Hierarchical Interconnects for On-chip Clustering

Aneesh Aggarwal
ECE Department
University of Maryland
College Park, MD 20742
aneesh@eng.umd.edu

Manoj Franklin
ECE Department and UMIACS
University of Maryland
College Park, MD 20742
manoj@eng.umd.edu

Abstract

In the sub-micron technology era, wire delays are becoming much more important than gate delays, making it particularly attractive to go for clustered designs. A common form of clustering adopted in processors is to replace the centralized instruction scheduler with multiple smaller schedulers that work in parallel within a single chip. Studies have found that existing interconnects connecting on-chip clusters, as well as proposed instruction distribution algorithms, are not scalable. The objective of this paper is to investigate alternate interconnects (we investigate hierarchical interconnects) that provide scalable performance with increase in on-chip clusters. We also investigate distribution algorithms that are best suited for these interconnects. Experimental results of these new interconnects with appropriate distribution techniques show that they more scalable than the existing techniques. achieve an IPC that is around 15-20% more than the most scalable existing configuration, and is also within 2% of that achieved by a hypothetical ideal processor having a 1-cycle latency crossbar interconnect, irrespective of the number of clusters; confirming their utility and applicability In this paper, we also discuss the many other design advantages that are obtained by the use of hierarchical interconnects.

1 Introduction

A major implication of going for sub-micron technology is that *wire delays* become more important than *gate delays* [14]. This effect will be predominant in global wires because their length depends on the die size, which is steadily increasing. A natural way to deal with the wire delay problem is to use the concept of *clustering*; build the processor as a collection of independent *clusters*, such that there are only a few *global wires* with very little communication through them. Fast localized communication can be done using short wires. Clustered processors typically execute groups of instructions independently (using decentralized hardware resources).

In the recent past, several decentralization proposals and evaluations have appeared in the literature [1] [2] [3] [4] [8] [9] [11] [12]. Commercial implementations are available in MIPS R10000 [13] and Alpha 21264 [7]. With continued increases in the number of on-chip transistors, we can integrate more on-chip clusters can be integrated, so as to exploit more parallelism. Some of the studies in the literature [1] specifically addressed this issue of performance scalability of distribution algorithms on existing interconnects for large number of on-chip clusters. These studies found that existing configurations are not scalable, as the number of clusters is increased.

In this paper, we investigate scalable on-chip cluster interconnects. In particular we investigate two different hierarchical interconnects. Although many interconnects have been studied in the context of multi-chip parallel processors [5], interconnects for on-chip clustering have not been studied in detail. The important issues while designing the two types of interconnects (on-chip and off-chip) are very different[1]. The hierarchical interconnects divide the clusters into groups. The clusters within a group are internally connected using crossbars. The groups, on the other hand, are connected together using either a single ring interconnect or a multiple rings interconnect. We also propose instruction distribution algorithms that take advantage of these hierarchical interconnects.

The rest of this paper is organized as follows. Section 2 highlights the important issues for performance scaling.

[1] Off-chip interconnects are used when different processors/chips need to communicate with each other. The communication latencies of off-chip interconnects are very high as compared to the clock speed of the processor, and minor variations in latency do not have much of an impact. Hence, their primary objectives are fault-tolerance, consistency of data, etc. On-chip interconnects, on the other hand, are used within a single processor, where communication is frequent and communication latencies are almost equal to the processor speed, and even a small variation in the latency affects the overall performance significantly. These reasons make communication latency one of the most important factors in determining the on-chip interconnects.

In Section 3, we investigate hierarchical interconnection topologies and suitable distribution algorithms. Section 4 presents the experimental methodology and evaluation of these new schemes. Section 5 discusses other advantages obtained from hierarchical interconnects. Finally, Section 6 concludes the paper with the major findings.

2 On-Chip Clustering

In this paper, a single-chip clustered processor is a collection of several clusters, as shown in Figure 1. Each cluster has a dynamic scheduler (DS), and several functional units (FUs). Instructions from multiple clusters are issued independently of each other, subject only to the availability of operand values. An interconnection network (ICN) connects the clusters together for supporting inter-cluster communication. In order to fetch a large number of instructions every cycle, the fetch mechanism predicts multiple branch outcomes at a time, and fetches a *trace* of instructions consisting of multiple basic blocks. Executed instructions are committed from the clusters in program order.

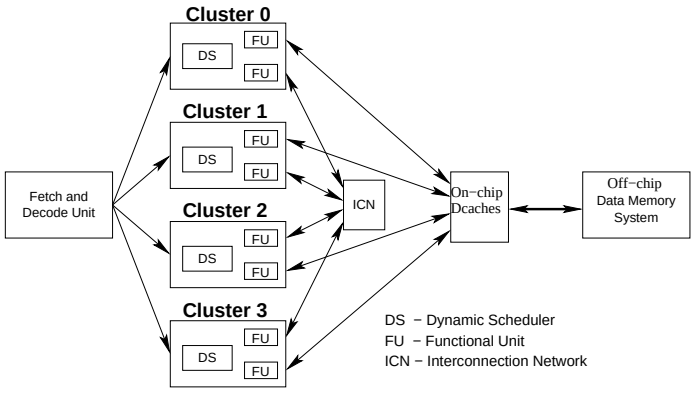

Figure 1. A Generic Single-Chip 4-Cluster Processor

2.1 Criteria for Performance

For good performance of clustered processors, two important criteria need to be considered: minimization of inter-cluster communication, and maximization of load balancing among the clusters. The former criterion attempts to reduce the number of cycles that instructions wait for operands, and the latter attempts to reduce the time instructions wait for an issue slot or functional unit. These two criteria are somewhat conflicting in nature. Good performance is obtained only when both criteria are satisfied. Minimum inter-cluster communication can be achieved only if data-dependent instructions are placed in the same cluster. To achieve maximum load balancing, along with minimum communication, data-independent instructions are placed in different clusters. In general, as the number of on-chip clusters is scaled up, the relative importance of inter-cluster communication becomes more and more important [1].

2.2 Existing On-chip Interconnects

A major factor affecting inter-cluster communication latency is the type of interconnect (proposed interconnects are bus, crossbar, and ring (uni-directional and bi-directional)) used to connect the clusters. The bus is a simple, fully connected network that permits only one data transfer at any time, providing a bandwidth of only O(1); a poor choice for large number of clusters.

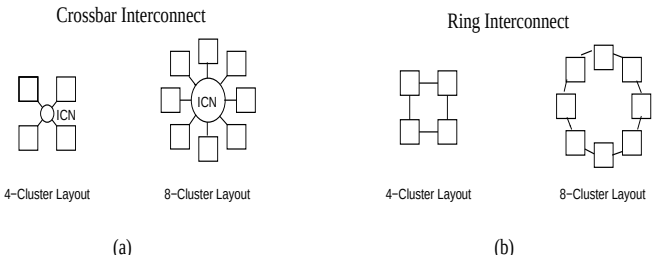

Figure 2. Single-Chip 4-Cluster and 8-Cluster Layouts having Minimum Worst-case Inter-Cluster Communication Delay: (a) Crossbar Interconnect; (b) Bi-directional Ring

A crossbar interconnect provides full connectivity from every cluster to every other cluster (O(N) bandwidth, where N is the number of clusters). With crossbar, the communication latency is the same irrespective of the clusters participating in the communication. So, as the number of clusters increases, the physical distance between the clusters increase, leading to an increase in the latency for all inter-cluster communication. Figure 2a shows this effect.

With a ring-type interconnect, the clusters are connected as a circular loop. The ring can be easily laid out with O(N) space using only short wires. A ring is ideal if most of the inter-cluster communication can be localized to neighboring clusters, but is a poor choice if a lot of communication happens across physically distant clusters. Again, as the number of clusters increase, even though the communication latency between neighboring clusters remains unchanged, the latency for communication between distant clusters increases. As shown in Figure 2b, the maximum latency increases from 2 cycles to 4 cycles, when the number of clusters is increased from 4 to 8.

3 Hierarchical Interconnects and Distribution Algorithms

From the studies done in [1], one of the major reasons for the lack of scalability is the increase in inter-cluster communication delays with increase in the number of clusters;

communication delays cannot be decreased arbitrarily. It was also shown in [1] that a crossbar interconnect is generally better for few clusters, whereas a ring interconnect for larger number of clusters. This is because, for ring interconnects, efforts can be made to localize communication to between neighboring clusters. But, distant cluster communication still hurts the scalability of the ring interconnects. To take advantage of both types of interconnects (ring and crossbar), we investigate hierarchical interconnects for on-chip clustering.

3.1 Hierarchical Interconnects

The basic idea behind hierarchical interconnects is that a small number of physically close clusters are interconnected using a low-latency crossbar, and the distant clusters are connected using a ring. When using such an interconnect along with an appropriate instruction distribution algorithm, most of the inter-cluster communication happens within the low-latency crossbars only. High communication latency may be incurred occasionally when communicating between distant clusters. The next subsection discusses these interconnects and the best distribution algorithms for these interconnects. We investigate two hierarchical interconnects : a single ring of crossbars and multiple rings of crossbars interconnect.

3.1.1 Single Ring of Crossbars

Figure 3 shows the layout of a ring of crossbars interconnect for 8 clusters and 12 clusters; 4 clusters form a group. The groups are connected by a single bi-directional ring. Any value that is to be communicated within a group is communicated using the crossbar interconnect, and incurs a communication latency of 1 cycle. Communication across groups uses the ring structure; the latency is 2 cycles for neighboring groups and an additional cycle for each hop. For example, in Figure 3, for 12 clusters, the communication of a value from a cluster in group 1 to another cluster in group 1 requires 1 cycle, but communication from a cluster in group 1 to a cluster in group 2 or 3 requires 2 cycles.

3.1.2 Multiple Rings of Crossbars

The layout for a multiple rings of crossbars interconnect is shown in Figure 4[2], where the groups are inter-connected using multiple bi-directional rings. Each of the 4 clusters within the 4-cluster group is connected to a corresponding cluster in its two neighboring cluster groups. For example, the bottom left cluster of group 2 is connected to the bottom left clusters of groups 1 and 3. The communication latency within the group is 1 cycle, whereas, the communication latency across the 4-cluster groups depends on the clusters

[2]Wires drawn in the figure are much longer than needed to improve the readability of the figure.

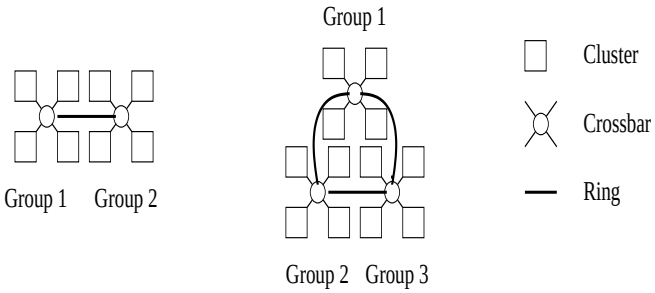

Figure 3. Layout of a Ring of Crossbars Interconnect for a 8-Cluster Processor and a 12-Cluster Processor

participating in the communication. If the communicating clusters are connected together, then the latency is 1 cycle; otherwise, a cycle gets added to latency for an additional hop across the 4-cluster group. For example, the communication between the bottom left cluster of groups 1 and 2 requires 1 cycle, whereas between the bottom left cluster of group 1 and the bottom right cluster of group 2 requires 2 cycles.

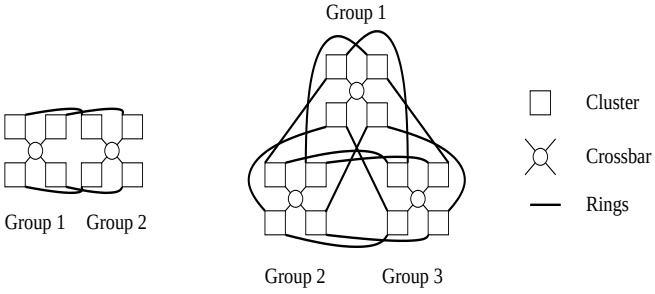

Figure 4. Layout of Multiple Rings of Crossbars Interconnect for 8 Clusters and 12 Clusters

3.2 Instruction Distribution

In this subsection, we discuss the instruction distribution algorithms for the hierarchical interconnects. Even though we investigated several algorithms, we present only the best performing ones in this section.

Ring of Crossbars : The results reported in [1] show that with 4 clusters, the best performance is generally obtained with the *ldst slice* algorithm (cf. Appendix), with the crossbar interconnect, whereas, *first-fit* algorithm (cf. Appendix) is generally the best with the ring interconnect connecting large number of clusters. Using this information, we investigated the following instruction distribution approach.

While distributing the instructions among the clusters,

optimize the distribution within a single cluster group using *ldst slice* algorithm, then consider the next group of clusters, and so on. Thus, the distribution algorithm used to distribute traces across cluster groups, is *first-fit*. The 2-level distribution approach thus attempts to get the benefits of two different algorithms, in the situation where each performs the best. During our experimental evaluation too, we found this hierarchical instruction distribution algorithm to be performing better than all other algorithms.

Rings of Crossbars : For the rings of crossbars interconnect, we use the *ldst slice* algorithm (explained in the appendix). We also performed experiments using the *2-level* distribution algorithm discussed in the previous paragraph along with other distribution algorithms. The reason that *ldst slice* performs better than the *2-level* for the multiple rings of crossbars interconnect is the presence of additional connectivity between cluster groups. This means that most of the communication is restricted to 1 cycle latency. The 2-level algorithm avoids communication within a chunk of instructions, while paying some communication costs across the chunks. Ldst slice on the other hand, tries to avoid any form of communication. If it is not able to achieve that, most of the times, it pays a communication cost of 1 cycle (most of the communication is restricted to 1 cycle in this interconnect), Hence performs better than *2-level*.

4 Experimental Results

In order to verify the potential of the hierarchical interconnect along with the appropriate distribution algorithms, we conducted a set of experiments. The results were compared with the most scalable configuration among the existing configurations, and a possible hypothetical ideal processor. We first discuss the experimental setup for our experiments and then give the results.

4.1 Experimental Setup

Our experimental setup consists of an execution-driven simulator based on the MIPS-I ISA. The simulator does cycle-by-cycle simulation, including execution along mispredicted paths. The hardware features and default parameters are given in Table 1. In addition, an 8-way, 128 Kbyte trace cache [10], with 1 cycle access time, and a block size of 16 instructions, is used.

The programs are compiled for a MIPS R3000-Ultrix platform with a MIPS C (Version 3.0) compiler using the optimization flags distributed in the SPEC benchmark makefiles. While reporting the results, the execution time is expressed in terms of instructions per cycle (IPC) without NOPs. We also measure the instruction stalls due to inter-cluster register traffic and due to load imbalance, so as to get more insight.

4.2 Performance Results

The results (IPC) obtained with the hierarchical configurations (cf. Section 3) are presented in Figure 5. In figure 5, we compare the results obtained with hierarchical interconnects against the *first-fit* algorithm on a bi-directional ring (has the best scalability). We also simulate a hypothetical clustered processor with an ideal 1-cycle crossbar. This hypothetical ideal processor uses the *ldst slice* algorithm for instruction distribution, and is included to model one of the best possible scenarios with a clustered processor. In Figure 5, each benchmark has 2 sets of 4 bars each; for 8-clustered and 12-clustered processors. Each bar in the graph gives the improvement in IPC over the 4-clustered crossbar processor using *ldst slice* algorithm along the Y-axis. A crossbar interconnect using *ldst slice* is used as the base case because of its best performance among all the 4-cluster configurations [1].

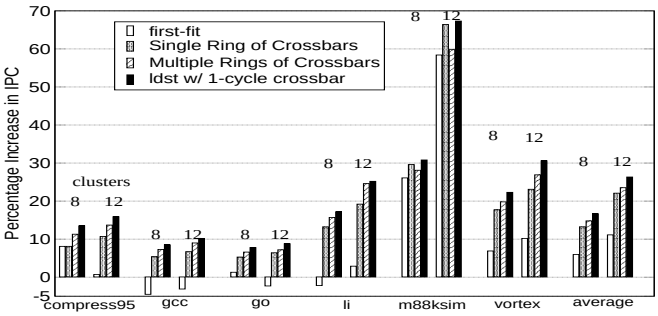

Figure 5. Percentage IPC Increase for 8 clusters and 12 clusters, comparing Hierarchical Interconnect configurations with other configurations

From the results, it can be seen that the most scalable of the existing configurations for clustered processors (*first-fit* on a bi-directional ring interconnect) is not scalable. In fact for `compress95` and `go`, the performance of *first-fit* decreases on going from 8 clusters to 12 clusters. The negative bars in the graph indicate that the performance with a 8-cluster/12-cluster processor using the *first-fit* algorithm on a ring is worse than a 4-cluster processor using the *ldst slice* algorithm on a crossbar. This is seen for benchmarks `gcc` and `li`. Also, the increase in IPC for this configuration on going from 4 clusters to 8 clusters is only around 5%, on an average, and around 12%, on an average, on going from 4 to 12 clusters.

The hierarchical approach, on the other hand, provides better scalability. For 8 clusters, there is a 0-16% increase in IPC over *first-fit* algorithm, when using the single ring of crossbar interconnect with hierarchical instruction distribution algorithm (cf. Section 3). For 12 clusters, the in-

Default Values for Processor Parameters		Default Values for Cluster Parameters	
Parameter	Value	*Parameter*	Value
Fetch/Commit Size	16 instructions (1 trace)	*Cluster window size*	16 instructions
Control flow predictor	2-level trace, 1024 entry	*Functional unit latencies*	1 cycle Int; 10 cycle Mul/Div
L1 - I-cache	2 cycle, 4-way, 16KB	*Cluster issue width*	2 instructions/cycle
L1 - D-cache	2 cycle, 4-way, 64KB	*Ring inter-cluster delay*	1 cycle
L2 - Unified cache	10 cycle latency, Infinite,	*Crossbar inter-cluster delay*	$\lceil \log_2 \frac{\#clusters}{2} \rceil$ cycles

Table 1. Default Parameters for the Experimental Evaluation

crease in IPC is between 8-15%. The corresponding values for the multiple rings of crossbars interconnect are 2-18% for 8 clusters and 2-22% for 12 clusters. Multiple rings of crossbars performs better than the single ring of crossbars by around 4%, except for `m88ksim`. This peculiar behavior of `m88ksim` is not because of the interconnect, but because of the distribution algorithm used. Because of `m88ksim`'s high prediction accuracy, the instruction windows of all the clusters are almost full. This means that the next trace to be distributed gets spread when all the clusters are considered while distribution (as is the case for multiple rings of crossbars interconnect and the hypothetical ideal processor). This spreading does not affect the single ring of crossbars interconnect since the spreading takes place only within the small group of clusters, which are anyway connected by a 1-cycle crossbar. *Note that the increase in performance is due to low latency communication and not due to increase in bandwidth when the connectivity is increased.* From the graphs, it can also be seen that the performance of our hierarchical approaches is very close to the performance of the hypothetical ideal 1-cycle crossbar processor; for a 12 cluster processor, the difference is only about 2%, with a maximum of about 5%.

4.3 Analysis

Next, we analyze the results through the stereoscope of inter-cluster communication and load balancing. The analysis is done for one of the hierarchical approaches (single ring of crossbars with 2-level instruction distribution), along with a monolithic bi-directional ring using *first-fit* algorithm and the ideal hypothetical 1-cycle crossbar using the *ldst slice* algorithm. Figure 6 presents statistics on the average number of instructions stalled in a cycle due to inter-cluster communication (cf. left part of figure) and due to load imbalance (cf. right part of figure). For the average number of instructions stalled due to inter-cluster communication, the instructions waiting for operands already produced in another cluster, but still in transit, were counted. For the average number of instructions stalled due to load imbalance, those instructions are counted that are ready to execute but are waiting for an issue slot. These instructions are counted only in the event of a free issue slot available in another cluster.

The format of this figure is as follows. For each benchmark, there is a set of 3 bars for each cluster configuration: representing the 3 cases— *first-fit*, *hierarchical approach* (single ring of crossbars), and *ldst with 1-cycle crossbar*. As the number of clusters is increased, the average number of instructions stalled due to load imbalance and inter-cluster communication increase. This explains the tapering performance results obtained as the number of clusters is increased. *First-fit* almost always has slightly fewer number of instructions stalled due to inter-cluster communication, but suffers from a lot of load imbalance as compared to *hierarchical approach* and *ldst with 1-cycle crossbar*, and hence performs the worst among the three. On the other hand, the average number of instructions stalled due to inter-cluster communication and load imbalance are almost the same for *hierarchical approach* and *ldst with 1-cycle crossbar*, with the *hierarchical approach* having slightly more stalled instructions. This leads to the slightly worse performance of the *hierarchical approach* as compared to *ldst with 1-cycle crossbar*.

5 Advantages of the Hierarchical Approach

The hierarchical approach also has many other advantages when the design issues of a processor are considered. In this section, we discuss some of the advantages of the hierarchical approach.

5.1 Resource Distribution

A clustered processor still has many centralized resources such as the branch prediction tables, data caches, etc. The access latency of these centralized resources increases with increase in on-chip clusters (increased distances). The advantage of the hierarchical approach lies in the distribution of these centralized resources among the clusters. With an hierarchical interconnect, each cluster group can share a centralized resource within itself. For a non-hierarchical approach, on the other hand, each cluster gets a separate resource during resource distribution. This could lead to over-distribution of resources, leading to performance impacts. For example, if a separate data cache is attached to each cluster in a 12-cluster ring interconnect processor, worst case cache access latency increases by 6

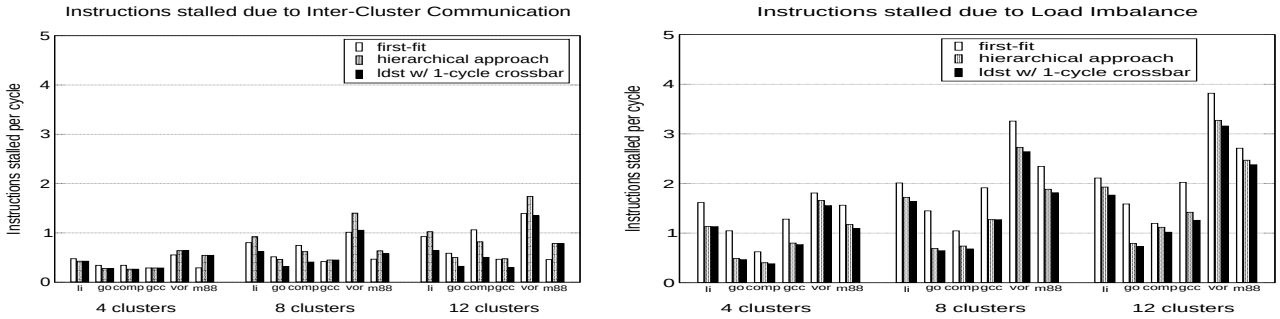

Figure 6. Average Number of Instructions Stalled per Cycle due to Inter-Cluster Communication and due to Load Imbalance

cycles. For a 12-cluster hierarchical interconnect with a separate cache for each 4-cluster group, access of any remote cache would require only 1 extra cycle. The distribution of the cache for a 8-cluster hierarchical single ring of crossbars interconnect is shown in Figure 7.

In this section, we focus on performance impact of data cache distribution. All the experiments in the previous sections were performed using a centralized cache with a constant access latency. Here, we distribute the cache among the different clusters for the single ring of crossbars configuration, and measure its performance against a centralized cache having higher access latency. The results for the multiple rings of crossbars are expected to be very similar.

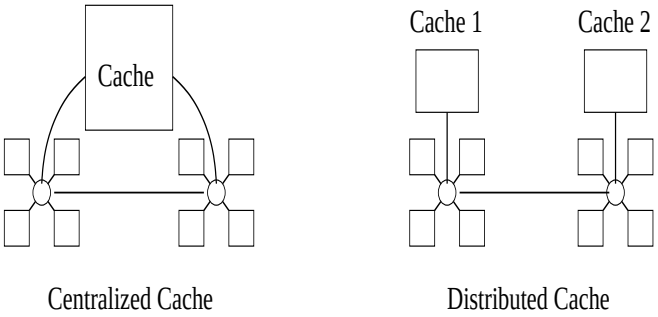

Figure 7. Increase in IPC for a distributed cache as compared to a centralized cache with increased access latency

For a centralized cache, the cache access latency is 3 cycles for 8 clusters, and 4 cycles for 12 clusters. For the distributed cache system, each 4-cluster group has a centralized cache, which is called the *local cache*, with a 2-cycle access latency. The accesses to the cache local to other groups (*remote cache*) requires an additional cycle. The distribution of the cache is a set-based distribution, where each distributed cache structure has the same number of sets of cache lines, and the adjacent addresses in a cache line map to the same cache. *Note that no changes were made to the 2-level instruction distribution algorithm.* One possible algorithm could be to take the cache distribution into consideration: assign loads to clusters local to the cache having the address. We found that doing this scattered the dependent instructions leading to a somewhat worse performance as compared to the original algorithm.

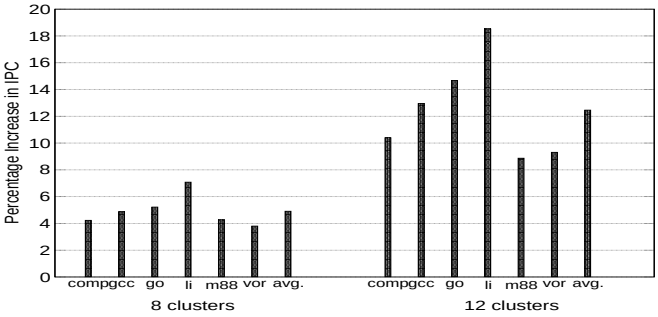

Figure 8. Increase in IPC for a distributed cache as compared to a centralized cache with increased access latency

Figure 8 gives the IPC results. As can be seen in the figure, a distributed cache is performing on an average 5% better than a centralized cache for 8 clusters, and on an average 12-13% better for 12 clusters. The maximum improvement is obtained for `li`: 7% for 8 clusters and 18.5% for 12 clusters. For a centralized cache, all the cache accesses incur the increased access latency, whereas for a distributed cache, only the accesses to the remote cache incur the high latency, hence the improvement.

5.2 Ease of Instruction Distribution

In all of the instruction distribution schemes discussed in the paper, instruction distribution is done using hardware

logic and lies in the critical path of program execution. In case of a non-hierarchical approach, this logic needs to consider the state of all the clusters while distributing the instructions. However, in hierarchical instruction distribution, only a small number of clusters are considered during distribution. This simplifies the hardware in the distribution logic (hardware complexity is proportional to the number of clusters considered for distribution) and makes the distribution fast.

We performed experiments comparing a single ring of crossbars hierarchical interconnect using the *2-level* hierarchical distribution algorithm (cf. Section 3) with the hypothetical ideal processor using the *ldst slice* algorithm. The time taken for instruction distribution for the hypothetical processor is increased by 1 cycle when going from 4 clusters to 8 clusters and by 2 cycles when going from 4 clusters to 12 clusters. For the hierarchical distribution, no change is made to the time taken for distribution, as the number of clusters input to the algorithm does not change. Figure 9 gives the results of the experiments.

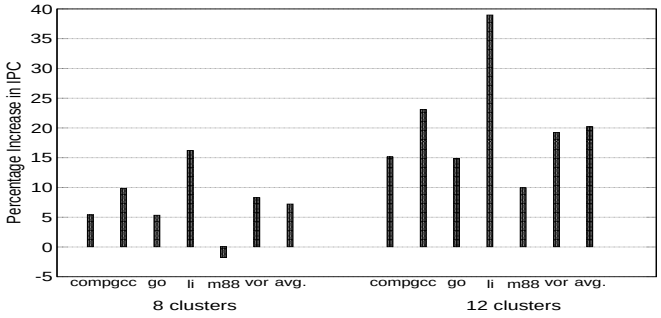

Figure 9. Increase in IPC for hierarchical instruction distribution on a single ring of crossbars as compared to the 1-cycle latency crossbar hypothetical ideal processor with increased time taken for instruction distribution

In Figure 9, the Y-axis gives the percentage of the increase in IPC for the hierarchical approach over the hypothetical ideal processor taking more cycles for instruction distribution. Earlier, we saw in Figure 5 that the hypothetical processor on an average performed 2-3% better than the single ring of crossbars interconnect. In Figure 9, it can seen that the hierarchical approach performs better (on an average 7% for 8 clusters and 20% for 12 clusters) than the hypothetical processor with increased distribution time.

An exception is the performance of `m88ksim` for 8 clusters; the hypothetical processor still performs better than the hierarchical approach. The reason again is the same as that explained in Section 4.2. Because of the high branch prediction accuracy of `m88ksim`, the clusters are almost full and the new instructions to be distributed get scattered. But if another cycle is taken for instruction distribution, more space becomes available (some instructions commit) for distribution, giving better distribution. For 12 clusters, however, the overhead of increased instruction distribution time out weighs the benefits obtained due to better instruction placement.

Similarly, for static instruction distribution [6] [12], the software (used for instruction distribution) for the hierarchical approach is expected to be much simpler than for a non-hierarchical approach (e.g a simpler compiler).

5.3 Adaptability to Multithreaded Environments

Another advantage of hierarchical approach is while running multi-threaded applications. In the hierarchical approach, there is a distinct demarcation between groups of clusters (with each group having its own local resources), a single thread or multiple threads could be assigned to a single group of clusters. Whereas, for non-hierarchical approach, the demarcation exists at cluster level. If a single thread or multiple threads are assigned to one cluster, then clusters go waste when there are not enough threads. To avoid the wastage of clusters, some complex method of assigning threads to clusters needs to be designed. For example, consider the case of a 8-cluster processor connected by a non-hierarchical interconnect. If there are just 2 or 3 threads to be executed, the 8 clusters need to be assigned to these threads (maybe using some complex method). But, for a hierarchical approach, there are only 2 groups , each of 4 clusters. So, even for 2 threads, each thread can be assigned to one group. In such assignments, for hierarchical interconnects, the communication between the clusters also is very simple, being localized within the group, and is not global as in the case of non-hierarchical interconnects.

6 Conclusions

In single-chip clustered processors, the common goal is to decrease hardware complexity, increase clock rate, and maintain high levels of parallelism by distributing the dynamic instruction stream among several on-chip clusters. The small size of the cluster windows allows a higher clock rate, while the combined issue rates of several clusters still allow large amounts of parallelism to be exploited. In the long run, as more transistors are integrated into a processor chip, the number of clusters will increase, necessitating the development of scalable inter-cluster interconnects and appropriate instruction distribution algorithms.

Most of the current interconnects for on-chip clustering along with the existing algorithms for instruction distribution do not scale well. To improve the scalability of clustered processors, we proposed two hierarchical interconnects and investigated techniques of distributing the instruction stream to take advantage of these new interconnects.

Using this new distribution approach, we achieve performance almost equal (around 2% less) to that obtained with a hypothetical 1-cycle latency crossbar (for a large number of clusters). These hierarchical interconnects achieve IPCs around 15-20% better than the most scalable existing configuration of clustered processors.

We also discussed some of the many other advantages of hierarchical interconnects in the design issues of on-chip clustered processors. In particular, we looked at its advantages for distributing resources (considered data caches in this paper) among the clusters, for achieving less complex and fast instruction distribution hardware/software, and finally for its ease of adaptability to run multi-threaded applications. We found that distributing the data cache among clusters connected with a hierarchical interconnect achieves an improvement of 12-13% on an average over increased access latency centralized cache. Also, simpler distribution hardware (because of hierarchical interconnects) achieves a relative performance improvement of almost 25% on an average, over the hypothetical 1-cycle latency crossbar processor.

Acknowledgements

This work was supported by the U.S. National Science Foundation (NSF) through a CAREER grant (MIP 9702569) and a regular grant (CCR 0073582).

References

[1] A. Aggarwal and M. Franklin, "An Empirical Study of the Scalability Aspects of Instruction Distribution Algorithms for Clustered Processors," *Proc. International Symp. on Performance Analysis of Systems and Software (ISPASS '01)*, 2001.

[2] A. Baniasadi and A. Moshovos, "Instruction Distribution Heuristics for Quad-Cluster, Dynamically-Scheduled, Superscalar Processors," *Proc. International Symp. on Microarchitecture (MICRO-33)*, 2000.

[3] R. Canal, J. M. Parcerisa and A. Gonzalez, "Dynamic Cluster Assignment Mechanisms," *Proc. Int. Symp. on High-Performance Computer Architecture (HPCA-6)*, 2000.

[4] R. Canal, J. M. Parcerisa and A. Gonzalez, "Dynamic Code Partitioning for Clustered Architectures," *International Journal of Parallel Programming*, 2000.

[5] K. Hwang, "Advanced Computer Architecture," *McGraw-Hill*, 1992.

[6] K. Kailas, K. Ebcioglu, and A. Agrawala, "CARS: A New Code Generation Framework for Clustered ILP Processors," *Proc. 7th International Symposium on High Performance Computer Architecture (HPCA-7)*, pp.133-143, 2001.

[7] D. Leibholz and R. Razdan, "The Alpha 21264: A 500 MHz Out-of-Order Execution Microprocessor," *Proc. Compcon*, pp. 28-36, 1997.

[8] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-Effective Superscalar Processors," *Proc. 24th Annual International Symposium on Computer Architecture*, pp. 206-218, 1997.

[9] N. Rangananathan and M. Franklin, "An Empirical Study of Decentralized ILP Execution Models," *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, 1998.

[10] E. Rotenberg, S. Bennett, and J. E. Smith, "Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching," *Proc. 29th International Symposium on Microarchitecture*, 1996.

[11] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. Smith, "Trace Processors," *Proc. 30th International Symposium on Microarchitecture*, 1997.

[12] S. S. Sastry, S. Palacharla, and J. E. Smith, "Exploiting Idle Floating-Point Resources For Integer Execution," *Proc. Intl. Conf. on Programming Lang. Design and Implementation*, 1998.

[13] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, pp. 28-40, April 1996.

[14] The National Technology Roadmap for Semiconductors, Semiconductor Industry Association, 1999.

Appendix A

Here, we briefly explain the already proposed instruction distribution algorithms used in this paper.

First-Fit: In this method [2], instructions are assigned to the same cluster until the cluster window fills up, and then assigned to the next cluster, and so on. The side effect of this scheme is to reduce inter-cluster communication. As register results produced are more likely to be used very soon, the dependent instructions either end up in the same cluster or neighboring clusters, and is very suitable for ring interconnects. The algorithm suffers in case of a crossbar interconnect, because independent instructions in a trace also end up in the same cluster leaving less space for dependent instructions.

Load-Store Slice (LdSt Slice): In this method, all the instructions on which loads or stores are dependent on are sent to the same cluster to minimize the inter-cluster delays. A static partitioning *ldst slice* scheme was proposed in [12]. A dynamic version of this scheme was proposed in [3]. This scheme aspires to reduce inter-cluster register communication (especially for *loads* and *stores*). It also gives some importance to load balancing by assigning the remaining instructions according to the load situation of the clusters. One side effect of this scheme is that the value produced by a load might be used by an instruction assigned to a far away cluster. It is best for crossbar interconnects, since all the communications have the same latencies, but not so good for ring interconnects where dependent instructions might end up in far away clusters.