International Journal of Parallel Programming (© 2007)
DOI: 10.1007/s10766-007-0045-2

Evaluation of Bus Based Interconnect Mechanisms in Clustered VLIW Architectures

Anup Gangwar,[1,3] M. Balakrishnan,[2] Preeti Ranjan Panda,[2] and Anshul Kumar[2]

Received September 7, 2006; accepted March 23, 2007

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 provide an attractive solution to address this issue. Recent studies with a wide variety of inter-cluster interconnection mechanisms have reported 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 calculated post-layout clock periods. The results demonstrate that while there is less that 10% variation in interconnect area, the bus based architectures are slower by as much as 400%. 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.

KEY WORDS: Performance evaluation; VLIW; ASIP; Clustered VLIW processors.

[1]Freescale Semiconductors Pvt. Ltd., Plot No. 18, Sector-16A, Noida 201 307, India.
[2]Department of Computer Science and Engineering, IIT Delhi, Hauz Khas, New Delhi 110 016, India. E-mail: {mbala, panda, anshul}@cse.iitd.ac.in
[3]To whom correspondence should be addressed. E-mail: anup.gangwar@freescale.com

1. Introduction

Media applications have large amount of instruction level parallelism (ILP) which justifies building very high issue-rate processors.[1–8] 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.[7,9,10] 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,[11] HP-ST Microelectronic's Lx[12] 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.[13] The RF delay, power and area are more sensitive to the increase in number of access (read/write) 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 architecture plays a crucial role in processor performance.[14,15]

Till recently only one type of inter-cluster interconnect, a simple bus connected to all the clusters (see Fig. 1f), 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[14] and[15] reported substantial performance gains in number of cycles when various other mechanisms are used. While these studies have established the need for more variety in inter-cluster interconnect mechanisms they do not report implementation characteristics of these mechanisms. Several questions need to be answered before one type of interconnect can be chosen over other: What is the impact on clock period of these interconnect mechanisms? What is the penalty in terms of increase in interconnect area which these mechanisms result in? Moreover, both these studies used few specific design points for evaluating bus based interconnects. In[14] a single cycle transfer bus was connected to all the clusters whereas in[15] two single cycle transfer buses were connected to all the clusters.

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

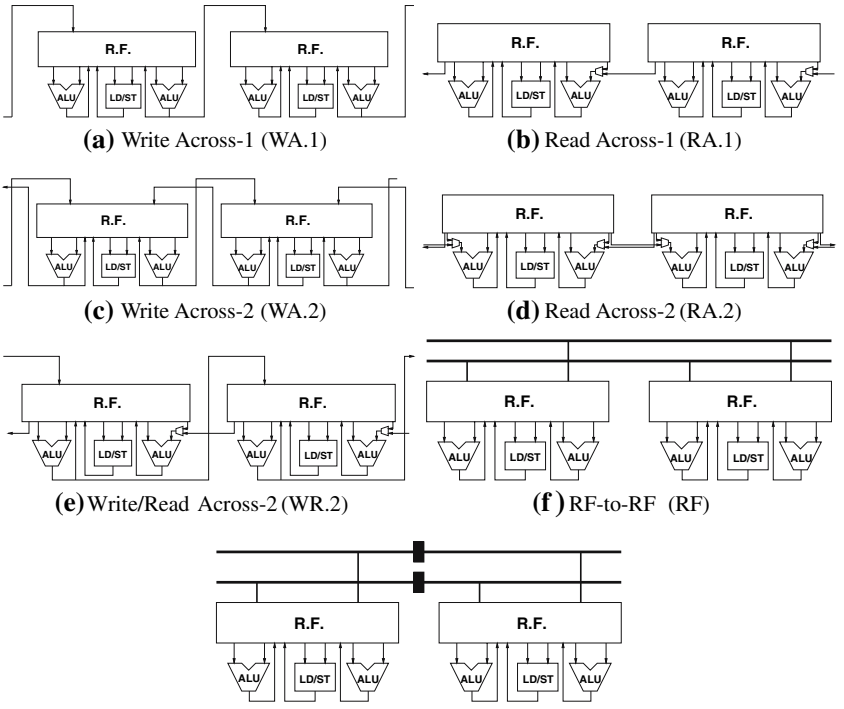


Fig. 1. Various inter-cluster interconnection mechanisms.

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. ARCHITECTURE OF MODELED PROCESSORS

To focus primarily on the inter-cluster interconnect mechanisms, we model a family of very simple processors. However, this set of processors support all the interconnect mechanisms as discussed in Ref. 15. These interconnect mechanisms have been replicated for convenience in Fig. 1. Interconnect mechanisms presented in Ref. 14 are a subset of this domain. Authors additionally dealt with the problem of issue slots in

Ref. 14. We have made specific choices regarding issue slots and these have been explained at appropriate places later on.

All the processors in our set have a common Instruction Set Architecture (ISA). The ISA resembles that of a simple RISC processor having load/store architecture with standard ALU, data movement and branch operations. This is shown in Table I. In this table the column *Mnemonic*, gives the instruction mnemonic, column *Explanation*, gives instruction semantics and column *Cycles*, shows number of cycles required to execute this instruction. The cycle count values themselves have been arrived at after analyzing the low-level implementations of each of the operations as well as existing processor architectures. For simplicity, our modeled processors contains only integer operations but addition of floating point operations is a simple extension. Each of the operations with the exception of immediate move (MVIH and MVIL) instructions takes register addresses in source and destination fields. Even the load (store) instructions require the address to be present in a register before the data can be loaded (stored) from (to) memory. This has been done so as to keep the control part as simple as possible, while the data path still mimics a real processor. Immediate move operation has been broken into MVIH and MVIL instructions, which load half the higher order and lower order bytes, respectively. This is done to keep the instruction size same for all instructions. For example if the data-width is set as 32, MVIH loads two most significant bytes and MVIL two least significant bytes into the destination register. RFMV instruction is special and is present only in RF-to-RF type of architecture. It moves values from source cluster (cluster in which this instruction is issued) to any destination cluster.

The processor has a reduced version of the classical five stage pipeline. The various pipeline stages are shown in Fig. 2. All the operations are fully pipelined. ALU instructions take variable number of cycles in their execution stage (1–3), while load and store instructions take three cycles each. Load/store unit is again fully pipelined and a new operation can be started on them in each cycle. The branch instructions, BRL and BRA take three cycles each to complete and require a pipeline flush.

Figure 3 gives the instruction encoding for various instructions. For *Write Across* type of architectures, one additional bit is required in the destination register field as these architectures can write to adjacent clusters. The *Read Across* architectures can read one operand from adjacent cluster and thus an additional bit is required in one of the source register fields. Using similar reasoning *Read/Write Across* instructions require an additional bit in one of the source operand fields and the destination register field, as they can both read from and write to the adjacent cluster. The *RF-to-RF* architectures contain specialized instructions to move data

Table I. Instruction Set Architecture of Modeled Processors

Mnemonic	Explanation	Cycles
NOP	No operation	1
ADD	$R_{dest} \Leftarrow R_{src1} + R_{src2}$	1
SUB	$R_{dest} \Leftarrow R_{src1} - R_{src2}$	1
MUL	$R_{dest} \Leftarrow R_{src1} * R_{src2}$	3
DIV	$R_{dest} \Leftarrow R_{src1}/R_{src2}$	3
MOV	$R_{dest} \Leftarrow R_{src}$	1
LD	$R_{dest} \Leftarrow MEM[R_{src1}]$	3
ST	$MEM[R_{dest}] \Leftarrow R_{src1}$	3
CMP	$R_{dest} \Leftarrow R_{src1} > R_{src2}$	1
BRL	Branch if less to $Address$	3
BRA	Branch always to $Address$	3
MVIH	Load (Immediate) higher order bytes of R_{dest}	1
MVIL	Load (Immediate) lower order bytes of R_{dest}	1
RFMV	Inter-cluster move for RF-to-RF Architectures	1

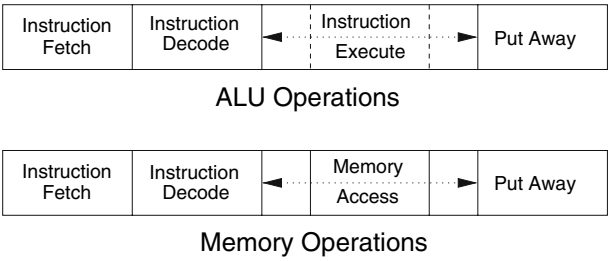


Fig. 2. Processor pipeline.

between clusters (copy operations). This instruction is issued in the regular ALU issue slot and not in any additional issue slot. Destination register in this case consists of the address of destination cluster and the register in that particular cluster. Since, we have considered architectures with up to 16 clusters, the destination cluster field contains only four bits. The destination register field requires $log_2(RF\ Size)$ bits to point to a unique location in the destination clusters RF. Encoding for other instructions is straight forward. The instruction encoding has deliberately been kept simple, to ensures a low complexity instruction decoder.

Figure 4, shows the detailed architecture of modeled processors for *WA.1* type of inter-cluster communication. It needs to be noted that the processor does not contain any instruction or data cache. In clustered architectures, a centralized cache presents scalability problems, an area

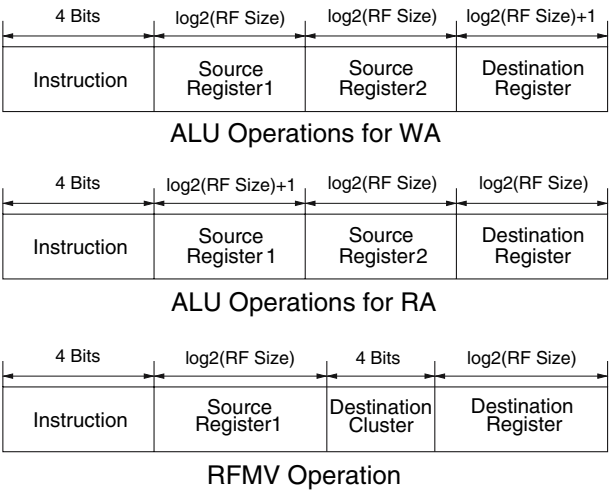


Fig. 3. Instruction encoding.

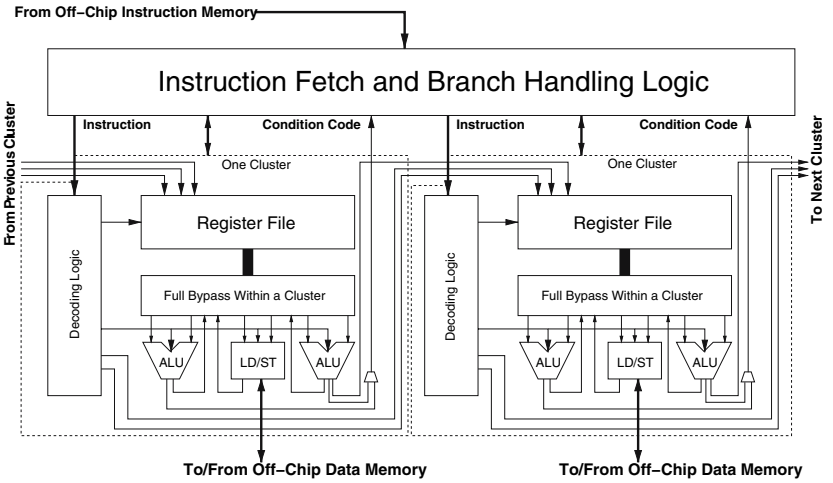


Fig. 4. Detailed architecture of clusters for WA.1.

which is actively being researched.[16] Assuming that an efficient cache architecture exists which does not constrain the critical path, the caches can be removed from our models without impacting the results.

The off-chip data memory is assumed to be banked, so, each load store unit has direct path to a particular data memory module. The instruction RAM supplies an entire VLIW word in parallel, i.e., single

instruction fetch. It is assumed that the RAM has sufficient data-width to handle such parallel data transfer either through banking or other mechanisms. This helps keep the instruction fetch mechanism simple and removes any need for an instruction align mechanism.

Any instruction execution progresses as follows: the centralized instruction fetch mechanism gets instruction from the off-chip instruction RAM in a single cycle. This is registered and presence of a branch instruction is tested. The branch instruction can only be issued in the first instruction slot. If a branch instruction is found, the next instruction is fetched from the branch target location depending on the branch instruction as well as the condition code. The condition code itself is obtained from the various clusters and is registered inside the branch handling logic, i.e., a centralized place. Since the branch instructions always contain an absolute location encoded in the instruction itself, no other information is required from the clusters to execute this branch instruction.

If a branch instruction is not found, then normal execution must continue. In this case, the long instruction (*MultiOp*) is broken up into cluster level instructions (*UniOps*) and forwarded to each cluster. Further decoding is done in the decode logic of each cluster itself. In case any compare operation is executed, then the condition code is latched in the control register inside the branch handling unit during the put away stage of pipeline. Memory operations requiring access to off-chip data memory have direct paths using the load/store units present in each of the clusters. For inter-cluster communication, read, write, control and data signals are generated individually from each of the clusters. In the *Write Across-1* type architecture, one write port of the RF in any cluster is connected directly to the control and data signals from adjacent cluster. These control signals are generated from the decode unit of adjacent cluster. The ALU output lines are directly connected to the data port. In case any inter-cluster write needs to be executed, the control logic sets the corresponding enable bit high. Each cluster contains a full bypass logic. However, inter-cluster transfer paths feed directly to the RF and not to the bypass logic. We believe that feeding to the bypass would result in a critical path through it.

The architectures for other interconnect mechanisms are similar. *Read Across* architectures have input paths to the ALUs after multiplexing. Generation of control signals for these are handled in the decode logic within each cluster, rather than the centralized unit. *RF-to-RF* architectures, rely on global broadcast of data and register address. This is handled by decode logic within each cluster. Bus arbitration in this case is not required as the instructions have been scheduled by the compiler resolving the bus contention. However, in case of multiple buses, additional

multiplexing logic is required at each RF's data input and output ports. This is assuming that each bus does not require any additional port in RF of each cluster. What this effectively means is that only one data element can be moved to any cluster at a time, however, multiple such transfers may be in progress at the same time. These mechanisms have been discussed at length in Ref. 14. The multiplexing logic if present, is again controlled by decode logic within each cluster.

The processors are modeled using parameterized RTL description (VHDL). 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 Ref. 17. RF for any processor is custom designed.[18,19] Since, we have neither designed nor have access to such a RF, we included a dummy RF during synthesis. This RF does not contain the complicated multiplexing logic which a real RF has, however, it does ensure that the proper port mappings are in place so that the static timing analysis (STA) reports proper paths. The entire VHDL has been written at Register Transfer Level (RTL) to obtain most efficient implementation. The parameterized VHDL model allows variation in the following features:

1. Interconnect architecture
2. Number of clusters
3. Register file size
4. Register file (and ALU) width
5. Data RAM size
6. Instruction RAM size

We use the industry standard, Synopsys frontend and Cadence backend flow. Synopsys Design Compiler, synthesizes the RTL to a gate-level netlist. To obtain the best results from Design Compiler, the system is given a very tight timing constraint. This gate-level netlist is then placed and routed using Cadence SoC Encounter. Functional simulation is done using test programs at various stages of implementation to ensure correctness of design. Initially we experimented with a flat place and route, however, the results were not satisfactory. In a flat place and route, the tools moved decode and other logic to far away locations, which naturally led to long critical paths through this logic. Thus to obtain analyzable results we perform a hierarchical place and route. Towards this end, the clusters, as shown in Fig. 4, are individually synthesized and a flat place and route done on them. After this, timing and hierarchical abstracted models are created which are used further in the flow. The top level flow does not redistribute the logic contained in these models, as it treats them as

black-boxes. The only limitation is that the tools currently do not support rearrangement of pins on black-boxes during a top level place and route run, once they have been created earlier. To circumvent this problem, we do a manual pin assignment based on final expected chip floorplan. The clk (clock) and reset pins are handled separately, for which buffer insertion is done so as to maintain the best possible timing.

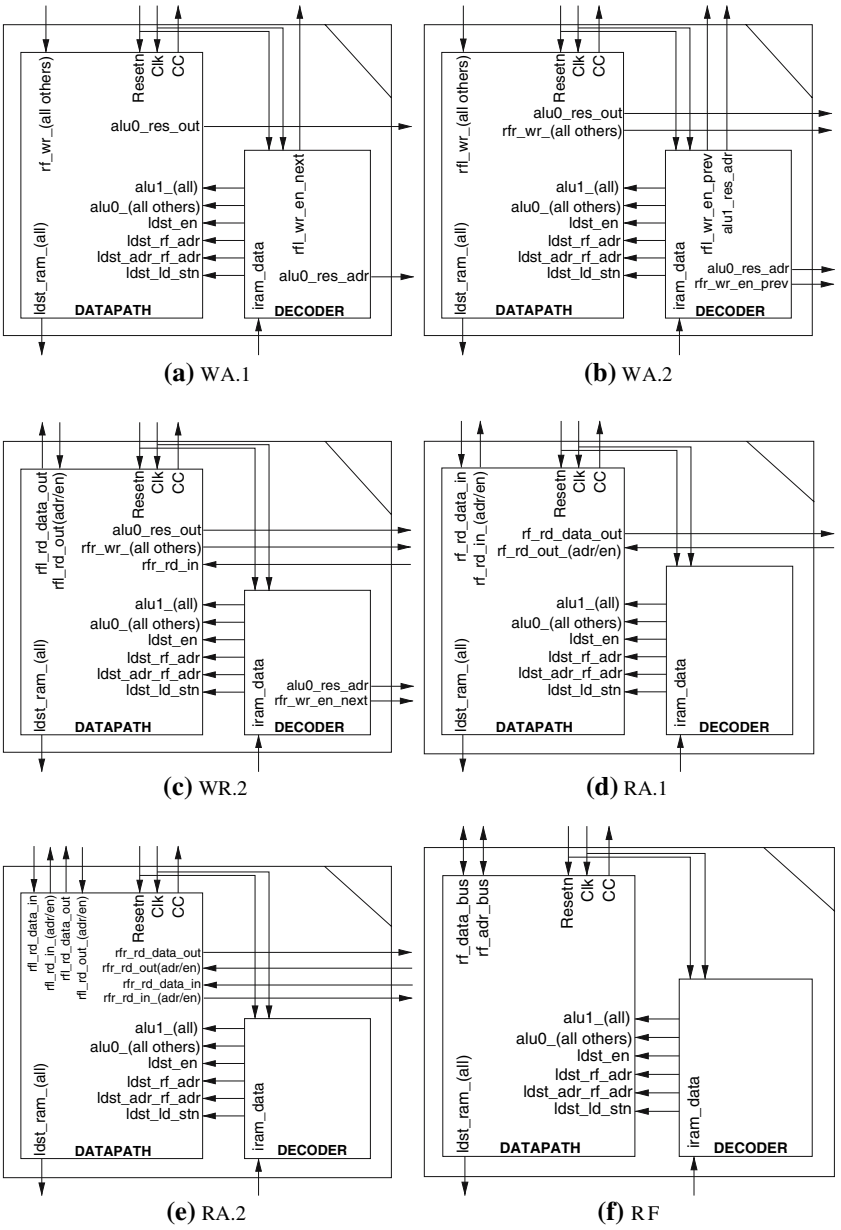
3. EVALUATING EFFECT ON CLOCK PERIOD

3.1. Evaluation Methodology

Figure 5 shows the pin organization for each of the cluster types. The clusters are placed and routed with our supplied pin positions. The clusters as shown, with the slit at the top right corner are in upright position. During top-level place and route, the tool re-positions as well as rotates the clusters to obtain better routing and timing. After extensive experimentation with various aspect ratios, we chose the aspect ratio of one as this gives better final floorplans and individual cluster timing. However, the final chip (entire processor) is not necessarily square for all cluster configurations.

To understand why the pin positions have been chosen in this manner, it needs to be noted that some of the wires go to centralized branch handling and instruction fetch unit, while other wires are needed for either connecting to the adjacent cluster or for load/store unit's communication with off-chip data RAM. The type of interconnects which we are evaluating can be efficiently routed if the clusters themselves are organized in a ring fashion. This is so because apart from *RF-to-RF* type of architecture, all other architectures only rely on neighbor connectivity. The pins are so organized that a simple rotation of the individual cluster can provide better connectivity. The pin positions on clusters aid such a final placement during floorplanning. Since, the final chip need not be square, we have fixed various aspect ratios for various cluster configurations. For example, in case of a two cluster configuration the ratio is 0.5, in case of a four cluster configuration it is 1.0 etc. The *RF-to-RF* type of architectures are also organized in a similar fashion. The inter-cluster buses are laid out in between the clusters.

For *Write Across* type of architectures, shown in Fig. 5, the *rf_wr*, wires are RF ports, which have directly been pulled out for adjacent cluster to be able to write to this RF. In case of bidirectional connectivity these are labeled with prefixes *rfl_* and *rfr_* which denote left and right cluster, respectively. The *alu0_* and *alu1_* signals are mapped to write ports of left and right clusters, respectively. For the *Read Across* type of



(a) WA.1 **(b)** WA.2

(c) WR.2 **(d)** RA.1

(e) RA.2 **(f)** RF

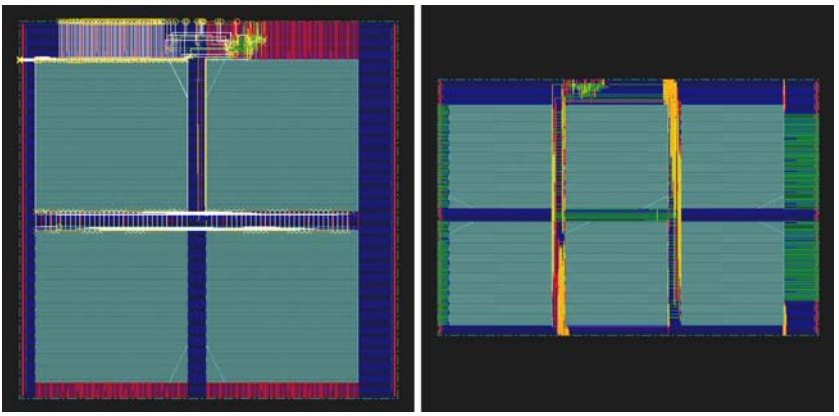
Fig. 5. Pin Organization for various clusters.

architectures, the *_in_* type of ports are used for reading in data from adjacent clusters. If the architecture has bidirectional connectivity, then the prefixes, *rfl_* and *rfr_*, denote the left and right clusters, respectively. The *_out_* type of wires are used to provide data to adjacent cluster. They are directly mapped to a read port on the RF. *Read/Write Across* architecture, contains a subset of connections of *Read Across* and *Write Across*. The most simple connectivity is in case of *RF-to-RF* architectures. They only require bidirectional address and data buses, which are neatly organized towards the top of each cluster.

Figure 6, shows the obtained floorplans after running the floorplanning tool for *RF-to-RF* type architectures. The slits, as mentioned previously, denote the top in an upright position of individual clusters. It is clearly seen that the clusters have been rotated and shifted to obtain the best timing. For the two cluster configuration, the inter-cluster transfer buses have been arranged in between them vertically, which is the best possible way to connect. For four cluster configuration, the buses run horizontally between all the clusters. For six cluster configuration, the buses have been arranged in the form of an *H*, which is one type of arrangement. The other option was to position the buses horizontally. However, the obtained timing in both these cases is same. The wires coming out of each cluster are the RAM signals from load/store units of clusters to off-chip data RAM blocks. The top level module (full processor) does not have any constraint on pin placements and so the off-chip connectivity pins are extended to nearest chip boundary. The floorplan for other interconnect architectures, *Write Across*, *Read Across* and *Read/Write Across*, is similar to what has been obtained for *RF-to-RF* architecture. At the most, the vertical and horizontal alignments are slightly different to provide straight line connectivity between clusters.

3.2. Results of Clock Period Evaluation

We ran our experiments for two ASIC technologies, UMC's 0.13μ and 0.18μ Six metal layer process and one FPGA technology, Xilinx XC2V8000-FF1152-5. The Xilinx Device itself has been fabricated in a 0.15μ CMOS process. The results are presented in Tables II–V. Tables II and III show variation of clock period for different interconnect mechanisms with an increase in the number of clusters. Tables IV and V, show variation of interconnect area as a percentage of the total chip area for different interconnect mechanisms with an increase in number of clusters.



(a) Four Cluster Configuration **(b)** Six Cluster Configuration

Fig. 6. Floorplans for RF-to-RF architectures.

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

$N_{Clusters}$	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 III. Clock Period (ns) for UMC 0.18μ ASIC Tech

$N_{Clusters}$	WA.1	RA.1	WA.2	RA.2	WR.2	RF
2	1.161	1.123	1.174	1.192	1.153	1.246
4	1.375	1.243	1.404	1.361	1.215	2.103
6	1.367	1.294	1.522	1.508	1.401	3.856
8	1.459	1.280	1.455	1.416	1.348	4.993
10	1.501	1.395	1.480	1.475	1.407	6.220

From the tables we conclude the following:

1. The increase in clock period for RF-to-RF architecture, with global buses is very high as soon as the number of clusters increases beyond two.

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

$N_{Clusters}$	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 V. Interconnect Area (as % of Chip Area) for UMC 0.18μ ASIC Tech

$N_{Clusters}$	WA.1	RA.1	WA.2	RA.2	WR.2	RF
2	37.01	37.05	37.11	36.89	37.01	37.01
4	34.92	35.00	35.02	34.90	34.95	34.97
6	34.21	34.17	34.26	34.13	34.19	34.17
8	33.60	33.54	33.63	33.53	33.59	33.63
10	33.55	33.56	33.64	33.55	33.53	33.62

2. For other interconnect architectures, the variation across different interconnect mechanisms is negligible.
3. There is a small increase in clock period with an increase in number of clusters for interconnect mechanisms other than *RF-to-RF*.
4. Bidirectional connectivity architectures, *WA.2* and *RA.2*, lose out somewhat in terms of clock period as compared to the unidirectional connectivity architectures and this impact is more in smaller cluster configurations.

To analyze why this is so, we traced the timing paths as reported after STA. For two cluster configuration the critical path (s) never run through the inter-cluster interconnect. This is understandable as in all the cases these interconnects can be efficiently laid out. Even in the *RF-to-RF* type inter-cluster interconnect reduces to connectivity among adjacent clusters. As shown in Fig. 6 this has been laid out very efficiently. The critical path is through the centralized part of microarchitecture, wires which carry condition code to the centralized control register. For higher cluster configuration (beyond 2), the critical path for *RF-to-RF* architectures is through the interconnect bus. As a consequence this increases monotonically with the increase in number of clusters. However, for other interconnect mechanisms the critical path is still through the condition code setting mechanism.

Tables IV and V, show the interconnect as percentage of chip area for two ASIC technologies. A few conclusions are immediately clear:

1. While the interconnect lengths do vary in case of unidirectional and bidirectional architectures, this increase is in proportion to the increase in logic area.
2. The variation in interconnect area even with an increase in number of clusters is not much. This is due to a corresponding increase in the logic area. Basically the interconnect area per cluster is more or less constant.

The following sections use the results obtained above to arrive at realistic bus latencies for multi-cycle bus configurations. Cumulative results are presented in Section 5.

4. EVALUATING EFFECT ON ILP FOR BUS BASED INTERCONNECTS

We use the methodology as discussed in Ref. 15 for evaluating effect on ILP. Herein only a summary of the salient features of this methodology are presented. Figure 7 shows the overall design space exploration methodology. The Trimaran[20] system is used to obtain an instruction trace of the whole application from which a Data Flow Graph (DFG) is extracted for each of the functions. Trimaran also performs a number of ILP enhancing transformations. 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.[21,22] However, this makes the problem intractable for large graphs. We thus divide the problem as follows: first operation to cluster binding is done followed by FU to cluster binding. Since during clustering, the partial schedules are calculated (explained in detail later), the typical phase coupling problem[21,22] 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

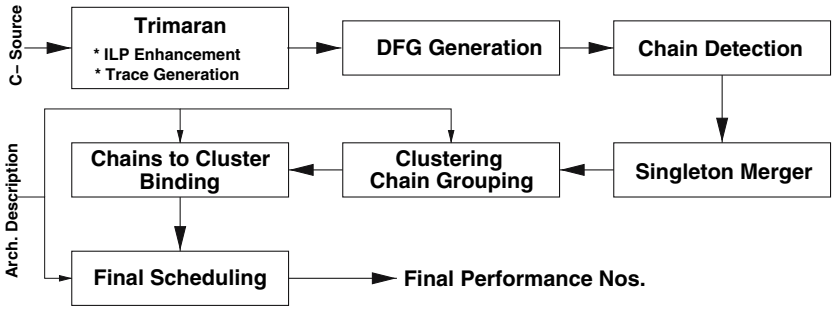


Fig. 7. DSE framework.

Predicate registers. Trimaran also performs a number of ILP enhancing transformations, 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. This is similar to the one discussed in Ref. 23.

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 do not have any source or destinations. Nodes, which have no destinations (sources) are merged with the shortest chain of their sources (destinations).

Clustering: During the clustering phase, 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 Fig. 1(f), 1(g). While in the former, data transfer from any cluster to any other cluster takes same number of cycles, in the latter this varies. During clustering it is assumed that finally the communication would happen through the best possible inter-cluster interconnects. Thus, the schedule length arrived at this stage effectively represents an upper bound on the actual schedule. The clustering algorithm does a pair-wise merger of each of the chains and reschedules the entire graph. A small number of pairs are selected for

merger and this process is repeated till the number of groups is reduced to the number of clusters.

Binding: During the binding phase, groups of chains are bound to individual clusters. Although the value of n_{clusters} is quite small (not more than 16 in our case), still the number of possible bindings is quite large. This effectively rules out any exhaustive exploration of the design space. We recognize that not all edges which denote inter-cluster data transfer as same i.e. some are more critical than the others. A weight is assigned to each of the edges based on mobilities and distance-from-sink node. A greedy assignment is in turn done to assign groups with highest cumulative edge weight amongst them to the closest clusters. Finally a local search is carried around this initial assignment by swapping groups in adjacent clusters.

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.[23–25] 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 is that the propagation algorithm tries to move data to clusters using the bus paths by scheduling transfer operations.

5. CUMULATIVE EXPERIMENTAL RESULTS AND OBSERVATIONS

The main criteria governing the choice of benchmarks was relevance. The primary benchmark sources were MediaBench I[26] and proposed Mediabench II.[27] A second smaller set of benchmarks was chosen from DSPStone.[28] The smaller set of benchmarks are complete programs, i.e., compilable and executable, whereas for the remaining benchmarks, a set of most time consuming functions have been chosen. We have tried to capture around 85% execution time for each of the larger benchmarks, though in case of MPEG Decoder this has not been possible due to unavailability of a small *critical set*. These are listed in Table VI.

Results are shown in Figs. 8–10. In Fig. 8, results are grouped under numbers of the form $\langle xB, yL \rangle$. 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 $\langle 8, 1 \rangle$, $\langle 16, 1 \rangle$ and $\langle 16, 2 \rangle$ 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 3.2, global buses with a

Table VI. Benchmarks used for experiments

Sr. No.	Benchmark Name
DSPStone Kernels	
1.	Matrix Initialization
2.	IDCT
3.	Biquad N Sections
4.	Lattice Transformation
5.	Matrix Multiplication
6.	Insertion Sort
MediaBench I Benchmarks	
7.	JPEG Decoder
8.	JPEG Encoder
9.	MPEG2 Decoder
10.	MPEG2 Encoder
11.	G721 Decoder
12.	G721 Encoder
MediaBench II Benchmarks	
13.	DES Decoder
14.	DES Encoder
15.	JPEG2000 Decoder
16.	JPEG2000 Encoder
17.	GSM Encoder
18.	GSM Decoder

single cycle latency lead to an increase in the overall clock period by up to a factor of five leading to a drastic decrease in overall performance (Fig. 10). Another interesting observation is that the performance varies monotonically with x/y, i.e., the average data transferred per cycle.

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 Fig. 9. Here results are again grouped under various numbers of the form $\langle xB, yS, zL \rangle$. Where, x denotes the number of buses, y the bus stride and z latency of each path between two consecutive appearance of registers in the 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 4, 8 and 16 cluster configurations.

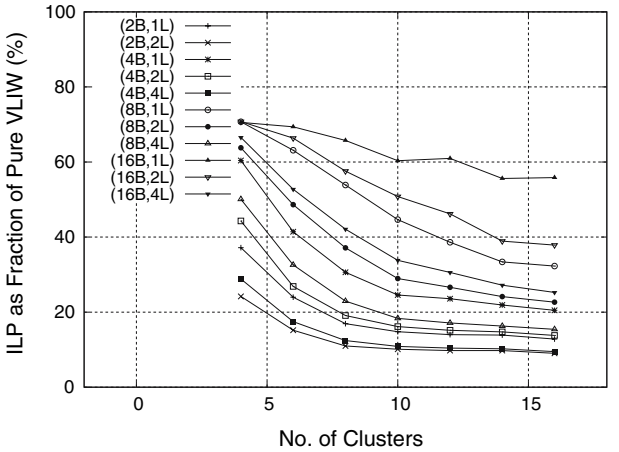


Fig. 8. Average variation in performance for multi-cycle configurations.

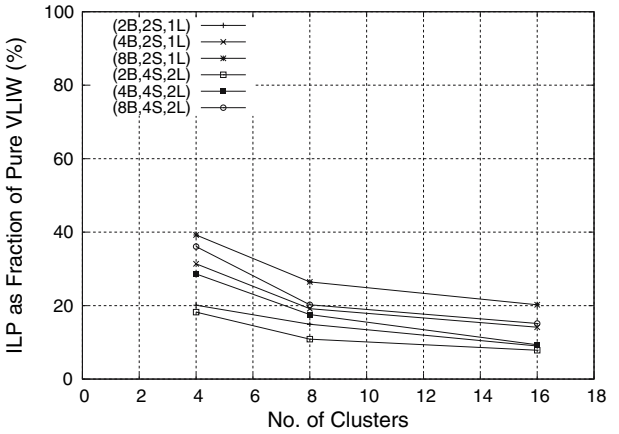


Fig. 9. Average variation in performance for pipelined bus configurations.

From Fig. 9 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

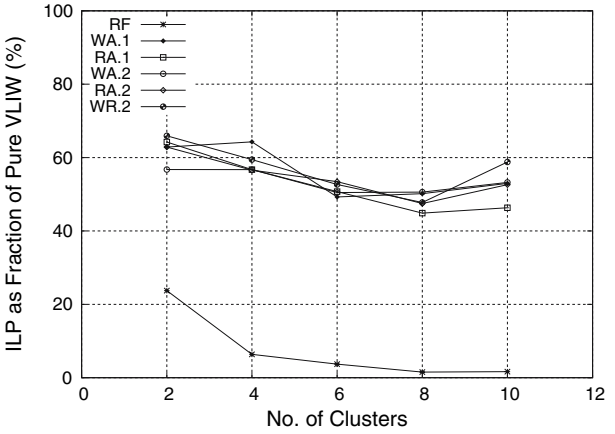


Fig. 10. Effective variation in performance for UMC 0.13μ ASIC technology and multi-cycle configurations.

penalty of one additional cycle latency. Correlating with results of multi-cycle buses in Fig. 8, 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, in any realistic configuration these transfers cannot be completed in a single cycle and thus such an advantage does not exist (see Table II for cycle time counts).

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 the 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 Ref. 15, $Clk\ Period_{WA.1}$ is the clock-period of *Write Across-1* type of architecture and $Clk\ Period_x$ is the clock period of architecture under consideration.

Figure 10 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 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-cluster 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 is established. 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 used bus based architectures. The primary consideration for this seems to be a *simple* compiler. This implies greater focus on compiler technology for clustered VLIW processors. 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. D. Stefanovic and M. Martonosi, Limits and Graph Structure of Available Instruction-Level Parallelism (research note). *Lecture Notes in Computer Science*, Vol. 1900, pp. 1018–1022 (2001).
2. A. Nicolau and J. A. Fisher, Using an Oracle to Measure Potential Parallelism in Single Instruction Stream Programs, *14th Annual Workshop on Microprogramming*, pp. 171–182 (1981).
3. K. Ebcioglu, E. Altman, S. Sathaye, and M. Gschwind, Optimizations and Oracle Parallelism with Dynamic Translation, *Proceedings of the 32nd Annual International Symposium on Microarchitecture*, pp. 284–295. ACM Press (November 1999).
4. J. Fritts and W. Wolf, Evaluation of Static and Dynamic Scheduling for Media Processors. *Second Workshop on Media Processors and DSPs in conjunction with 33rd Annual International Symposium on Microarchitecture*, ACM Press (2000).
5. J. Huang and D. J. Lilja, *Improving Instruction-Level Parallelism by Exploiting Global Value Locality*, Technical Report HPPC-98-12, High-Performance Parallel Computing Research Group, Univ. of Minnesota (1998).
6. D. Talla, L. John, V. Lapinskii, and B. Evans. Evaluating Signal Processing and Multimedia Applications on SIMD, VLIW and Superscalar Architectures. *International Conference on Computer Design (ICCD)*, pp. 163–174 (September 2000).
7. Y. N. Patt, S. J. Patel, M. Evers, D. H. Friendly, and J. Stark, One Billion Transistors One Uniprocessor One Chip, *IEEE Computer* (1997).
8. H. Liao and A. Wolfe, Available Parallelism in Video Applications, *Annual International Symposium on Microarchitecture*, pp. 321–329 (1997).
9. J. Fritts, Z. Wu, and W. Wolf, Parallel Media Processors for the Billion-Transistor Era. *International Conference on Parallel Processing*, pp. 354–362 (1999).

10. D. Burger and J. R. Goodman, Billion-Transistor Architectures: There and Back Again, *IEEE Comput.*, **37**:22–28 (March 2004).
11. P. Glaskowsky, MAP1000 unfolds at Equator, *Microprocessor Report*, 12(16) (December 1998).
12. P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. (Mark Owen) Homewood. Lx: A Technology Platform for Customizable VLIW Embedded Processing, *International Symposium on Computer Architecture (ISCA'2000)*, ACM Press (2000).
13. S. Rixner, W. J. Dally, B. Khailany, P. R. Mattson, U. J. Kapasi, and J. D. Owens. Register Organization for Media Processing, *Proceedings of 6th International Symposium on High Performance Computer Architecture*, pp. 375–386 (May 2000).
14. A. Terechko, E. Le Thenaff, M. Garg, J. van Eijndhoven, and H. Corporaal, Inter-Cluster Communication Models for Clustered VLIW Processors. *9th International Symposium on High Performance Computer Architecture, Anaheim, California*, pp. 298–309 (February 2003).
15. A. Gangwar, M. Balakrishnan, and A. Kumar, Impact of Inter-cluster Communication Mechanisms on ILP in Clustered VLIW Architectures, Second Workshop on Application Specific Processors (WASP-2), in conjunction with 36th Annual International Symposium on Microarchitecture (December 2003).
16. E. Gibert, J. Sanchez, and A. Gonzalez, Flexible Compiler-Managed L0 Buffers for Clustered VLIW Processors, *36th Annual International Symposium on Microarchitecture*, p. 315 (December 2003).
17. J. Gaisler, *LEON: A Sparc V8 Compliant Soft Uniprocessor Core*, http://www.gaisler.com/leon.html.
18. J. H. Tseng and K. Asanovi, Banked Multiported Register Files for High-frequency Superscalar Microprocessors. *International Symposium on Computer Architecture (ISCA-2003)*, pp. 62–71 (June 2003).
19. J.-L. Cruz, A. Gonzalez, and M. Valero, Multiple-banked register file architectures, *International Symposium on Computer Architecture (ISCA-2000)* (2000).
20. Trimaran Consortium, *The Trimaran Compiler Infrastructure, http://www.trimaran.org* (1998).
21. K. Kailas, K. Ebcioglu, and A. K. Agrawala, CARS: A New Code Generation Framework for Clustered ILP Processors. *HPCA*, pp. 133–144 (2001).
22. E. Ozer, S. Banerjia, and T. M. Conte, Unified Assign and Schedule: A New Approach to Scheduling for Clustered Register File Microarchitectures, *International Symposium on Microarchitecture*, pp. 308–315 (1998).
23. V. Lapinskii, M. F. Jacome, and Gustavo de Veciana, High Quality Operation Binding for Clustered VLIW Datapaths. In *IEEE/ACM Design Automation Conference (DAC'2001)* (June 2001).
24. G. Desoli, *Instruction Assignment for Clustered VLIW DSP Compilers: A New Approach*, Technical Report HPL-98-13, Hewlett-Packard Laboratories (February 1998).
25. J. Sanchez and A. Gonzalez, Instruction Scheduling for Clustered VLIW Architectures. *International Symposium on System Synthesis (ISSS'2000)* (2000).
26. C. Lee, M. Potkonjak, and W. H. Mangione-Smith, Mediabench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems, *International Symposium on Microarchitecture*, pp. 330–335 (1997).
27. J. Fritts and B. Mangione-Smith, MediaBench II – Technology, Status, and Cooperation. *Workshop on Media and Stream Processors, Istanbul, Turkey* (November 2002).
28. V. Zivojinovic, J. M. Velarde, C. Schlager, and H. Meyr, DSPStone – A DSP-Oriented Benchmarking Methodology, *International Conference on Signal Processing Application Technology, Dallas, TX*, pp. 715–720 (October 1994).