Evaluation of Bus Based Interconnect Mechanisms in Clustered VLIW Architectures

Anup Gangwar
Calypto Design Systems (I) Pvt. Ltd.,
LogixPark, A4 Sector-16,
NOIDA, India-201301
E-mail: *agangwar@calypto.com*

M. Balakrishnan, Preeti R. Panda & Anshul Kumar
Dept. of Computer Science & Engineering,
Indian Institute of Technology Delhi,
Hauz Khas, New Delhi-110016, India
E-mail: {*mbala,panda,anshul*}*@cse.iitd.ernet.in*

Abstract

With new sophisticated compiler technology, it is possible to schedule distant instructions efficiently. As a consequence, the amount of exploitable instruction level parallelism (ILP) in applications has gone up considerably. However, monolithic register file VLIW architectures present scalability problems due to a centralized register file which is far slower than the functional units (FU). Clustered VLIW architectures, with a subset of FUs connected to any RF are the solution to this scalability problem.

Recent studies with a wide variety of inter-cluster interconnection mechanisms have presented substantial gains in performance (number of cycles) over the most studied RF-to-RF type interconnections. However, these studies have compared only one or two design points in the RF-to-RF interconnects design space. In this paper, we extend the previous reported work. We consider both multi-cycle and pipelined buses. To obtain realistic bus latencies, we synthesized the various architectures and found out post layout clock periods. The results demonstrate that while there is very little variation in interconnect area, all the bus based architectures are heavily performance constrained. Also, neither multi-cycle or pipelined buses nor increasing the number of buses itself is able to achieve performance comparable to point-to-point type interconnects.

1 Introduction

Media applications have large amount of instruction level parallelism (ILP) which justifies building very high issue-rate processors [5, 17, 18]. While, the exact amount of achievable ILP is still a point of debate, published work on future directions for architecture research points in the direction of very high issue-rate processors [6, 14]. Whereas SuperScalar processors are more popular in the general application domain, VLIW processors have gained wide acceptance in the embedded systems domain e.g. TriMedia, TiC6x, Equator's MAP-CA and MAP-1000 [9], HP-ST Microelectronic's Lx [3] etc.

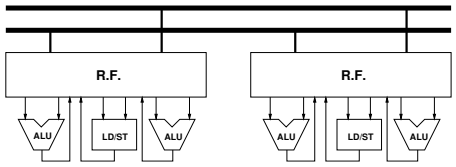
VLIW architectures with a monolithic register file with all the FUs connected to it lead to an increase in the register file (RF) delay, power and area. The centralized RF thus becomes a bottleneck in performance on all these metrics [15]. The RF delay, power and area are more sensitive to the increase in number of ports rather than the RF size (number of registers) itself. Clustering the VLIW architecture into a number of small sets of FUs, wherein each such set has its own RF is a natural solution to such a scalability problem. However, the inter-cluster interconnect in such architectures plays a crucial role in processor performance [19] and [8].

Till recently only one type of inter-cluster cluster interconnect, a simple bus connected to all the clusters, was used and researched in both industry and academia. However, two published results on evaluation of various inter-cluster interconnect mechanisms in clustered VLIW processors [19] and [8] reported substantial performance gains in number of cycles when various other mechanisms are used. However, both these studies used few specific design points for evaluating bus based interconnects. In [19], a single cycle transfer bus was connected to all the clusters and in [8] two single cycle transfer buses were connected to all.

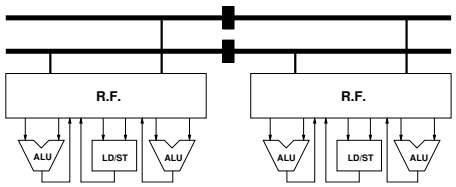
In this paper we extend the previous reported work on performance of bus based inter-cluster interconnects in clustered VLIW processors. We start by showing that the clock-period is heavily constrained beyond two cluster architectures if single cycle transfer buses are used. Towards this end, we model the various clustered architectures in VHDL and perform synthesis, place and route using industry standard tools. We next use these results to arrive at realistic latencies for multi-cycle and pipelined buses such that the clock-period is not affected. Finally we carry out exhaustive performance evaluation of these bus-based architectures using the obtained latencies.

2 Inter-cluster Interconnect Architectures

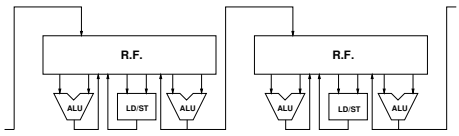
Figure 1 shows the various interconnect architectures. Figures 1(a) and 1(b) show two alternatives for RF-to-RF architectures. In Figure 1(a) buses either run at processor frequencies, in which case the overall operation frequency of the processor comes down or they run at frequencies slower than the actual processor clock (multi-cycle). In Figure 1(b), buses are explicitly pipelined so that the processor operation frequency is not affected i.e. the inter-cluster in-



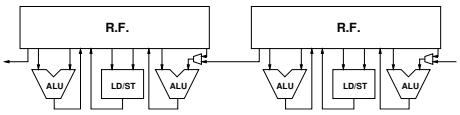
(a) RF-to-RF



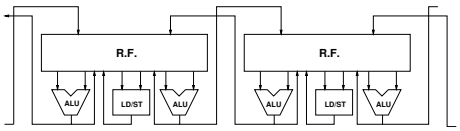
(b) Pipelined RF-to-RF



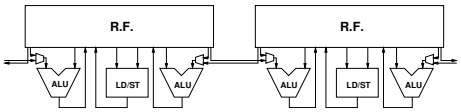
(c) Write Across-1



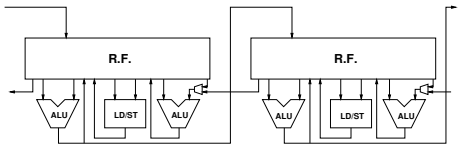
(d) Read Across-1



(e) Write Across-2



(f) Read Across-2



(g) Write/Read Across-2

Figure 1. Some Inter-cluster Interconnect Architectures

terconnect path does not become the critical path. Figures 1(c), 1(d), 1(e), 1(f) and 1(g) show what have been termed as the direct communication architectures [8]. In these architectures, FUs from one cluster can directly communicate with the RF of another cluster.

3 Clock Period and Chip Interconnect Area for Clustered Architectures

To obtain realistic bus latencies and also to get a comprehensive performance estimate we evaluated the clock-period obtained for various interconnect and cluster configurations. Towards this end we modeled the processors using parameterized VHDL. Our modeled processor has a standard load store architecture with RISC style instructions. The ALU components: adder, multiplier and divider are obtained using the Synopsys DesignWare libraries. This ensures that the critical path is not constrained due to inefficient implementation of these modules. The load/store unit follows standard Static RAM (SRAM) timing, such as that presented in [7]. RF for any processor is custom designed [1, 20]. Since RFs cannot be synthesized efficiently by the common synthesizers, these are usually hand crafted. For lack of availability of such a handcrafted RF design, we included a dummy RF during synthesis. This RF doesn't contain the complicated muxing logic which a real RF has, however, it does ensure that proper port mappings are in place so that static timing analysis (STA) reports proper paths. The entire VHDL has been written at Register Transfer Level (RTL) to obtain most efficient implementation. Our parameterized VHDL model allows variation of: Interconnect architecture, Number of clusters, Register file size, Register file (and ALU) width, Data RAM size and Instruction RAM size. The flow used is industry standard Synopsys frontend (Design Compiler) and Cadence backend (SoC Encounter). We followed a hierarchical place and route approach in which clusters are first placed and routed and then processor floorplan is done. Pin positions on clusters are fixed manually to obtain good top-level floorplan.

We ran our experiments for two ASIC Technologies, UMC's 0.13μ six metal layer process and UMC's 0.18μ five metal layer process. The results are presented in Tables 1 and 2. Table 1 shows variation of clock period for different interconnect mechanisms with an increase in the number of clusters. Table 2, shows variation of interconnect area as a percentage of the total chip area for different interconnect mechanisms with an increase in number of clusters. Abbreviation *RF* has been used in this table for single cycle transfer RF-to-RF architecture, *WA* for *Write Across*, *RA* for *Read Across* and *WR* for *Write/Read Across*. Results of both clock period and interconnect area are similar for 0.18μ technology and hence have not been shown.

Looking at these results, it is clear that point-to-point type interconnects scale very well with increase in number of clusters. In these cases the critical path is not through

N_{Clust}	WA.1	RA.1	WA.2	RA.2	WR.2	RF
2	0.851	0.859	1.036	0.894	0.891	1.152
4	0.883	0.944	1.030	1.001	0.956	1.987
6	0.929	0.977	1.104	1.015	1.051	2.760
8	1.005	1.065	1.077	1.108	1.112	3.891
10	1.026	1.064	1.126	1.091	0.995	5.110

Table 1. Clock Period (ns) for UMC 0.13μ ASIC Tech.

N_{Clust}	WA.1	RA.1	WA.2	RA.2	WR.2	RF
2	39.45	39.34	39.42	39.16	39.33	43.43
4	36.71	36.65	36.63	36.53	36.65	36.67
6	35.69	35.66	35.73	35.61	35.55	35.71
8	34.90	34.62	35.02	34.82	34.84	34.98
10	34.89	34.84	34.95	34.73	34.81	34.89

Table 2. Interconnect Area (as % of Chip Area) for UMC 0.13μ ASIC Tech.

the inter-cluster interconnects. However, for RF-to-RF type interconnects beyond the two cluster configuration, the critical path does lie through inter-cluster communication bus and hence scalability is limited if the buses are run at processor frequency. We next use these clock-period numbers to arrive at realistic bus latencies for pipelined buses. These various bus configurations (pipelined and multi-cycle) are in turn used to perform an exhaustive design-space exploration.

4. Design Space Exploration

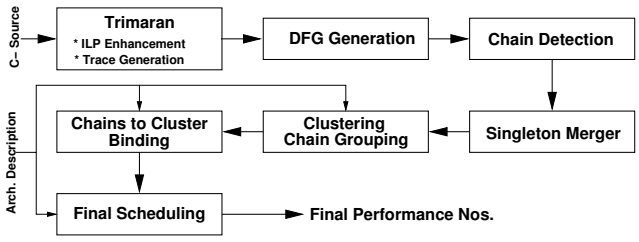


Figure 2. DSE Framework

Figure 2 shows the overall design space exploration methodology. The Trimaran system is used to obtain an instruction trace of the whole application from which a DFG is extracted for each of the functions. Trimaran also performs a number of ILP enhancing transformation. The chain detection phase finds out long sequences of operations in the generated DFG. The clustering phase, which comes next, forms groups of chains iteratively, till the number of groups is reduced to the number of clusters in the architecture. The binding phase, binds these groups of chains to the clusters. We agree that ideal results are obtained, when all the problems i.e. operation to cluster and FU assignment, register allocation and scheduling are done simultaneously [10, 13]. However, this makes the problem intractable for large graphs. We thus divide the problem as follows: First operation to cluster binding is done and next operation to FU in a cluster binding is done. Since, during clustering, the partial schedules are calculated (explained in detail later), the typical phase coupling problem [10, 13], is contained to a large extent, while still keeping the overall problem size manageable. Lastly, a scheduling phase schedules the operations into appropriate steps. More details on each of these phases is presented below.

DFG Generation: The DFG generation is carried out by matching source destination pair of register and memory usage. The machine model used for Trimaran is an extremely flexible one, with 256 GPR and 1024 Predicate registers. Trimaran also performs a number of ILP enhancing transformation, which in turn lead to an increase in the achievable ILP.

Chain Detection: The second phase is the chains detection phase. In this phase long sequences of operations are found in this DFG. The idea is to bind these chains to one cluster (similar to the one discussed in [11].

Singleton Merger: The chain detection phase returns a large number of chains. Including a significant number which have only one element (singleton). After observing the large number of singletons we specifically introduced a singleton merging phase. In this phase, the singletons which do not have any source or destination are distributed in $n_{clusters}$ number of chains (groups). The idea is that these would not constrain the scheduler in any way, as they donot have any source or destinations. Nodes, which have no destinations (sources) are merged with the shortest chain of their sources (destinations).

Clustering:

Algorithm 1 Clustering Algorithm

1: $resources \Leftarrow \bigcup_{All\ Clusters} Res.\ per\ cluster$
2: $do\_pure\_vliw\_scheduling(graph, resources)$
3: **while** $(no\_of\_chains(graph) > n\_clust)$ **do**
4: **for** $(i = 1$ to $no\_of\_chains(graph))$ **do**
5: **for** $(j = 0$ to $i)$ **do**
6: $dup\_graph \Leftarrow graph\_dup(graph)$
7: $dup\_chains \Leftarrow graph\_dup(chains)$
8: $merge\_chains(dup\_graph, du\_chains, i, j)$
9: $c_{i,j} \Leftarrow est\_sched(dup\_graph, dup\_chains)$
10: **end for**
11: **end for**
12: $SORT(C)$; Sorting priority function:
i) Increase in sched_length
ii) Larger communication edges, if sched_length is same
iii) Smaller chains, if sched_length and communication edges are equal
13: $n\_merge \Leftarrow MIN(0.1 * n_{chains}, n_{chains} - n_{clust})$
14: merge top n_merge chains from A
15: **end while**

The next phase is the clustering phase (Algorithm 1).

Here the number of chains is reduced to the number of clusters, by grouping selected chains together. The idea here is to reduce the inter-cluster communication between various groups. However, the closeness between these groups is architecture dependent. This can be better understood by examining architectures shown in Figures 1(a) and 1(b). While in the former, data transfer from any cluster to any other cluster takes same number of cycles, in the latter this varies. This stage assumes that finally the communication would happen through the best possible inter-cluster interconnects. Thus, this schedule length effectively represents an upper bound on the actual schedule.

The loop (steps 3 to 15) reduces the number of chains to $n_{clusters}$ by pairwise merger of chains. The chains are examined pairwise for their affinity (step 4 to 11) and a lower triangular matrix C is formed. Here each element c_{ij} gives the estimated schedule length due to merger of i and j (step 9). The C matrix is sorted according to multiple criteria (step 12). At each stage most beneficial n_{merge} pairs are selected and merged (steps 13 to 14).

The schedule estimation algorithm, which has not been shown due to paucity of space works as follows: Each of the nodes in the particular merged group of chains is scheduled taking into account data dependency. For scheduling we assume that each operation takes one cycle. The basic scheduling algorithm is list scheduling with distance from sink as the heuristic. If at any stage a node has any outgoing edge, then the node connected to that particular edge is marked dirty. However, the schedule of this particular node and all its children is not updated till it is needed. If at a later stage any node has an incoming edge from this node or its children, then the schedule of the dirty node along with the schedule of all its connected nodes is updated. This leads to a significant saving in computation.

Binding: The next step is to bind these groups of chains to clusters. Although the value of $n_{clusters}$ is quite small (not more than sixteen in our case), still the no. of possible bindings is quite large. This effectively rules out any exhaustive exploration of the design space. The following observation, established through experimentation, makes this stage extremely important: *while a good result propagation algorithm can affect the schedule length by around a factor of two, a poor binding at times can lead to schedules which are more than three to four times larger than the optimal ones.*

The binding heuristics are driven by the impact which communication latency of a particular node will have on final schedule. In effect we recognize that the data transfer edges from each of the merged group of chains to some other group are not equivalent. Some are more critical than the others in the sense that they would affect the schedule to a larger extent. The heuristics try to capture this, without explicit scheduling. We calculate the ASAP and ALAP schedules for each of the individual nodes. A first order estimate of this impact is given by the mobility of each individual node. Say, we have a communication edge from A_i to A_j, and ASAP and ALAP schedules for these nodes are a_i, a_j and l_i, l_j respectively. Then if $(a_j - l_i) \geq \delta$, where δ is the maximum communication distance between any two clusters, then this edge is not critical at all. As there is enough slack to absorb the effect of even the largest communication latency. On the other hand if, $(l_j = a_i + 1)$ the node has zero mobility and is thus most critical. We calculate the weight of each communication edge as follows:

$W_{i,j} = max\left(0, \delta - \left(\frac{a_j+l_j}{2} - \frac{a_i+l_i}{2}\right)\right)$

While weighting measure is able to segregate non-critical edges from critical ones, it is not able to distinguish clearly between edges whose nodes have equal mobility. To take this second effect into consideration, we also pull in the distance from sink, or path length for each of the source nodes. This path length when multiplied with Wi, j, gives us the final weight for each of the communication edges.

Algorithm 2 Binding Algorithm

```
1:  connect_graph ⇐ gen_connect_graph(graph, chains)
2:  while (Not all nodes in connect graph are bound) do
3:      source_node ⇐ find_highest_wt_edg(connect_graph)
4:      Bind both nodes of this edge to closest clusters
5:  end while
6:  while (Not all clusters have been considered) do
7:      prev_sched_len ⇐ sched_len(graph)
8:      Swap binding for two adjacent clusters
9:      sched_len ⇐ schedule_graph(graph)
10:     if (prev_sched_len < sched_len) then
11:         Swap back bindings for these clusters
12:     end if
13: end while
```

Algorithm 2, shows the binding algorithm. The algorithm works on a weighted connectivity graph, which is generated as discussed above. The initial part of the algorithm (step 2 to 5) is basically a greedy one. While it seems to work well for architectures which communicate only in one *direction*, the algorithm fails to take advantages for architectures with bidirectional connectivity (buses). Partially motivated by this and partially by [11], we thus bring in an additional iterative improvement phase, by performing a local search around this initial binding (step 6 to 13).

Scheduling: The scheduling phase is architecture specific. We have separate schedulers for multi-cycle RF-to-RF type architectures and pipelined bus architectures. Although a few scheduling algorithms for clustered architectures have been reported in literature, they all are specifically meant for single cycle RF-to-RF type of architectures [2, 11, 16]. Our scheduling algorithm again is a list scheduling algorithm with distance of the node from sink as the heuristic. What it additionally contains is steps to transfer data from

one cluster to other. The result propagation algorithm, tries to move data to clusters using the bus paths by scheduling transfer operations.

5 Experimental Results and Observations

We ran our experiments for a set of media applications. The main sources of benchmarks are MediaBench I [12], proposed MediaBench II [4] and DSP-Stone [21]. Results are shown in Figures 3, 4 and 5. In Figure 3, results are grouped under numbers of the form (xB, yL). Here x denotes the number of buses and y denotes the bus latency. From the figure it is clear that performance comparable to point to point connected architectures is achieved by $8,1$, $16,1$ and $16,2$. configuration if the cycle time is not considered. However, such a solution is very expensive due to the extremely high number of buses required. Also, as shown in Section 3, global buses with a single cycle latency lead to a decrease in the overall clock period by up to a factor of five leading to a drastic decrease in overall performance (Figure 5. Another interesting feature is that the performance varies asymptotically with x/y i.e. the average data transferred per cycle.

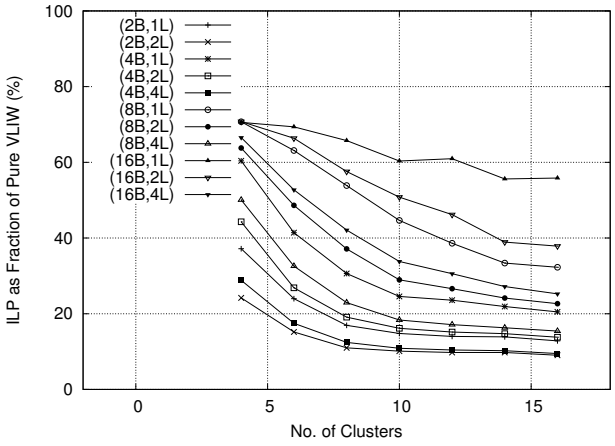


Figure 3. Average Variation in Performance for Multi-cycle Configurations

While running the buses at frequencies slower than the processor is one alternative, another one is to explicitly pipeline the buses taking clock-period into account. To simplify scheduling, we assume a homogeneous configuration of buses. Results for these configurations are shown in Figure 4. Here results are again grouped under various numbers of the form (xB, yS, zL). Where, x denotes the number of buses, y, bus stride and z latency of each path between two consecutive appearance of registers in bus. Bus stride is the number of clusters skipped after which a pipeline register is introduced. Whereas for a stride of two, the inter-cluster interconnect does not appear in the critical path for a stride of four, it does. Therefore, for this stride we have shown results only for a latency of two. Another consideration is the divisibility of $N_{clusters}$ by y. As a consequence we have only been able to show results for four, eight and sixteen cluster configurations.

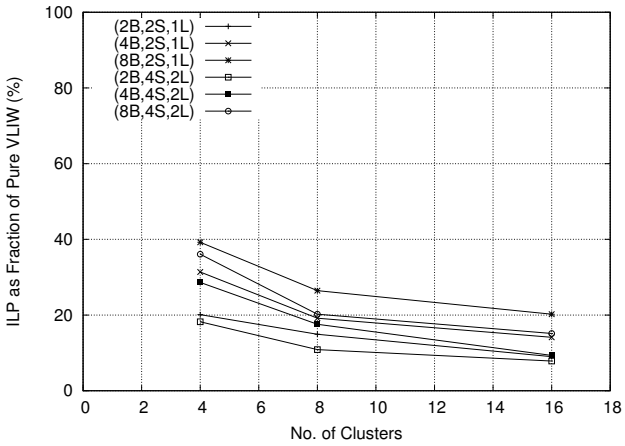


Figure 4. Average Variation in Performance for Pipelined Bus Configurations

From Figure 4 it is clear that the performance of pipelined buses is severely limited. To analyze this we need to consider the factors which limit performance of clustered architectures. These are required communication bandwidth and communication latency. Bus based architectures invariably suffer from increased latencies as compared to direct communication. This is due to the fact that an additional transfer (copy) instruction needs to be scheduled if data has to be moved between clusters. Whereas introducing registers in buses allows the simultaneous usage of various *links* in buses for local communication, this does come with a penalty of one additional cycle latency. Correlating with results of multi-cycle buses in Figure 3, it is evident that a large bandwidth is required for local transfers, however this is advantageous if the transfer does not incur any additional penalty. However, this is not the case with configurations which we have shown here as that is unrealistic (see Table 1).

Using the clock-period results it is possible to obtain a better picture of the final processor performance. Taking *Write Across - 1* architecture as the baseline case, we scale the reported ILPs using following formula: $ILP_{effective,\ x} = ILP_{original,\ x} * \frac{Clk\ Period_{WA.1}}{Clk\ Period_x}$. Here x is the architecture type under consideration, $ILP_{original,\ x}$, is the original ILP reported in [8], $Clk\ Period_{WA.1}$ is the clock-period of *Write Across-1* type of architecture and $Clk\ Period_x$ is the clock period of architecture under consideration.

Figure 5 shows effective average ILP. It is evident from this figure that global connectivity would annul an advantage which multiple buses may bring. The clock period scales too rapidly with global connectivity and no amount of increase in bandwidth can compensate for this effect. It is quite interesting to compare this figure with that obtained

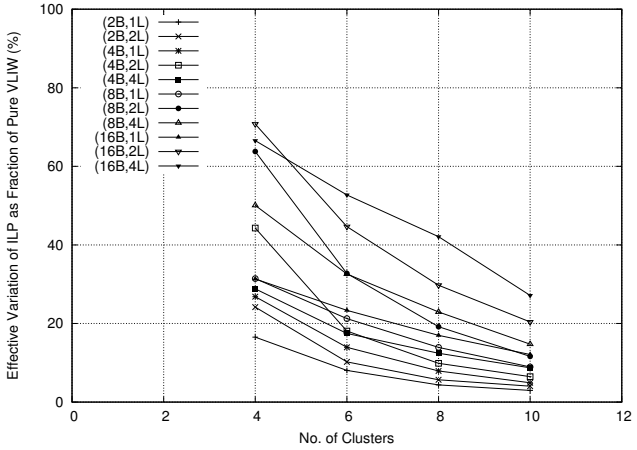


Figure 5. Effective Variation in Performance for UMC 0.13μ ASIC Technology and Multi-cycle Configurations

considering only number of cycles. However, it needs to be noted that there is considerable variation in performance if applications are considered individually.

6 Conclusions and Future Work

In this paper we have presented an exhaustive evaluation of bus based inter-cluster interconnects in clustered VLIW processor. We have synthesized the processors with a variety of inter-clutter interconnects and obtained the achievable clock-period values. These values have in turn been used to arrive at realistic bus transfer latencies. The poor performance of such bus based interconnect in sharp contrast to the direct-communication type of interconnects speaks heavily against their usage. These results become all the more significant if one takes into view the fact that almost all the research architectures and quite a few of the commercial ones have based their work on these architectures. Our current work in progress focuses on high-level estimations for direct-communication type architectures to customize them on a per application (domain) basis.

References

[1] J.-L. Cruz, A. Gonzalez, and M. Valero. Multiple-banked register file architectures. In *International Symposium on Computer Architecture (ISCA-2000)*, 2000.

[2] G. Desoli. Instruction Assignment for Clustered VLIW DSP Compilers: A New Approach. Technical Report HPL-98-13, Hewlett-Packard Laboratories, Feb. 1998.

[3] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. M. O. Homewood. Lx: A technology platform for customizable VLIW embedded processing. In *International Symposium on Computer Architecture (ISCA'2000)*. ACM Press, 2000.

[4] J. Fritts and B. Mangione-Smith. MediaBench II - Technology, Status, and Cooperation. In *Workshop on Media and Stream Processors, Istanbul, Turkey*, Nov. 2002.

[5] J. Fritts and W. Wolf. Evaluation of static and dynamic scheduling for media processors. In *2nd Workshop on Media Processors and DSPs in conjunction with 33rd Annual International Symposium on Microarchitecture*. ACM Press, 2000.

[6] J. Fritts, Z. Wu, and W. Wolf. Parallel Media Processors for the Billion-Transistor Era. In *International Conference on Parallel Processing*, pages 354–362, 1999.

[7] J. Gaisler. LEON: A Sparc V8 Compliant Soft Uniprocessor Core. http://www.gaisler.com/leon.html.

[8] A. Gangwar, M. Balakrishnan, and A. Kumar. Impact of Inter-cluster Communication Mechanisms on ILP in Clustered VLIW Architectures. In *2nd Workshop on Application Specific Processors (WASP-2), in conjunction with 36th Annual International Symposium on Microarchitecture*, Dec. 2003.

[9] P. Glaskowsky. MAP1000 unfolds at Equator. *Microprocessor Report*, 12(16), Dec. 1998.

[10] K. Kailas, K. Ebcioglu, and A. K. Agrawala. CARS: A new code generation framework for clustered ILP processors. In *HPCA*, pages 133–144, 2001.

[11] V. Lapinskii, M. F. Jacome, and G. de Veciana. High quality operation binding for clustered VLIW datapaths. In *IEEE/ACM Design Automation Conference (DAC'2001)*, June 2001.

[12] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.

[13] E. Ozer, S. Banerjia, and T. M. Conte. Unified assign and schedule: A new approach to scheduling for clustered register file microarchitectures. In *International Symposium on Microarchitecture*, pages 308–315, 1998.

[14] Y. N. Patt, S. J. Patel, M. Evers, D. H. Friendly, and J. Stark. One billion transistors one uniprocessor one chip. In *IEEE Computer*, 1997.

[15] S. Rixner, W. J. Dally, B. Khailany, P. R. Mattson, U. J. Kapasi, and J. D. Owens. Register organization for media processing. In *Proceedings of 6th International Symposium on High Performance Computer Architecture*, pages 375–386, May 2000.

[16] J. Sanchez and A. Gonzalez. Instruction scheduling for clustered VLIW architectures. In *International Symposium on System Synthesis (ISSS'2000)*, 2000.

[17] D. Stefanovic and M. Martonosi. Limits and graph structure of available instruction-level parallelism (research note). *Lecture Notes in Computer Science*, 1900:1018–1022, 2001.

[18] D. Talla, L. John, V. Lapinskii, and B. Evans. Evaluating Signal Processing and Multimedia Applications on SIMD, VLIW and Superscalar Architectures. In *International Conference on Computer Design (ICCD)*, pages 163–174, Sept. 2000.

[19] A. Terechko, E. L. Thenaff, M. Garg, J. van Eijndhoven, and H. Corporaal. Inter-Cluster Communication Models for Clustered VLIW Processors. In *9th International Symposium on High Performance Computer Architecture, Anaheim, California*, pages 298–309, Feb. 2003.

[20] J. H. Tseng and K. Asanovi. Banked multiported register files for high-frequency superscalar microprocessors. In *International Symposium on Computer Architecture (ISCA-2003)*, pages 62–71, June 2003.

[21] V. Zivojinovic, J. M. Velarde, C. Schlager, and H. Meyr. DSPStone – A DSP-oriented Benchmarking methodology. In *International Conference on Signal Processing Application Technology, Dallas, TX*, pages 715–720, Oct. 1994.