Impact of Inter-cluster Communication Mechanisms on ILP in Clustered VLIW Architectures

Anup Gangwar
anup@cse.iitd.ernet.in

M. Balakrishnan
mbala@cse.iitd.ernet.in

Anshul Kumar
anshul@cse.iitd.ernet.in

Department of Computer Science and Engineering
Indian Institute of Technology Delhi
New Delhi-110016, India

ABSTRACT

VLIW processors have started gaining acceptance in the embedded systems domain. However, monolithic register file VLIW processors with a large number of functional units are not viable. This is because of the need for a large number of ports to support FU requirements, which makes them expensive and extremely slow. A simple solution is to break up this register file into a number of small register files with a subset of FUs connected to it. These architectures are termed as clustered VLIW processors.

In this paper we first build a case for clustered VLIW processors with more than four clusters by showing that the available ILP in most of the media applications for a 16 ALU and 8 LD/ST VLIW processor is around 20. We then provide a classification of the inter-cluster interconnection design space, and show that a large part of this design-space is currently unexplored. Next, using our performance evaluation methodology, we evaluate a subset of this design space and show that the most commonly used type of interconnection, RF-to-RF, fails to meet achievable performance by a large factor, while certain other types of interconnections can lower this gap considerably. We also establish that this behavior is heavily application dependent. We also present results about the statistical behavior of these different architectures by varying the number of clusters in our framework from 4 to 16. These results clearly show the advantages of two of the architectures over others. Finally, based on our results, we give some proposals for a new interconnection network, which should lower this performance gap.

Categories and Subject Descriptors

C.1.1 [**Processor Architectures**]: VLIW architectures

Keywords

VLIW, ASIP, Clustered VLIW Processors

1. INTRODUCTION

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WASP-2 San Diego, California USA
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

Available Instruction Level Parallelism (ILP) in applications, specifically media applications, has been a point of debate for long. A few evaluation mechanisms and studies are presented in [2, 6, 7, 14]. The main problem with these studies is that most of them are carried from the compiler perspective, rather the design-space-exploration perspective. Towards, this end the achievable ILP and not the available ILP is measured. However, what a designer is basically interested in is the available ILP. These achievable studies use a compiler for extracting parallelism. This restricts the achieved ILP by the ability of the compiler to extract operation level concurrency. The simulation driven methodologies also get bogged down by the ability of caches to supply data to the FUs and as a consequence the achieved ILP reported by these studies is quite low. However, a simple look at the kernels of media applications shows that there is tremendous ILP present. One study which presents results for available ILP is [14]. However, they have reported results for SPEC95 benchmarks and not media applications. The problem with [22] is that they deal purely with application behavior totally disregarding architectural constraints.

A large amount of ILP naturally justifies large number of FUs for multiple-issue processors, which in turn due to the prohibitively expensive register file [18] justify clustering. Clustering is nothing new as a large number of architectures, both commercial as well as research, have had clustered architectures. These include Siroyan[20], TiC6x, Sun's MAJC, Equator's MAP-CA, TransMogrifier [15]. A large variety of inter-cluster interconnection mechanisms is seen in each of these processors. However, the tools which have been developed for these architectures are not retargetable, and are specific to the particular architecture developed. Also, what is missing is a concrete classification of interconnection design-space as well as a formal study of the impact of different interconnections on performance. This is the focus of our paper.

A first study of various inter-cluster communication mechanisms has been presented in [23]. However, they have considered only five different communication mechanisms and also the amount of ILP in their benchmarks is quite low (maximum is around 4). Our classification broadens their range and brings in a new domain of architectures into consideration. Since, they only have an ILP of four, they have not explored beyond four clusters. It was our hypothesis and which has been validated by the results, that restricting to four clusters would not bring out the effects of different interconnection mechanisms. Another limitation of [23] is that they have used compiler scheduling for result generation. Thus amount of detected parallelism is directly proportional to the size of the block

DSP-Stone Kernels		
S.No.	Kernel Name	Avg. ILP
1.	Matrix Init.	20.42
2.	IDCT	21.97
3.	Biquad N Sect.	21.64
4.	Lattice Transfor.	23.33
5.	Matrix Mult.	19.64
6.	Insert. Sort	18.94

MediaBench Benchmarks				
S.No.	Bench. Name	ILP		
		Evaluated	Others	Avg.
7.	JPEG Dec.	19.94	3.1	18.98
8.	JPEG Enc.	20.86	2.4	20.71
9.	MPEG2 Dec.	17.23	2.1	13.90
10.	MPEG2 Enc.	13.17	3.2	12.55
11.	G721 Dec.	16.95	2.3	15.06
12.	G721 Enc.	19.14	2.1	19.14

Table 1: Average ILP in Benchmarks (16-ALU, 8-MEM)

being formed by the VLIW compiler for scheduling. While they do work on a specialized block called Treegion [1], which is larger than the Hyper Blocks [16] or Super Blocks [8] typically formed by a VLIW processor, the extracted parallelism is still quite small. Based on these observations, we work directly with the data-flow of an application, as described in [14]. An instruction trace is obtained from a VLIW compiler, Trimaran [24], and then the data-flow graph (DFG) is generated from this. This DFG is scheduled and bound to the various FUs to obtain the final performance numbers. Using such a methodology allows us to bypass compiler limitations.

The rest of this paper is organized as follows; Section 2, gives the available ILP in benchmarks as well as discusses the techniques used to evaluate these benchmarks. Section 3, gives a classification for the design space of inter-cluster interconnection networks for clustered VLIW processors. Section 4, gives our design space exploration methodology and gives an overview of the various algorithms. Section 5, tabulates the results obtained for various benchmarks for different architectures and also discusses their implications. Finally Section 6, sums up our work as well as gives directions for further research.

2. ILP IN MEDIA APPLICATIONS

For the available ILP, we present the results for a set of media benchmarks. The main source of benchmarks is MediaBench I [13]. To examine relevance, we have focused only on those MediaBench I applications which are part of proposed MediaBench II [5]. A second smaller set of benchmarks was chosen from various sources, primarily, Trimaran [24] and DSP-Stone [25]. A profile was obtained for the entire application and a set of most time consuming functions was chosen. This set consumes more than 85% of the application execution time.

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. The machine model used for Trimaran is an extremely flexible one, with 256 GPR and 1024 Predicate registers to reduce the number of false register reuse dependences. Also, it has 32 Float, Integer, Memory and Branch Units each to remove any ILP losses due to insufficient resources. Trimaran also performs a number of ILP enhancing transformation, out of which the loop unrolling transformation is the most important one. This reduces false register dependences, introduced due to register reuse. The DFG generation phase picks up each instruction in sequence and searches backwards from that instruction. In the process it finds out the first occurrence of the source registers/address of this instruction in the destination field of a previous instruction. If the source register is found, a data-flow edge is introduced. However, if the addresses are found to match, then the communication has happened through memory and a false edge is introduced. This false edge doesn't denote data dependency, rather, it denotes a constraint for the scheduler that this instruction should not be scheduled before the instruction which stores this value in memory.

Finally, a resource constrained list scheduling, with distance-to-sink as the priority function, is done on these operations to obtain ILP numbers. The final results are shown in Table 1. Here the column *Evaluated*, shows the ILP obtained for the functions of this application which we have considered based on profile results to capture more than 85% of the application execution. The column *Others* shows the average ILP numbers taken from [6] for the remaining parts of the application. The *Avg.* column is a weighted sum of columns 3 and 4 with weights based on profiles. These results show, that the amount of achievable ILP in media applications is quite high.

3. INTERCONNECTION DESIGN SPACE FOR CLUSTERED VLIW PROCESSORS

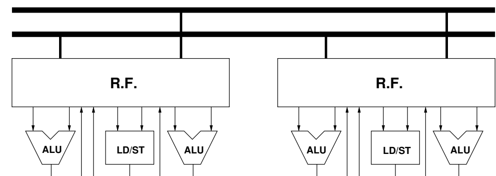
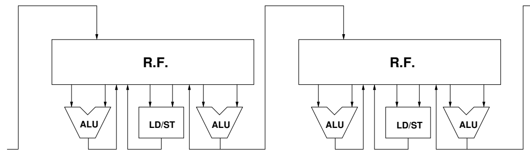


Figure 1: RF-to-RF $(\delta = 1)$

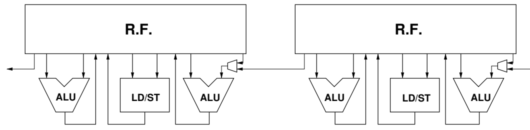
Clustered VLIW processors can be classified on the basis of their inter-cluster communication structures. At the top level we can divide them into two sub-categories: a) Those supporting inter-cluster RF-to-RF copy operations and b) Those supporting direct inter-cluster communication between FUs and RFs. There is very little variety in the architectures supporting RF-to-RF copy operations. At most these can be classified on the basis of the interconnect mechanism which they use for supporting these transfers. The examples of such architectures are: Lx [4], NOVA [9], Sanchez et. al. [19], IA-64 [21]. An example RF-to-RF architecture is shown in Figure 1.

We use the RF→FU (read) and FU→RF (write) communication mechanisms to classify direct inter-cluster communication architectures. The reads and writes can be from either the same cluster or across clusters. The communication can be either using a point-to-point network, a bus or a buffered point-to-point connection. An underlying assumption is that FUs always have one path to their RF (both read and write) which they may or may not use. A few examples of these architectures are shown in Figures 2 and 3. The architecture shown in Figure 3(a) has been used by Siroyan [20], Transmogrifier [15] etc.

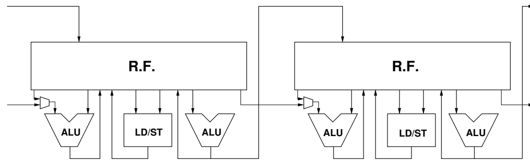
Our complete design space of clustered architectures is shown in Table 2. The Columns 1 and 2 marked as *Reads* and *Writes* denote whether the reads and writes are across (*A*) clusters or within the same (*S*) cluster. Columns 3 and 4 marked as *RF→FU* and *FU→RF*, specify the interconnect type from register file to FU and



(a) Write Across-1

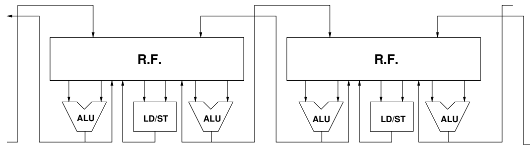


(b) Read Across-1

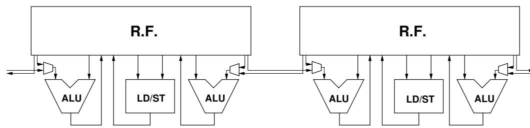


(c) Write/Read Across-1

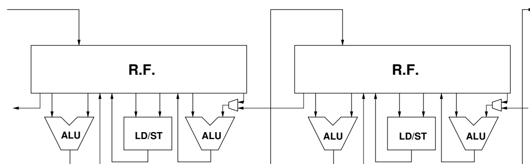
Figure 2: Architectures with $(\delta = 8)$



(a) Write Across-2



(b) Read Across-2



(c) Write/Read Across-2

Figure 3: Architectures with $(\delta = 4)$

from FU to register file respectively. Here, *PP* denotes *Point-to-Point* and *PPB* denotes *Point-to-Point Buffered*. This table also shows in Column 5, the commercial or research architectures which have been explored in this complete design space. For example the TiC6x is an architecture, which reads across clusters and writes to the same cluster; it uses buses for reading from RFs and point-to-point connections for writing back results to RFs.

We would like to contrast here our classification with what has been presented in [23]. They have only considered five different communication mechanisms and have not presented a classification. The *bus-based* and *communication FU* based interconnects which they have considered are part of the RF-to-RF type communication domain in our classification. The *extended results* type architecture is a *write across* architecture in our classification and *extended operands* is basically a *read across* type of architecture as per our classification. However, here again, they have considered only one type of interconnect, point-to-point, whereas others such as point-to-point buffered, buses are also possible. These have been shown in Table 2.

It can be clearly seen from Table 2 that a large range of architectures has not been explored. For each of these architectures an important metric is the maximum hop distance between any two clusters (δ). For example in case of architecture in Figure 2(a) $\delta = 8$ and for architecture in Figure 3(b), $\delta = 4$ assuming an 8-cluster configuration. δ is not an independent parameter, it can be calculated from the architecture type and number of clusters ($n_{clusters}$).

Reads	Writes	RF→FU	FU→RF	Available Archs.
S	S	PP	PP	TriMedia, FR-V, MAP-CA, MAJC
A	S	PP	PP	
A	S	Bus	PP	Ti C6x
A	S	PPB	PP	
S	A	PP	PP	Transmogrifier, Siroyan, A|RT
S	A	PP	Bus	
S	A	PP	PPB	
A	A	PP	PP	
A	A	PP	Bus	
A	A	PP	PPB	
A	A	Bus	PP	
A	A	Bus	Bus	
A	A	Bus	PPB	
A	A	PPB	PP	
A	A	PPB	Bus	
A	A	PPB	PPB	

Table 2: Overall Design-Space for Direct Communication Architectures

4. DESIGN SPACE EXPLORATION METHODOLOGY

Figure 4 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. This trace is fed to the DFG generating phase, which generates a DFG out of this instruction trace. 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 numbers of groups is reduced to the number of clusters in the architecture. The binding phase, binds these groups of chains to the clusters. It needs to be noted that the operation to FU binding is done in two steps. 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 [11, 17], is contained to a large extent, while still keeping the overall problem size manageable. Lastly, a scheduling phase schedules the operations into appropriate schedule steps. More details of each of these phases are presented in subsections below.

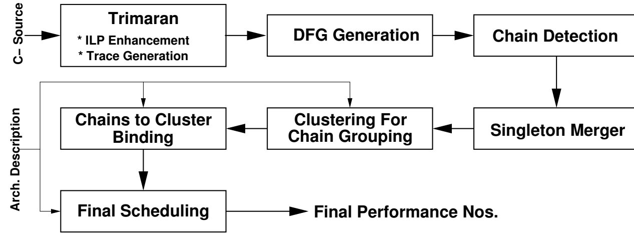


Figure 4: DSE Framework

4.1 DFG Generation

The DFG generation is carried out as described in Section 2.

4.2 Chain Detection

Our main emphasis is on minimizing communication amongst various clusters. The long sequences of operations denote compatible resource usages, as well as production and consumption of values. This makes them ideal candidates for merger into a single cluster [10]. Since the operations are sequential any ways, no concurrency is lost due to this merger. In the chains detection phase long sequences of operations are found in this DFG. The idea is to bind these chains to one cluster. Since, the DFG is acyclic, standard algorithms can be applied to find the long sequence of operations (chains). It needs to be noted that the detected chains are not unique, and will vary from one chain detection algorithm to another.

4.3 Singleton Merger

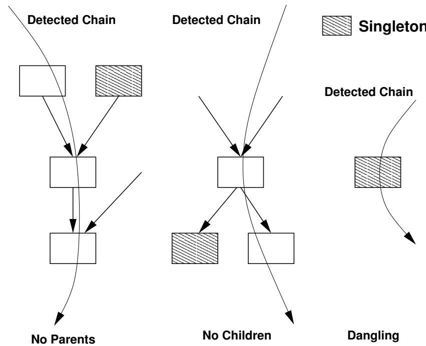


Figure 5: Reasons for Singleton Generation

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. The singletons can be generated due to a number of reasons. The three most prominent of these are shown in Figure 5. In the first case one of the source nodes is left alone as a consequence of merger of its only child. In the second case the destination node is left alone as a consequence of merger of both its parents in separate chains. In the last case, the node doesn't have any parents or children and as a consequence is dangling. In the singleton merger 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 do not have any source or destination dependency. Nodes, which have no destinations (sources) are merged with the shortest chain of their sources (destinations).

4.4 Clustering

Algorithm 1 Clustering Algorithm

1: $resources \Leftarrow resources_per_cluster * n_clusters$
2: $schedule_pure_vliw(graph, resources)$
3: **while** $(no_of_chains(graph) > n_clusters)$ **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:       $merge_chains(dup_graph, dup_chains, i, j)$
9:       $a_{i,j} \Leftarrow estimate_sched(dup_graph, dup_chains)$
10:     **end for**
11:   **end for**
12:   $SORT(A)$ giving first priority to increase in sched_length. If the sched_length is equal, give priority to chains which have more communication edges. If this is also the same give priority to smaller chains.
13:   $n_merge \Leftarrow 0.1 * n_{chains}$ or number of mergers required to make new $n_{chains} = n_{clusters}$, whichever is lower.
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 2(a) and 3(c). While in the former, each cluster is write close to the adjacent cluster, in the latter, any two adjacent clusters are both read as well as write close. During the clustering process, in this stage we assume that finally the communication would happen through the best possible inter-cluster interconnects. Thus, this schedule length effectively represents a lower bound on the actual schedule length.

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 (Algorithm 2), 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 priority function. 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

Algorithm 2 Estimating Schedule (all except RF-to-RF)

1: Initialize the schedule step of all nodes in this chain to 0
2: prior_list $\Leftarrow$ Build priority list
3: $curr_step \Leftarrow 1$
4: **repeat**
5: Initialize all used capacities to zero
6: ready_list $\Leftarrow$ Build ready list
7: **for all** $(Resources)$ **do**
8: $curr_node \Leftarrow list_head(ready_list)$
9: **while** $(curr_node \neq NULL)$ **do**
10: $rqd_cap \Leftarrow$ *Incoming External Edges*
11: $resv_step \Leftarrow 0$
12: **if** $(rqd_cap + usd_cap > avail_cap_rd)$ **then**
13: **for all** (*Incoming valid external edges*) **do**
14: Update schedule of dirty nodes
15: Find first free write slot from this cluster
16: $usd_cap += 1$
17: $wrt_slot[src_cluster] \Leftarrow sched_step[src_node] + 1$
18: **if** $(usd_cap \geq max_cap_wr)$ **then**
19: $usd_cap \Leftarrow 0$
20: $wrt_slot[src_cluster] += 1$
21: **end if**
22: $resv_step \Leftarrow max(all\ write\ slots)$
23: **end for**
24: **end if**
25: $curr_node.sched_step \Leftarrow max[resv_step, curr_step]$
26: Mark all out nodes as dirty
27: $list_remove(ready_list, curr_node)$
28: **end while**
29: **end for**
30: $curr_step += 1$
31: **until** $(prior_list.n_nodes \neq 0)$
32: Return max. schedule length

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.

Steps 1 and 2, prepare the graph for scheduling. The priority list, contains the nodes in this chain sorted in descending order of distance-to-sink. The main loop (steps 4 to 31) is repeated till all the nodes in the graph have been scheduled. The second loop body, (steps 7 to 29), tries to schedule as many operations in this cycle as possible. It starts by picking the node at the head of the ready list (step 8) and checks if the external edges feeding this node, exceed the read capacity (step 12). The read capacity is simply the number of external values which can be simultaneously read by a cluster. For example for the architecture shown in Figure 2(b), the read capacity is one and for architecture shown in Figure 3(b) it is two. The node assumes best connectivity between any two clusters. If the required capacity is exceeded it tries to estimate the effect of transfer delay as well as book transfer slots (steps 14 to 22). When all the nodes in the chain have been scheduled the algorithm return the value of maximum schedule length in the graph.

We observed the results of this phase on small graphs using a graph visualization tool. The set of heuristics is effectively able to capture all the connected components as described in [3]. Figures 6 and 7, show the detected connected components for two examples. The numbers inside the nodes, show *node no.::group no*. The source code in both these cases was doing some computation and the resultant value was being assigned to distinct matrix elements. It needs to be noted that we have nowhere done an explicit connected component detection as in [3].

4.5 Binding

The next step is to bind these groups of chains to clusters. Although the value of $n_{clusters}$ is quite small (8 in our case and generally not more than 16), still the number 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 four times larger than the optimal ones.*

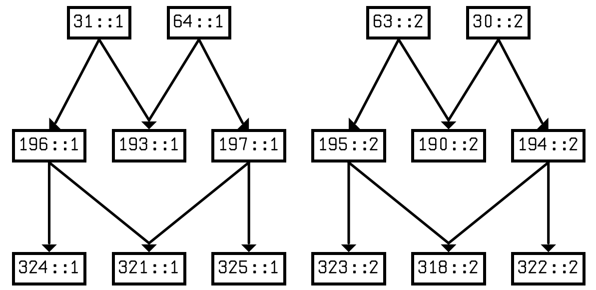


Figure 6: Detected Connected Components (I)

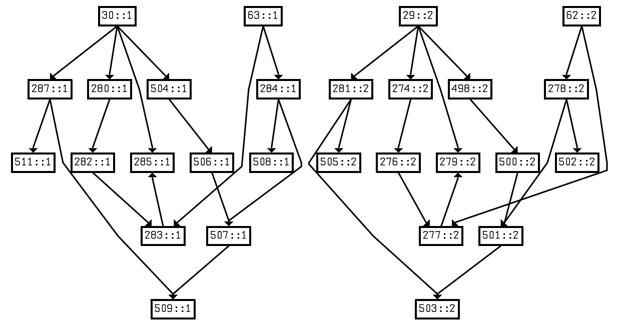


Figure 7: Detected Connected Components (II)

Algorithm 3 Binding Algorithm

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

The binding heuristics are driven by what impact the communication latency of a particular node will have on the 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 V_i to V_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 consider 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.

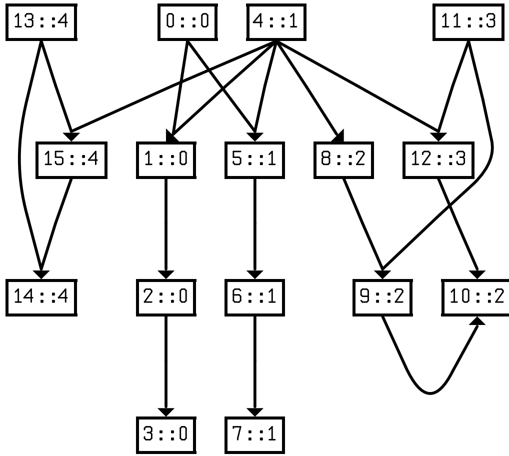


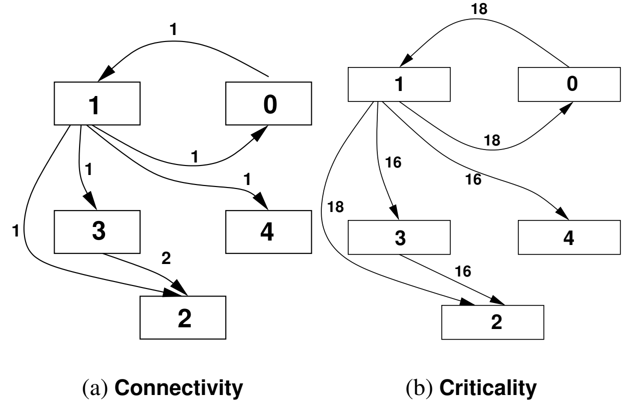
Figure 8: Input Graph for Binding

Algorithm 3, 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 of architectures which can both read from as well as write to the adjacent clusters. Partially motivated by this and partially by [12], we thus bring in an additional iterative improvement phase, by performing a local search around this initial binding (step 6 to 13).

An example of this appears in Figures 8 and 9. The input graph is shown in Figure 8 and the corresponding connectivity, with each of the group of chains as node is shown in Figure 9(a). For the connectivity graph, the edge weights represents number of transfers across groups of chains. Looking at this graph, it appears that groups, 2 and 3 which are the most heavily connected need to be assigned to clusters which are close. However, from the input graph in Figure 8, it is clear that this is not the case. Both the groups accept input from group 1, and till the time that communication doesn't take place, nodes 8 and 12 cannot be scheduled. This makes the connections between groups 1 and 2 more critical than those between groups 2 and 3. The criticality graph shown in Figure 9(b), shows that using the set of heuristics above, this fact has actually been brought out.

4.6 Final Scheduling

The scheduling phase is architecture specific. We have separate



(a) **Connectivity** (b) **Criticality**

Figure 9: Connectivity and Criticality Between Clusters

schedulers for RF-to-RF type architectures and direct inter-cluster communication architectures. Although a few scheduling algorithms for clustered architectures have been reported in literature, they all are specifically meant for RF-to-RF type of architectures [3, 12, 19]. 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 hop paths in direct communication architectures. It finds the shortest path to the closest cluster to which data needs to be transferred and moves data to this cluster. If the communication mechanism is bidirectional as in Figure 3, it moves data in both the directions. It repeats this process till data has been moved to all the required clusters. During experimentation we realized that efficient result propagation is very important for processors with large number of clusters (more than four).

5. EXPERIMENTAL RESULTS AND OBSERVATIONS

Our framework is very flexible, as can be seen from the results. It supports architecture exploration for a wide design space. However, results based on several parameters could not be presented in this paper due to paucity of space e.g. those with each cluster connected to two or more clusters for each of the architecture types, those having dedicated communication FUs for inter-cluster communication etc. The architectures for which we present results are shown in Figures 1, 2 and 3. For the RF-to-RF architectures it is assumed that the number of buses is 2 and the bus latency is 1.

The obtained ILP numbers for the various architectures for different cluster configurations are shown in Tables 3 and 4. Here the different benchmarks are the actual functions from the benchmark suites of DSP-Stone and MediaBench as has been discussed in Section 2. The register file for each of the architectures has sufficient number of ports to allow simultaneous execution on all the FUs e.g. for a monolithic RF architecture the RF has $24*3 = 72$ *Read* and $24*1 = 24$ *Write* ports (assuming 1 read port for predicate input). This is done so as to ensure that there is no concurrency loss due to insufficient number of ports and the effect of interconnection architecture stands out. Abbreviation *PV* (column 2) has been used in this table for *Pure VLIW (or single RF)*, *RF* (column 3) for RF-to-RF, *WA* for *Write Across* (column 4 and 7), *RA* for *Read Across*

Bench.	PV	RF	$\delta = 4$ WA.1	RA.1	WR.1	$\delta = 2$ WA.2	RA.2	WR.2
matrix_init	9.80	6.28	6.12	6.12	6.12	6.12	6.12	6.12
biquad	10.67	2.61	8.56	8.47	8.75	8.75	8.85	9.06
mm	11.68	3.19	6.81	9.28	8.48	8.48	7.80	8.52
insert_sort	9.45	4.09	7.22	7.19	7.80	7.92	7.87	7.94
h2v2_fancy	9.71	2.04	6.02	5.64	6.85	6.84	6.17	6.67
encode_one	10.28	2.19	6.46	6.50	7.50	6.72	6.65	6.69
h2v2_down	11.06	3.43	5.61	5.49	5.94	5.94	5.50	5.90
form_comp	10.75	5.18	5.21	5.43	5.61	5.61	5.52	5.58
decd_mpeg1	10.90	2.16	7.43	7.60	8.61	8.61	8.14	8.67
dist1	7.44	2.62	6.41	7.00	7.02	7.15	6.41	7.15
pred_zero	9.48	4.88	5.26	5.37	5.57	5.57	5.53	5.57
pred_pole	9.56	5.49	5.16	5.16	5.38	5.27	5.16	5.27
tndm_adj	10.19	5.71	6.85	6.34	6.85	6.85	6.73	6.85
update	9.70	2.20	6.31	6.36	7.36	7.36	7.16	7.43
g721enc	10.50	3.62	6.54	6.34	6.72	6.69	6.60	6.63
ILP as Fraction of PV (%)								
Avg. (%)	-	35.32	65.56	65.45	70.57	69.64	66.56	69.82
Max. (%)	-	64.08	86.16	94.09	94.35	96.1	86.16	96.1
Min. (%)	-	19.82	48.47	49.64	52.19	52.19	49.73	51.91

Table 3: ILP for (8-ALU, 4-MEM) 4-Clust Architectures

Bench.	PV	RF	$\delta = 8$ WA.1	RA.1	WR.1	$\delta = 4$ WA.2	RA.2	WR.2
matrix_init	20.42	11.67	11.14	11.14	11.14	11.14	11.14	11.14
biquad	21.64	2.00	12.98	13.91	15.58	15.58	15.90	16.23
mm	19.64	2.12	9.90	13.06	15.60	15.60	14.69	14.69
insert_sort	18.94	2.21	11.03	12.32	13.48	13.48	13.39	13.62
h2v2_fancy	19.38	2.07	10.00	9.63	13.04	13.08	11.71	12.55
encode_one	19.14	2.18	12.02	13.04	14.44	14.39	14.39	14.70
h2v2_down	19.04	2.05	4.90	6.45	6.35	6.35	6.49	6.45
form_comp	21.72	4.63	7.85	9.91	10.86	10.98	10.98	10.98
decd_mpeg1	21.81	2.18	9.45	10.24	12.40	12.52	12.29	12.64
dist1	7.44	3.38	5.03	5.72	7.02	7.02	5.47	6.64
pred_zero	19.20	4.98	5.37	5.41	5.82	5.77	5.77	5.77
pred_pole	19.85	3.35	9.92	9.92	10.32	10.32	10.32	10.32
tndm_adj	17.95	2.58	11.78	11.42	12.16	12.16	12.16	12.16
update	19.39	3.46	11.20	11.36	13.25	13.03	12.42	12.42
g721enc	21.14	2.07	13.64	12.47	14.17	14.17	13.14	13.51
ILP as Fraction of PV (%)								
Avg. (%)	-	18.23	49.70	54.46	62.81	62.66	60.10	61.77
Max. (%)	-	57.15	67.61	76.88	94.35	94.35	75.18	89.25
Min. (%)	-	9.09	25.74	28.18	30.31	30.05	30.05	30.05

Table 4: ILP for (16-ALU, 8-MEM) 8-Clust Architectures

(column 5 and 8) and *WR* for *Write/Read Across* (column 6 and 9). The numbers in these tables, show the ILP for each of the benchmark functions e.g. for insert sort for a 4-cluster configuration, the ILP for a write across architecture (Figure 2(a)) is 7.22, for read across (Figure 2(b)). From the results we conclude the following:

1. Loss of concurrency vis-a-vis pure VLIW is considerable and application dependent.

2. In few cases the concurrency achieved is almost independent of the interconnect architecture. This denotes that a few grouped chains in one cluster are limiting the performance along with a few critical transfers.

3. For applications with consistently low ILP for all architectures the results are poor due to large number of transfers amongst clusters.

4. In some cases the performance in case of $n_{clusters} = 4$ architecture is better than performance in case of $n_{clusters} = 8$ architecture (dist1). This is because of the reduced hop distance amongst clusters and this behavior is common across different architectures. In such cases communication requirements of the application are spread across all the clusters, so, as the average number of hops comes down the performance increases. This happens even though the supported concurrency has decreased due to less number of FUs.

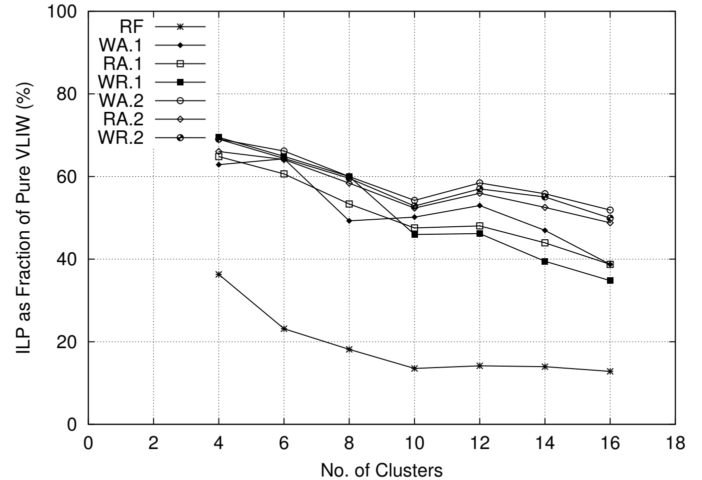


Figure 10: Average Variation in Performance With $n_{clusters}$

Figure 10, shows the variation of average ILP as fraction of Pure VLIW versus number of clusters for each of the architectures. While for $n_{clusters}$ less than 8, the behavior of different interconnection networks is not brought out, once $n_{clusters}$ grows beyond 8, the superiority of *WR.2* and *WA.2* is clear. Both of these are able to deal well with the applications which require heavy communication. The minimum loss of concurrency in these cases is around 37% and 30% for architectures with $n_{clusters} = 8$ and $n_{clusters} = 4$ respectively. It needs to be noted that *WR.2* is the type of interconnect employed by most commercial as well as research architectures as shown in Table 2. The performance of *RA.2* in general, is inferior to *WA.2* because if there is more than one consumer of the propagated value in the target cluster, the value first needs to be moved to target cluster using the available read path, which amounts to an additional processor cycle. It is interesting to note that the performance of RF-to-RF type of architecture is quite poor, with an average loss of 81% for $n_{clusters} = 8$ and 64% for $n_{clusters} = 4$ and deteriorates further with increase in number of clusters. This is ignoring the latency of such transfers using global buses (assumed bus latency is 1) vis-a-vis local transfers. It would be quite interesting to identify application characteristics by which the suitability of an architecture can be established.

6. CONCLUSIONS AND FUTURE WORK

The main contributions of this paper are as follows:

1. We have proposed and implemented a framework for evaluation of inter-cluster interconnections in clustered VLIW architectures.

2. We have provided a concrete classification of the various inter-cluster interconnection mechanisms in clustered VLIW processors.

3. We have evaluated a range of clustered VLIW architectures and results conclusively show that application dependent evaluation is critical to make the right choice of an interconnection mechanism.

As can be seen once the propagation delay dominates, the different inter-cluster transfer mechanisms fail to achieve performance close to Pure VLIW. Also, the RF-to-RF type of architectures fail to perform to expected levels because even for adjacent clusters they have to gain access to global buses which are shared. Based on these observations, we propose to investigate an architecture which has both buses as well as direct connections. Another issue which has not been addressed by us is effect of interconnection architecture on clock period as this will also impact performance.

7. ACKNOWLEDGMENTS

The authors would like to acknowledge the partial support of *Naval Research Board*, Govt. of India towards the Multi-processor Embedded System Project at IIT Delhi of which this work is a part. The authors would also like to thank Rohit Khandekar, Department of Computer Science and Engineering, IIT Delhi for the many useful discussions and suggestions.

8. REFERENCES

[1] Sanjeev Banerjia, William A. Havanki, and Thomas M. Conte. Treegion scheduling for highly parallel processors. In *European Conference on Parallel Processing*, pages 1074–1078, 1997.

[2] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu. IMPACT: An architectural framework for multiple-instruction-issue processors. In *Proceedings of the 18th International Symposium on Computer Architecture (ISCA)*, volume 19, pages 266–275, New York, NY, 1991. ACM Press.

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

[4] Paolo Faraboschi, Geoffrey Brown, Joseph A. Fisher, Giuseppe Desoli, and Fred (Mark Owen) Homewood. Lx: A technology platform for customizable VLIW embedded processing. In *International Symposium on Computer Architecture (ISCA'2000),*, 2000.

[5] Jason Fritts and Bill Mangione-Smith. MediaBench II - Technology, Status, and Cooperation. In *Workshop on Media and Stream Processors, Istanbul, Turkey*, November 2002.

[6] Jason Fritts and Wayne Wolf. Evaluation of static and dynamic scheduling for media processors. In *2nd Workshop on Media Processors and DSPs (held in conjunction with MICRO-33)*, 2000.

[7] Jason Fritts, Zhao Wu, and Wayne Wolf. Parallel Media Processors for the Billion-Transistor Era. In *International Conference on Parallel Processing*, pages 354–362, 1999.

[8] W. W. Hwu, Mahlke, S. A., W. Y. Chen, P. P. Chang, N. J. Warter, R. A. Bringmann, R G Ouellette, R. E. Hank, T. Kiyohara, G. E. Haab, J. G. Holm, and D. M. Lavery. The superblock: An effective technique for VLIW and superscalar compilation,. *J. Supercomputing,*, 1993.

[9] M. Jacome and G. de Veciana. Design challenges for new application specific processors. In *IEEE Design and Test of Computers*, number 2, pages 40–50, 2000.

[10] Margarida F. Jacome, Gustavo de Veciana, and Viktor Lapinskii. Exploring performance tradeoffs for clustered VLIW ASIPs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'2000)*, November 2000.

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

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

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

[14] Hsien-Hsin Lee, Youfeng Wu, and Gary Tyson. Quantifying instruction-level parallelism limits on an EPIC architecture. In *Proc. of ISPASS*, 2000.

[15] D. Lewis, D. Galloway, M. Ierssel, J. Rose, and P. Chow. The transmogrifier-2: A 1-million gate rapid prototyping system, 1997.

[16] S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, and R. A. Bringmann. Effective compiler support for predicated execution using the hyperblock. In *25th Annual International Symposium on Microarchitecture*, 1992.

[17] Emre Ozer, Sanjeev Banerjia, and Thomas 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.

[18] Scott Rixner, William J. Dally, Brucek Khailany, Peter R. Mattson, Ujval J. Kapasi, and John D. Owens. Register organization for media processing. In *Proceedings of 6th International Symposium on High Performance Computer Architecture*, pages 375–386, May 2000.

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

[20] Siroyan. http://www.siroyan.com.

[21] Peter Song. Demystifying EPIC and IA-64. *Microprocessor Report*, 1998.

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

[23] Andrei Terechko, Erwan Le Thenaff, Manish Garg, Jos van Eijndhoven, and Henk Corporaal. Inter-Cluster Communication Models for Clustered VLIW Processors. In *9th International Symposium on High Performance Computer Architecture, Anaheim, California*, pages 298–309, February 2003.

[24] Trimaran Consortium. The Trimaran Compiler Infrastructure, http://www.trimaran.org, 1998.

[25] 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, October 1994.