniques are evaluated in this paper: (i) a dynamic critical path predictor that identifies results that are not urgently consumed, and (ii) an address prediction mechanism that requires addresses to be transferred off the critical path for verification purposes. Our results demonstrate that 49% of all interconnect transfers can be effected on power-efficient wires, while incurring a performance penalty of only 2.5%.

1. Introduction

The shrinking of process technologies has enabled huge transistor budgets on a single chip. To exploit these transistors for high performance processing, numerous partitioned architectures have been proposed [9, 15, 17, 18, 19, 20]. A partitioned architecture employs small processing cores (also referred to as clusters) with an interconnect fabric and distributes instructions of a single application across the processing cores. By implementing small cores, fast clock speeds and low design complexity can be achieved. Since a single application is distributed across the clusters, partitioned architectures inevitably entail frequent data transfers across the chip. Thus, future processors are likely to be extremely *communication-bound*, from the point of view of performance and power.

Studies [1, 14, 19] have shown that wire delays do not scale down at the same rate as logic delays. As a result, the delay to send a signal across the diameter of a chip will soon be of the order of 30 cycles. These long wire delays serve as a serious performance limiter at future technology generations. Further, it has been shown that on-chip interconnects account for roughly 50% of the total power dissipation in modern processors [16]. Thus, the design of the interconnect fabric has a strong influence on processor performance and power.

This paper examines performance and power tradeoffs in the design of the inter-cluster communication network. Most evaluations on partitioned or clustered architectures have focused on instruction distribution algorithms that minimize communication and load imbalance. However, power optimizations of the interconnect at the microarchitectural level have received little attention. We propose and evaluate microarchitectural techniques that can exploit a heterogeneous network with varying performance and power characteristics.

A performance-centric approach attempts to optimize wires to minimize delay. This entails the use of optimally spaced repeaters, large drivers, and wide wires, all of which increase power dissipation in the in-

terconnect. By optimizing wires for power efficiency, performance is compromised. For example, by eliminating repeaters, wire delay becomes a quadratic function of wire length. To alleviate power consumption bottlenecks in future processor generations, we propose the design of a heterogeneous network, where half the wires are optimized for delay and the other half for power. We demonstrate that data transfers on the interconnect fabric have varying delay requirements. Many transfers are not on the program critical path and can tolerate longer communication delays. By effecting these transfers on the power-efficient network, significant reductions in power are observed with minimal impact on performance. We identify two major sources of non-critical transfers - (i) values that are not urgently sourced by consuming instructions, and (ii) values that are being transferred to verify address predictions.

The rest of the paper is organized as follows. Section 2 describes the communication-bound processor that serves as the evaluation framework. We propose techniques that can exploit a heterogeneous interconnect in Section 3 and evaluate them in Section 4. Section 5 discusses related work and we conclude in Section 6.

2. The Base Clustered Processor

Most proposals for billion-transistor processors employ a partitioned architecture [9, 15, 17, 18, 19, 20]. The allocation of instructions to computational units can be performed either statically [9, 12, 15, 17, 18, 23] or dynamically [5, 19, 20]. All of these designs experience a large number of data transfers across the chip. For the purpose of this study, we focus on one example implementation of a partitioned architecture – a dynamically scheduled general-purpose clustered processor. The solutions proposed in this paper are applicable to other architectures as well and are likely to be equally effective.

The clustered processor that serves as an evaluation platform in this study has been shown to work well for many classes of applications with little or no compiler enhancements [2, 5, 8, 11, 12, 26]. In this processor (shown in Figure 1), the front-end is centralized. During register renaming, instructions are assigned to one of 16 clusters. Each cluster has a small issue queue, physical register file, and a limited number of functional units with a single cycle bypass network among them. If an instruction's source operands are in a different cluster, copy instructions are inserted in the producing clusters and the values are copied into physical registers in the consuming cluster. To minimize communication cost and load imbalance, we implement instruction steering heuristics that represent the state-of-the-art. These heuristics incorporate information on dependence chains, critical operands, load, physical location, etc., and have been extensively covered in other papers [5, 11, 24].

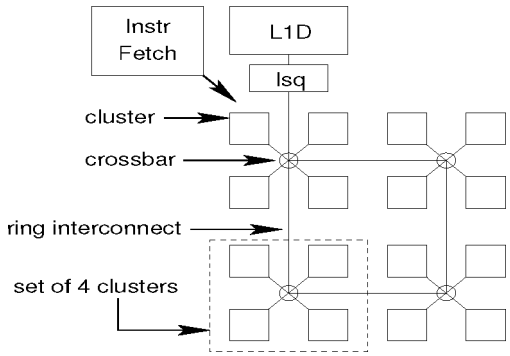

Figure 1. The 16-cluster system with four sets of four clusters each and a centralized LSQ and data cache. A crossbar interconnect is used for communication within a set of clusters and a ring connects the four crossbar routers.

The load/store queue (LSQ) and L1 data cache are centralized structures. Effective addresses for loads and stores are computed in one of the clusters and then sent to the centralized LSQ. The LSQ checks for memory dependences before issuing loads to the data cache and returning data back to the requesting cluster. While distributed LSQ and cache implementations have been proposed [13, 26], we employ a centralized LSQ and cache because a distributed implementation entails significant complexity and offers only modest performance improvements [4, 13].

Aggarwal and Franklin [3] point out that a crossbar has better performance when connecting a small number of clusters, while a ring interconnect performs better when the number of clusters is increased. To take advantage of both characteristics, they propose a hierarchical interconnect (Figure 1), where a crossbar connects four clusters and a ring connects multiple sets of four clusters. This allows low-latency communication between nearby clusters. Each link on the interconnect has a throughput of two transfers per cycle to deal with

the high traffic that is inevitable in such a partitioned architecture.

3. Proposed Techniques

3.1 Motivation

A major bottleneck limiting the performance of any partitioned architecture is the communication of data between producer and consumer instructions in different processing units. An ideal design would employ a high speed wire that consumed as little power as possible. Magen *et al.* [16] show that 90% of the dynamic power consumed in the interconnect is in 10% of the wires. The global wires in the partitioned architecture, such as the crossbar and inter-crossbar wires that connect the different clusters are likely to be major contributors to interconnect power. Data communciation on the interconnect can be categorised as follows: register values, store data, load data, and effective addresses for loads and stores. Register value transfers account for 54% of all inter-cluster communication, while the transfer of data for store instructions accounts for around 5%, data produced by load instructions accounts for 15%, and the transfer of effective addresses accounts for 24% of all communication. In the subsequent subsections, we show that a number of transfers in each of these categories is latency insensitive.

3.2 Power-Performance Trade-Offs in Interconnect Design

The premise behind the paper is that there exists a trade-off between performance and power consumption in the design of on-chip interconnects. An interconnect that is optimized for low power is likely to have longer delays and this has a debilitating effect on program performance (quantified in Section 4). Therefore, we propose the design of a heterogeneous interconnect where half of the wires are optimized for low delay and the other half are optimized for low power. As part of a preliminary evaluation, our focus has been the identification of latency insensitive interconnect transfers and a quantification of the performance impact of effecting them on low power, long latency wires. A detailed characterization of the power and performance of different interconnect implementations remains future work. Here, we qualitatively discuss one of the techniques that can be employed to reduce interconnect power, although at a performance cost.

Banerjee [7] developed a methodology to calculate the repeater size and interconnect length that minimizes the total interconnect power dissipation for any given delay penalty. The premise behind their approach is also based on the fact that not all global interconnects are on the critical path and hence, a delay penalty can be tolerated on these non-critical power efficient interconnects. As process technologies scale beyond 130nm, leakage power contributes significantly to the interconnect power and hence their technique provides greater power savings. For a delay penalty of 20%, the optimal interconnect saves 50% and 70% power in 100 and 50 nm technologies. Clearly, with technology scaling, leakage power dissipation becomes the dominating component of the total power dissipation and the use of small repeaters can help lower the total interconnect power.

3.3 Identifying Critical Register Operands

To identify latency insensitive communications, the first group of data transfers that we target are register values that get bypassed to consuming instructions as soon as they are generated by producing instructions.

Bypass of Register Values

In general, each instruction executed by a processor requires one or more input register operands for its execution. The instructions producing the operands wake up the consumer as soon as they generate the operand values. A consumer instruction cannot issue until all of the input operands have arrived. The operand that is produced last is more critical and needs to arrive at the consumer as early as possible, while the other operands can arrive as late as the last operand. In some cases, even if all the operands are ready, the instruction can wait in the issue queue if there is heavy contention for the functional units. A table that can classify data based on their arrival time and usage can be used to send them either through a fast critical network or a slow non-critical network.

We begin with a high-level description of how criticality information is gathered. Each instruction in the issue queue keeps track of the time difference between the arrival of an input operand and its actual execution.

If the time difference is significant, the transfer of the input operand is considered latency insensitive. When the instruction completes, along with the completion signal to the reorder buffer (ROB), a couple of bits are transmitted, indicating the criticality nature of each input operand. The ROB is augmented to keep track of a few PC bits for the producer of each input operand. These bits are used to index into a *Criticality Predictor* that is a simple array of saturating counters, similar to a branch predictor. When an instruction is dispatched to a cluster, its criticality prediction is also sent to the cluster. When the instruction completes, its result is transferred to consumers on the low latency or low power network depending on the criticality prediction. Thus, some non-trivial hardware overhead has been introduced in the centralized front-end. This overhead may be acceptable in an architecture where the power consumed by on-chip communications far outweighs the power consumed in the criticality predictor, rename, and ROB stages. This implementation serves as an example design point to evaluate the number of non-critical transfers. Other simpler implementations of criticality predictors are possible – for example, instructions following low-confidence branches. As future work, we plan to evaluate the behavior of other criticality predictors.

Transfer of Ready Register Operands

The above discussion targets register values that have to be urgently bypassed to dependent instructions as soon as they are produced. There exists another class of register input operands that are already ready when an instruction is dispatched by the front-end. Since it might take many cycles for the instruction to be dispatched to a cluster and for it to begin execution, the transfer of its ready register operands to the consuming cluster is often latency insensitive. We observed that all such transfers can be effected on the slow power-efficient network with a minimal impact on performance. Such an implementation is especially favorable as it does not entail additional hardware overhead for prediction mechanisms.

3.4 Communications for Cache Access

Cache Access in the Base Case

The earlier subsection describes the identification of register values that are not urgently consumed by other instructions. Register values represent a subset of the traffic observed on the inter-cluster network. All loads and stores compute their effective addresses in the clusters and forward them to the centralized LSQ. A load in the LSQ waits until its memory dependences are resolved before accessing the L1 data cache and forwarding the result back to the requesting cluster. For store instructions, the data to be stored in the cache is computed in one of the clusters and forwarded to the centralized LSQ. The LSQ forwards this data to any dependent loads and eventually writes it to the data cache when the store commits. Thus, in addition to the transfer of 64-bit register values, the inter-cluster network is also responsible for the transfer of load effective addresses, store effective addresses, load data, and store data.

Non-Critical Load Data

The data produced by a load instruction may be critical or not, depending on how quickly the consuming instructions issue. The criticality predictor described in the previous subsection can identify load instructions that produce critical data. Thus, it is possible to employ the techniques described earlier to send load data on either the critical or non-critical network. In order to not further complicate the design of the criticality predictor, for the purposes of this study, we assume that all load data is sent to the requesting cluster via the fast critical network.

Non-Critical Store Data

Store data, on the other hand, is often non-critical. The late arrival of store data at the LSQ may delay execution in the following ways: (i) dependent loads have to wait longer, (ii) the commit process may be stalled if the store is at the head of the reorder buffer. As our results in the next section show, both of these events are infrequent and store data can be indiscriminately sent on a slower, power-efficient network. In our benchmark set, about 5% of all interconnect traffic can be attributed to store data, providing ample opportunity for power savings.

Non-Critical Load and Store Effective Addresses

Load and store effective address transfers are usually on the critical path – store addresses are urgently required to resolve memory dependences and load ad-

dresses are required to initiate cache access. For many programs, accurate address prediction and memory dependence speculation can accelerate cache access [4]. For loads, at the time of instruction dispatch, the effective address and memory dependences can be predicted. This allows the cache access to be initiated without waiting for the clusters to produce load and store effective addresses. As soon as the cache is accessed, data is returned to the cluster that houses the load instruction. When the cluster computes the effective address, it is sent to the centralized LSQ to verify that the address prediction was correct. The effective address and memory dependence predictors can be tuned to only make high-confidence predictions, causing the mispredict rate to be much lower than 1% for most programs [4]. With such an implementation, the transfer of the effective address from the cluster to the centralized LSQ is no longer on the critical path – delaying the verification of address predictions does not significantly degrade the instruction execution rate.

3.5 Summary

Thus, we observe that there are two primary sources of delay-insensitive transfers in a communication-bound processor. (i) *Data that is not immediately consumed by dependent instructions*: Of these, register results produced by instructions, including loads, can be identified as non-critical by our criticality predictor. Store data is always considered non-critical. Input register operands that have already been generated when the consumer instruction is dispatched, are also always considered non-critical. (ii) *Data transfers that verify predictions*: If load and store effective addresses are accurately predicted at the centralized LSQ, the transfer of these addresses from the clusters to the LSQ happens only for verification purposes and does not lie on the critical path for any instruction's execution.

Section 4 identifies the performance impact of sending delay-insensitive transfers on a power-efficient and slower network.

4. Results

4.1 Methodology

Our simulator is based on Simplescalar-3.0 [10] for the Alpha AXP ISA. Separate issue queues and physical register files are modeled for each cluster. Contention on the interconnects and for memory hierarchy resources (ports, banks, buffers, etc.) are modeled in detail. To model a wire-delay-constrained processor, each of the 16 clusters is assumed to have 30 physical registers (int and fp, each), 15 issue queue entries (int and fp, each), and one functional unit of each kind. While we do not model a trace cache, we fetch instructions from up to two basic blocks in a cycle. Important simulation parameters are listed in Table 1.

Fetch queue size	64
Branch predictor	comb. of bimodal and 2-level
Bimodal predictor size	2048
Level 1 predictor	1024 entries, history 10
Level 2 predictor	4096 entries
BTB size	2048 sets, 2-way
Branch mispredict penalty	at least 12 cycles
Fetch width	8 (across up to 2 basic blocks)
Dispatch and commit width	16
Issue queue size	15 per cluster (int and fp, each)
Register file size	30 per cluster (int and fp, each)
Re-order Buffer size	480
Integer ALUs/mult-div	1/1 (in each cluster)
FP ALUs/mult-div	1/1 (in each cluster)
L1 I-cache	32KB 2-way
L1 D-cache	32KB 2-way set-associative, 6 cycles, 4-way word-interleaved
L2 unified cache	2MB 8-way, 25 cycles
I and D TLB	128 entries, 8KB page size
Memory latency	160 cycles for the first chunk

Table 1. Simplescalar simulator parameters.

The latencies on the interconnects would depend greatly on the technology, processor layout, and available metal area. The estimation of some of these parameters is beyond the scope of this study. For the base case, we make the following reasonable assumptions: it takes a cycle to send data to the crossbar router, a cycle to receive data from the crossbar router, and four cycles to send data between crossbar routers. Thus, the two most distant clusters on the chip are separated by 10 cycles. Considering that Agarwal *et al.* [1] project 30-cycle worst-case on-chip latencies at 0.035μ technology, we expect this choice of latencies to be representative of wire-limited future microprocessors[1]. We assume that each communication link is fully pipelined, allowing the initiation of a new transfer every cycle. Our results also show the effect of a network that has communication latencies that are a factor of two higher. By modeling such a communication-bound processor, we clearly isolate

[1] It must be noted that the L2 would account for a large fraction of chip area.

the performance differences between the different simulated cases. Similar result trends were also observed when assuming latencies that were lower by a factor of two. As Agarwal *et al.* predict [1], IPCs observed are low due to a very high clock rate and presence of wire delay penalties.

We use 21 of the 26 SPEC-2k programs as a benchmark set [2]. The programs were fast-forwarded for two billion instructions, simulated in detail for a million instructions to warm up various structures, and then measured over the next 100 million instructions. The reference input set was used for all programs.

To understand the inherent advantages of the criticality predictor, the simulations were performed without any address prediction. This is because address prediction adds to the benefits of the existing methodology and the combined results would not reflect the use of the criticality predictor clearly. At the end of the section, potential benefits of the criticality predictor with address prediction are discussed. For all simulations, three different interconnect configurations have been used. In the base (*high-performance*) case, the interconnects between the clusters are optimized for speed and allow two transfers every cycle. To model a processor that has interconnects optimized for power, we simulate a model (*low-power*) like the base case, but with wire latencies twice as much as those in the *high-performance* case. The difference in the average IPC between these two cases was found to be 21%, underlining the impropriety of an approach that focuses solely on low power. The *criticality-based* case assumes a heterogeneous interconnect with a combination of both the power optimized and the performance optimized wires. In every cycle,we can start one transfer each on the performance-optimized link and on the power-optimized link. The latency of the performance-optimized link is like that of the *high-performance* case, while the latency of the power-optimized link is like that of the *low-power* case. In the *criticality-based* case, all store data goes through the power optimized wires while all load data utilizes the fast wires. The ready operands, operands which are available for the consuming instruction at the time of dispatch, are all sent through the power optimized wires. The criticality predictor is used to choose between the fast and the slow interconnect for bypassed register values.

[2] *Sixtrack*, *Facerec*, and *Perlbmk* were not compatible with our simulation infrastructure, while *Ammp* and *Mcf* were too memory-bound to be affected by processor optimizations.

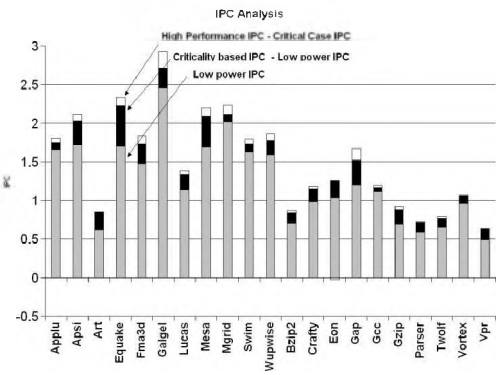

Figure 2. Graph showing performance of the *criticality-based* approach, compared to the *high-performance* and *low-power* cases.

4.2 Effects on Performance

To understand the performance of the criticality predictor, we refer to Figures 2 and 3. In Figure 2, the grey bars depict IPC for the *low-power* case, the black bars depict the improvements in IPC obtained by moving to the *criticality-based* case, and the white bars show the difference in IPC between the *criticality-based* and *high-performance* cases. The graph clearly shows that increasing the delay for every single transfer has a much larger impact on performance than increasing the delay for selected transfers based on criticality. Figure 3 shows the IPC gap between the *high-performance* and *criticality-based* cases and the corresponding percentage of transfers that get sent on the low-power interconnect. Note that these models assume no address prediction techniques, requiring that all effective addresses be sent on the low-latency interconnect.

The difference in the average IPC between the *high-performance* and *low-power* cases is 21%. For the *criticality-based* case, the criticality predictor is employed only for bypassed register transfers. The percentage of bypassed register transfers that get sent on the low-power network is roughly 29%. In addition, all store data and ready register operands get sent on the low-power network. From Figure 3, We see that about 36.5% of all transfers can happen on the low-

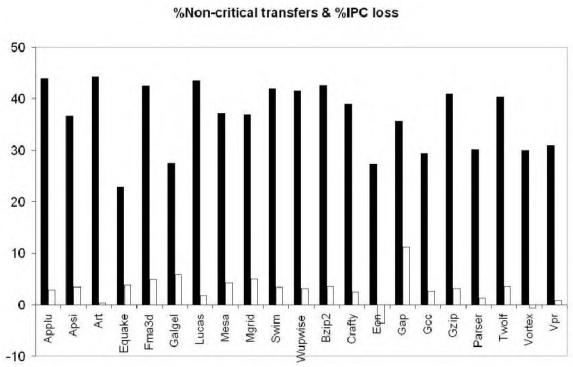

Figure 3. Plot showing the percentage of all transfers that happen on the low-power interconnect (not including predicted effective addresses) and the corresponding performance loss.

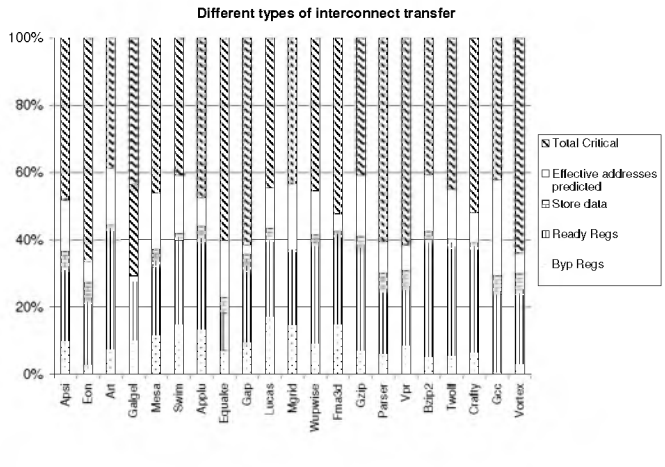

Figure 4. Graph showing different kinds of non-critical transfers as a fraction of total transfers

power network. The overall loss in IPC is only 2.5%, showing that we have identified a very favorable subset of transfers that are relatively latency insensitive.

In *eon* and *vortex*, performance increases slightly compared to the base case – the ability of the *criticality-based* interconnect to accomodate more data reduces contention in the links. *Gap* is the only program where the unpredictable nature of the code results in a 11% IPC loss because of inaccuracies in the criticality predictor.

This study highlights the importance of considering the design of wires with different levels of performance and power in different parts of the processor. Figure 2 shows us that ready register transfers tend not to be on the critical path and hence can use the slower and power optimized wires. It also tells us that the design of bypasses requires careful performance and power considerations. The use of a criticality predictor helps us steer data to preserve performance while reducing power.

Studies [4] have shown that around 52% of effective addresses have high confidence predictions. Transferring high confidence address predictions on the *criticality-based* interconnect can potentially achieve greater power savings. Figure 4 depicts the different kinds of transfers that happen on the interconnect as a fraction of the total transfers. We see that high confidence predictions that go through the *criticality-based* links account for 12.5% of the total transfers. This is in addition to the already existing non-critical register and store data transfers. Overall, we have 49% of the traffic (including address prediction values) through the power optimized links.

In future communication-bound processors, we expect more than 50% of chip power to be consumed within the interconnects. Assuming that our approach targets roughly 80% of on-chip interconnects and that the low-power wires consume half as much power as the high-power wires, overall chip power should reduce by about 10% with our optimizations.

5. Related Work

Tune *et al.* [24] and Srinivasan *et al.* [22] developed several techniques for dynamically predicting the criticality of instructions. Seng *et al.* [21] used the QOLD heuristic suggested by Tune to find critical instructions and redirect them to different units optimized for power and performance. The QOLD heuristic predicts an instruction to be critical if it reaches the top of the issue queue, *i.e.* the instruction has been waiting for a long time for its operands. They used slow low power units for executing the non-critical instructions to limit the performance impact. A recent study by Balasubramonian *et al.* [6] evaluates the design of heterogeneous cache banks and the use of criticality to assign instructions and data to each bank.

Magen [16] show that around 50% of a proces-

sor's dynamic power is dissipated in the interconnect alone. They characterize interconnect power in a current microprocessor designed for power efficiency and propose power aware routing algorithms to minimize power in the interconnect. They show that tuning interconnects for power optimizations could yield good results.

Several microarchitectural power simulators have been proposed to date. Recently, Wang *et al.* [25] proposed an interconnection power-performance simulator to study on-chip interconnects in future processors.

6. Conclusions

The power consumed by on-chip interconnects is already a major contributor to total chip power [16]. Future billion transistor architectures are likely to expend significantly more power transferring values between the different computational units. Various techniques, such as fewer repeaters, low-capacitance drivers, low voltage swings, etc., can be employed to reduce the power within the interconnects, but at the cost of longer wire delays.

Our results show that a large fraction of on-chip communications are latency tolerant. This makes the case for a heterogeneous interconnect, where some of the wires are optimized for high speed and the others for low power. Assuming such a heterogeneous interconnect is possible, we evaluate its potential to limit performance degradation, while effecting a majority of transfers on the power-efficient network. Our results show that latency tolerant non-critical transfers can be classified as follows: (i) register values that are not urgently read by consuming instructions (accounting for 32.3% of all interconnect transfers), (ii) store data that is often not forwarded to subsequent loads (accounting for 4.1% of all transfers), (iii) predicted load and store effective addresses that are being transferred only for verification purposes (12.8% of all transfers). As a result, roughly 49% of all inter-cluster communications can be off-loaded to power-efficient wires while incurring a performance loss of only 2.5%. If the interconnect is entirely composed of power-efficient long-latency wires, the performance degradation is as high as 21%. Thus, a heterogeneous interconnect allows us to strike a better balance between overall processor performance and power.

This paper serves as a preliminary evaluation of the potential of a heterogeneous interconnect. A more detailed analysis of the power-performance trade-offs in interconnect design will help us better quantify the performance and power effect of such an approach. The techniques proposed here can be further extended to apply to other processor structures, such as the ALUs, register files, issue queues, etc.

References

[1] 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.

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

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

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

[5] R. Balasubramonian, 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.

[6] R. Balasubramonian, V. Srinivasan, and S. Dwarkadas. Hot-and-Cold: Using Criticality in the Design of Energy-Efficient Caches. In *Workshop on Power-Aware Computer Systems, in conjunction with MICRO-36*, December 2003.

[7] 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.

[8] 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.

[9] 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.

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

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

[12] 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.

[13] E. Gibert, J. Sanchez, and A. Gonzalez. Effective Instruction Scheduling Techniques for an Interleaved Cache Clustered VLIW Processor. In *Proceedings of MICRO-35*, pages 123–133, November 2002.

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

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

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

[17] 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.

[18] 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.

[19] S. Palacharla, N. Jouppi, and J. Smith. Complexity-Effective Superscalar Processors. In *Proceedings of ISCA-24*, pages 206–218, June 1997.

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

[21] J. S. Seng, E. S. Tune, , and D. M. Tullsen. Reducing Power with Dynamic Critical Path Information. In *Proceedings of the 34th International Symposium on Microarchitecture*, December 2001.

[22] S. Srinivasan, R. Ju, A. Lebeck, and C. Wilkerson. Locality vs. Criticality. In *Proceedings of ISCA-28*, pages 132–143, July 2001.

[23] J. Steffan and T. Mowry. The Potential for Using Thread Level Data-Speculation to Facilitate Automatic Parallelization. In *Proceedings of HPCA-4*, pages 2–13, February 1998.

[24] 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.

[25] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A Power-Performance Simulator for Interconnection NEtworks. In *Proceedings of the 35th International Symposium on Microarchitecture*, November 2002.

[26] V. Zyuban and P. Kogge. Inherently Lower-Power High-Performance Superscalar Architectures. *IEEE Transactions on Computers*, March 2001.