Inter-cluster Communication Models for Clustered VLIW Processors

Andrei Terechko[1], Erwan Le Thenaff [1], Manish Garg[1], Jos van Eijndhoven[1], Henk Corporaal[2]
[1] - Philips Research, Eindhoven, The Netherlands
[2] - IMEC, Leuven, Belgium; TUE, Eindhoven, The Netherlands
andrei.terechko@philips.com

Abstract

Clustering is a well-known technique to improve the implementation of single register file VLIW processors. Many previous studies in clustering adhere to an inter-cluster communication means in the form of copy operations. This paper, however, identifies and evaluates five different inter-cluster communication models, including copy operations, dedicated issue slots, extended operands, extended results, *and* broadcasting. *Our study reveals that these models have a major impact on performance and implementation of the clustered VLIW. We found that copy operations executed in regular VLIW issue slots significantly constrain the scheduling freedom of regular operations. For example, in the dense code for our four cluster machine the total cycle count overhead reached 46.8% with respect to the unicluster architecture, 56% of which are caused by the copy operation constraint. Therefore, we propose to use other models (e.g. extended results or broadcasting), which deliver higher performance than the copy operation model at the same hardware cost.*

1 Introduction

Convergence of different media in modern consumer electronic devices demands high performance at low cost. Moreover, keeping pace with constantly emerging content standards and dynamic reconfiguration of these devices requires flexibility of these products. To satisfy these market demands, media processors capable of handling multiple media streams in software have been devised. Media processors are used, for example, in high-definition digital TV, videophones, and 3D video games. These applications require thousands lines of high-level language code, characterized by high Instruction-Level Parallelism (ILP) and numerous operations on low-precision data of 8 and 16 bits. To exploit these features, media processors extensively use parallel hardware, for example, in the VLIW and/or SIMD fashion. Obviously, efficient exploitation of these parallel resources and interactive design of large software demand a fast parallelizing compiler.

Many existing media processor cores have a VLIW architecture [1][7][9]. The classical VLIW data-path contains a number of parallel function units (FUs), a multi-ported register file (RF) and a bypass network, see Figure 1. The single uniform RF simplifies code compilation for the processor, while the bypass network enables fast forwarding of the produced results to the operations in the earlier pipeline stages. However, the large *multi-ported register file* and *bypass network* hamper scalability of the processor, which only aggravates with advance of VLSI technologies [15][5].

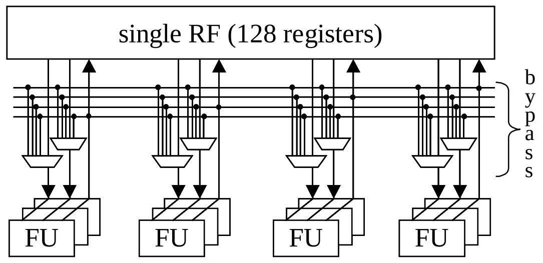

Figure 1. Unicluster architecture

This paper explores clustered data-paths of VLIW architectures, which resolve these two bottlenecks. In a clustered VLIW, see Figure 2, the RF is split in several RFs with fewer registers and ports.

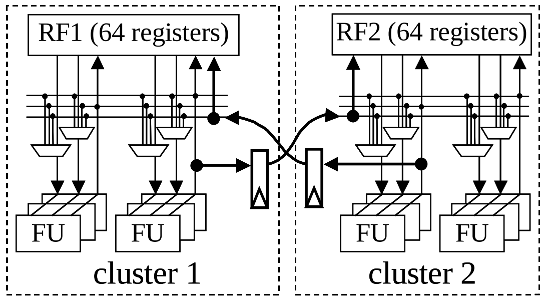

Figure 2. Two-cluster architecture

The partitioned RFs have lower access time, occupy smaller area and dissipate less power [2][3]. However, the compiler for such architectures has to schedule operations addressing multiple register files. The bypass network of a clustered VLIW is partitioned too. Therefore, the number of inputs of the multiplexer network decreases, and the wiring gets shorter.

Currently, there exist diverse inter-cluster communication models [6][7][8][10][16][17]. For example, we can differentiate between *fully* and *partially connected networks* of VLIW clusters. In the fully connected network [7][6] each cluster has a direct connection to all others. Although partial interconnects promise better scalability, the instruction scheduler for such architectures has to deal with a complex problem of avoiding deadlock (when a copy path can not be scheduled) [16]. This paper focuses only on fully connected deadlock-free networks. We also discriminate two inter-cluster connectivity types: the *point-to-point* network presented in Figure 2 and the *bus-based* interconnect presented in Figure 3. Although we concentrate on the point-to-point type, our results are also applicable to the bus-based inter-cluster communication.

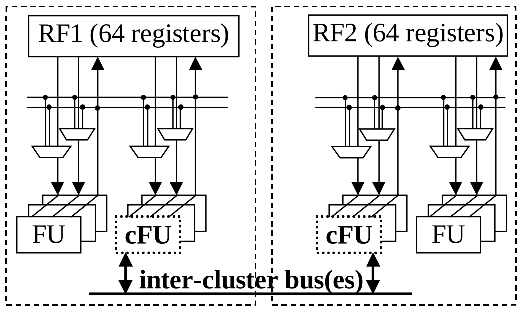

Figure 3. Bus-based two-cluster architecture

In this study we address the taxonomy of inter-cluster communication options and evaluate them based on the cycle count and implementation complexity. The remainder of the paper is organized as follows. First, we define five inter-cluster communication models in Section 2 and analyze them from the implementation point of view. Second, the composition of the cycle count overhead is evaluated for all the models in Section 3. Third, we compare our research to other state-of-the-art studies and, finally, sum up the comparison of the models.

2 Inter-cluster communication models

The inter-cluster data transports have to satisfy constraints of the implementation of a clustered VLIW. In the VLIW tradition, mapping of operations to time slots and function units is visible in the code. Hence, the number of time slots between data-dependent operations scheduled in different clusters must increase by the latency of inter-cluster data transfers. Moreover, the number of inter-cluster transports per instruction should not exceed the inter-cluster bandwidth. These hardware restrictions require explicit specification of the inter-cluster communication in the VLIW code.

There exist numerous ways to add inter-cluster communication (ICC) in the instruction set architecture (ISA). As this paper shows, the ICC model to a large extent determines the cycle count overhead, the code size, and the implementation complexity of the clustered processor. In this section we define and analyze five models of ICC in terms of implementation complexity. Table 1 summarizes the complexity characteristics of the considered models.

Each description of the ICC model is accompanied by a simple code example to show inter-cluster transport in this model. Below is an example code showing two instructions of a four issue slot unclustered VLIW. Semicolons separate VLIW instructions and symbol | separates operations within an instruction. Symbol * designates an operation that is irrelevant for the example. Commas are used to divide the operands and results, and symbol → separates operands from the result. Indices in square brackets identify the cluster. In the examples for the ICC models left-aligned and right-aligned sequences of operations belong to different clusters. The latency of the inter-cluster transfers in the examples is one cycle.

```
op1 r1,r2→r3 | * | * | * ;
* | * | * | op2 r3,r4→r5;
```

2.1 Copy operations

Inter-cluster communication in this model is specified as copy operations in regular VLIW issue slots. In the operand read stage of a copy operation the value is read from the local RF, passed through the bypass network, and clocked in the inter-cluster pipeline register, see Figure 4.

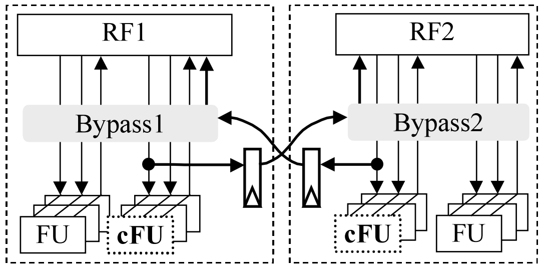

Figure 4. Copy operations

In the next cycle in the execute stage of the copy, the value is sent to the other cluster and fed into the remote bypass. This model, evidently, requires only one extra write port on the RFs per inter-cluster path and has a rather simple bypass network compared to other models, see columns 2,3, and 4 in Table 1.

This encoding of ICC implies that in a number of VLIW instructions some issue slots will be occupied with inter-cluster copy operations. The copy operations will consequently block scheduling of regular operations, which evidently increases the schedule length. On the other hand, this model does not expand the VLIW instruction, and keeps the instruction decoder simple. However, in the scheduled code there will be extra operations – copies enlarging the code size.

An example code with inter-cluster transport by means of copy operations is presented below. The inter-cluster data exchange is carried out solely by copy operations. All other operations access only their local RFs.

```
op1 r1,r2→r3 | * |                         * | * ;
* | copy r3→r1[2] |                        * | * ;
* | * |                         * | op2 r1,r2→r3;
```

A variant of this ICC model is used in the ISA of STMicroelectronics and Hewlett Packard Lx [8]. In fact, Lx requires two copy operations per inter-cluster transfer: send and receive. The send initiates the data transfer in the source cluster and the receive gets the data in the destination cluster.

2.2 Dedicated issue slots

In this model inter-cluster communication is executed in extra dedicated issue slots of the VLIW instruction, see Figure 5. For example, each processing element (cluster) of BOPS's ManArray [6] has a dedicated issue slot to control the cluster switch that exchanges data among the processing elements. Inter-cluster transport in the dedicated issue slot model can take place in any VLIW instruction between the producer and consumer operations without blocking regular operations. In fact, this model provides the highest operation scheduling freedom among the considered models. Although this model seems similar to the copy operation model, the dedicated slots have rather different performance and implementation characteristics.

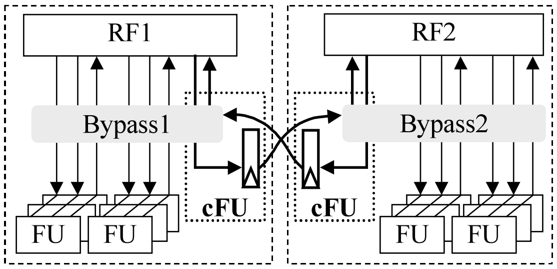

Figure 5. Dedicated issue slots

Implementation of this model is rather expensive. Extra dedicated issue slots lead to expansion of the VLIW instruction, complicating the instruction decoder. Moreover, this ICC model needs two extra RF ports per dedicated slot and the number of multiplexers in the bypass network is comparatively high, see Table 1. Below is the code for the machine shown in Figure 5 with two dedicated issue slots for inter-cluster transport in slots 3 and 4. In fact, the two extra slots make the total slot count equal to six. The inter-cluster transfer of r3 in cluster 1 to r1 in cluster 2 takes place in slot 3 in the second instruction.

```
op1 r1,r2→r3 | * | * |                     * | * | * ;
* | * | r3→r1[2] |                         * | * | * ;
* | * | * |                   * | * | op2 r1,r2→r3;
```

2.3 Extended operands

The source operands in this ICC model are extended with cluster identification. For example, the Texas Instruments VelociTI architecture [7] extends some of the *operands* with cluster id fields. These fields specify the RF where the values should be read from. VelociTI restricts the inter-cluster bandwidth to two inter-cluster reads per instruction. This model allows using a value from a remote RF without storing it in the local RF, see Figure 6, which evidently lessens the register pressure. On the other hand, since the transferred value is immediately consumed by the operation without being stored in the local RF, "reuse" of the copied value is rather hard. Note that this model has more multiplexers in the bypass than the unicluster.

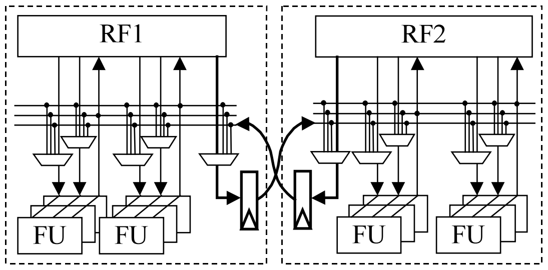

Figure 6. Extended operands

The code below illustrates the extended operand model. The first argument of the operations is extended with specification of the RF. op2 consumes the result of op1 without storing it in RF2. A downside of this ICC model is that the hardware should detect and initiate the inter-cluster transfer rather early in the pipeline, which may be difficult in short pipelines. Moreover, the cluster id extension is always fixed to the VLIW instruction with the corresponding operand, which limits the scheduling freedom of inter-cluster transfers in time.

```
op1 r1[1],r2→r3 | * |                          * | * ;
* | * |                                        * | * ;
* | * |                       * | op2 r3[1],r1→r2;
```

2.4 Extended results

An architecture with extended results is presented in Figure 2. In this model the result of an operation is stored in the cluster specified by the cluster id bits attached to the destination register address. Since the operation's result is not stored locally, this model implements inter-cluster moves rather than copies. In the code example below the result of op1 is moved to register r1 in cluster 2 without being stored in cluster 1.

```
op1 r1,r2→r1[2] | * |                              * | * ;
* | * |                                            * | * ;
* | * |                              * | op2 r1,r2→r3[2];
```

If the cluster id *and* an extra destination field were attached to the results, the architecture would implement multicast – writing the same result to a number of clusters as the sendb operation from the CRB scheme [18]. In the code below operation op1 writes its result to the local register r1 and to the register r1 in cluster 2. The latency of transporting the result to the cluster executing op2 is accounted for by delaying op2 till the third instruction. We used this variant of the extended results model in our performance evaluation in Section 3.

```
op1 r1,r2→r1,r1[2] | * |                           * | * ;
* | * |                                            * | * ;
* | * |                                 * | op2 r1,r2→r3;
```

2.5 Broadcasting

This model can use shared register addresses to communicate between clusters. The registers corresponding to the shared addresses can be both read and written in all clusters. This naturally suggests a shared resource, which contradicts to our notion of clustering. However, for example, the Sun MAJC architecture [9] avoids the shared resource by replicating the 'shared registers' in all clusters [10]. The contents of the replicated registers are kept synchronized, see Figure 7.

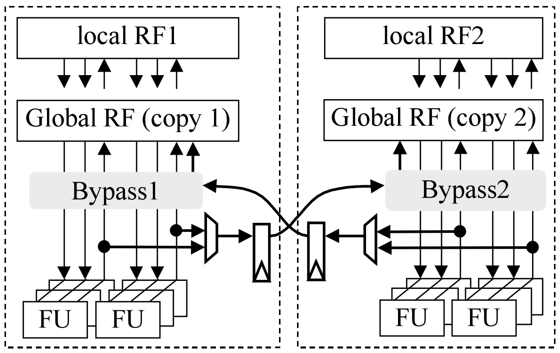

Figure 7. Shared register addresses

The FUs always read the 'shared registers' from the local copy, whereas the writes to the 'shared registers' are broadcasted to all replicas. Consequently, all clusters receive the values written to the shared RF, but not at the same time. A remote cluster can only read the broadcasted value from its copy of the shared registers after the delay of the inter-cluster transfer. Besides the 'shared registers', this model allows local RFs that are accessible only within one cluster. The FUs and the local RF of a cluster are fully connected.

From the implementation point of view, replication of the registers costs significant area. Moreover, the inter-cluster write bandwidth, and, consequently, the number of RF ports is rather high in this model. To lessen the number of write ports on the replicated RFs we can restrict the number of writes to the 'shared registers' per instruction. In Figure 7, for example, only one operation per cluster can write to a 'shared register'. Nevertheless, broadcasting to all clusters is never power-efficient.

Using the operand and result fields, the inter-cluster transport can be easily encoded by splitting the register address space into local and shared registers. For example, the four-cluster Sun MAJC-5200 has a total of 224 logical registers using 7 bit operands and results. The 7-bit register address space is partitioned into 96 shared registers accessible by all clusters and 32 registers local for each cluster.

In the example code below the register address space is split like in MAJC-5200. Shared register r127 is used to communicate the result of op1 to op2. The result of op1 arrives in the cluster of op2 one cycle later than in the cluster of op1, see Figure 7. Op2 is, consequently, delayed till the third instruction.

```
op1 r1,r2→r127 | * |                               * | * ;
* | * |                                            * | * ;
* | * |                              * | op2 r127,r1→r2;
```

Another encoding alternative of the broadcasting ICC model is to specify with an additional bit whether an operation stores the result locally or broadcasts it to all clusters. In the code example below operation op1 broadcasts its result to all clusters, which is specified by *b* after destination register r3. op2, on the contrary, stores the result only locally in register r2. Obviously, register r3 must be made free in *all clusters* from the cycle with op1.

```
op1 r1,r2→r3b | * |                                * | * ;
* | * |                                            * | * ;
* | * |                                 * | op2 r3,r1→r2;
```

Note that this encoding does not imply replication of the registers. We used this variant of the broadcasting model for our performance evaluation in Section 3.

Table 1. Complexity characteristics of the inter-cluster communication models

Inter-cluster communication model	**extra #read ports**	**extra #write ports**	**bypass (#muxes: #inputs)** [*]	**VLIW instruction size increase (bits)**	**total number of registers**
Unicluster	–	–	8:5	0	$2^{N_{bits}}$
Copy operations	0	$(C-1)B_w$	8:5	0	$C \bullet 2^{N_{bits}}$
Dedicated slots	$(C-1)B_w$	$(C-1)B_w$	10:4	$B_w \bullet [\log_2(C-1) + 2\bullet N_{bits}]$	$C \bullet 2^{N_{bits}}$
Extended operands	$(C-1)B_r$	0	10:4	$B_r \bullet [\log_2(C-1) + N_{bits}]$	$C \bullet 2^{N_{bits}}$
Extended results	0	$(C-1)B_w$	8:4	$B_w \bullet [\log_2(C-1) + N_{bits}]$	$C \bullet 2^{N_{bits}}$
Broadcasting	0	$(C-1)B_w$	8:4	$B_w \bullet [\log_2(C-1) + N_{bits}]$	$C \bullet 2^{N_{bits}}$

C – the number of clusters
N_{bits} – the number of bits per operand/result field
N_{glregs} – the number of registers globally accessible
B_w, B_r – the number of processor words a cluster can write to or read from the other clusters (bandwidth)
[*] – for the 4 issue slot two-cluster machine with the inter-cluster bandwidth of 1 inter-cluster transfers per cluster per instruction

3 Composition of the cycle count overhead

Although clustering enables higher clock speed, it also incurs overhead in the number of execution cycles. This section evaluates the total cycle count overhead for two-cluster and four-cluster 8-issue slot machines. On top of that, we identify different factors of this overhead and quantify their separate contributions.

We distinguish the following factors in the cycle count overhead:

1. *Extra latency of inter-cluster data transfers*. According to our notion of clustered VLIWs, obtaining a value from a remote cluster costs one or more extra cycles.
2. *Limited inter-cluster bandwidth*. In the clustered VLIW the reduced number of RF ports limits the inter-cluster bandwidth. Obviously, the conflicts on the RF ports will stretch the schedules.
3. *Higher register pressure*. In the schedules for clustered VLIWs some data gets replicated in multiple RFs. This increases the register pressure, which may lead to spill-code or serialization of the register live ranges and, hence, extra execution cycles.
4. *Inter-cluster communication model constraints.* The encoding of inter-cluster communication in the ISA may constrain operation scheduling. Most of the five ICC models described in Section 2 impose certain scheduling constraints. For example, the copy operations occupy VLIW issue slots, which become unavailable for regular operations. This effect extends the schedules. Yet another example of an ICC model constraint is present in the extended operand model, in which inter-cluster transfers are coupled to the VLIW instruction consuming the value to be transferred. ICC models free of these constraints (e.g. dedicated issue slots) avoid the associated cycle count overhead completely.
5. *Extra cache stall cycles due to code size overhead and higher register pressure*. The code for clustered VLIWs specifies ICC. This may result in the code size overhead, which can lead to extra instruction cache stall cycles. Moreover, the increased requirements on the registers may cause spilling, and, hence, extra data cache stall cycles.

The contributions of these factors to the cycle count overhead are not always independent from each other. For example, the extra inter-cluster latency increases the live ranges of the local variables and, consequently, increases the register pressure. Therefore, the extra register pressure overhead is influenced by the increased inter-cluster latency. Another example is the dependence of the extra data cache stall cycles on the increased RF pressure. The higher the RF pressure, the more data cache stalls occur due to spilling.

Due to the interdependent nature of the factors, the sum of the individual contributions measured separately will not be equal to the total cycle count overhead. Therefore, we decided to evaluate the contributions incrementally, see Figure 8. First, the baseline cycle count was measured on the 8-slot unicluster machine. Second, the cycle count overhead due to extra inter-cluster latency was quantified on the specially configured architecture *latency*, which features only one factor – extra latency. This portion is caused by the technological trend increasing wire delay and can hardly be avoided. Second, we evaluated the cycle count overhead caused by extra latency and limited inter-cluster bandwidth using architecture *bandwidth*. The bandwidth was decreased to 1 inter-cluster transfer per cluster per VLIW instruction, which significantly contributes to the overall cycle count overhead. Obviously, the added contribution of the limited bandwidth can be calculated by subtracting the overhead of architecture *latency* from architecture *bandwidth*. Third, architecture *RFpressure* with extra latency, limited bandwidth and 128 registers gives us the extra overhead caused by the increased RF pressure. And finally, we evaluated the added contributions of the five ICC models.

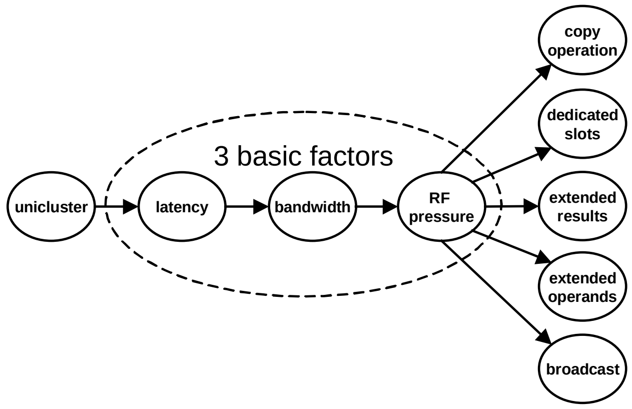

Figure 8. Incremental evaluation of the factors

Clustering affects the code size with respect to the unicluster, and, consequently, the instruction cache stall cycles. Smaller partitioned register files obviously require fewer bits to encode operands and results of the operation. However, the instruction of a clustered VLIW must include specification of the inter-cluster communication (e.g. in the form of the copy operation). Furthermore, longer schedules for the clustered VLIW require more instructions to encode a program. This also expands the code. In our experiments we found that these effects mostly compensate each other, which leads to the static code size deviation within +/-5% with respect to the unicluster. Having discovered the small variation of the code size, we decided to neglect the instruction cache effects.

The design space of data memory hierarchies for clustered processors is huge. One of the major decisions is whether to distribute the caches among the clusters (and somehow maintain cache coherence) or have a unified cache for all clusters. If the caches are distributed, a decision should be taken whether the cluster blocks on a miss of the corresponding cache or not, etc. Having considered this variety of trade-offs, we conclude that evaluation of data memory for clustered processors deserves a separate study, and, therefore, we assume ideal memory in the remainder of this paper.

3.1 Benchmarks

We evaluated the cycle count overhead on multimedia C benchmarks optimized for the TriMedia VLIW with a rich SIMD operation set [1][11]. To cover a significant area of the application domain, the benchmarks were chosen from different categories, see Table 2.

Table 2. Optimized benchmarks

Benchmark	Category	ILP	lines of C code
Viterbi decoder	Data communication	4.1	140
Peaking	Video processing	2.3	2544
Median filter	Video processing	3.3	588
MPEG II encoder	Video coding	4.5	12649
Layered Natural Motion	Video processing	3.2	16025
DVC decoder	Video coding	4.3	4491
Renderer	3D graphics	2.8	7159
Transform	3D graphics	3.3	1399

The benchmarks underwent optimizing source-to-source transformations. Furthermore, the code was enhanced with 64-bit SIMD intrinsics, cache prefetch operations, loop unrolling, software pipelining, function inlining, restricted pointers, etc. The presented ILP rates were measured dynamically in the simulator for the unicluster 8-issue slot machine. Note that SIMD operations are equivalent to 8-24 RISC operations. This makes the actual ILP for the optimized code higher than presented in Table 2. In total, the optimization of these applications brought 10x-40x speedups with respect to the initial source code.

3.2 Instruction scheduler

We measured the cycle count overhead using the state-of-the-art TriMedia C/C++ compiler TCS2.1. For our study we built an instruction scheduler to schedule the inter-cluster communication in all five ICC models. The scheduling unit is the guarded decision tree of basic blocks [23][24][11]. The guarded decision tree is an acyclic control flow graph without join point, which is a more general case than superblocks [21] or traces [22]. Inspired by [18], [19], and [20], we integrated cluster assignment, instruction scheduling, and register allocation in a single phase. Thanks to the integration of the phases our algorithm avoids the well-known problem of phase coupling [18][12] and yields a significantly denser code [19].

The core of our scheduling algorithm is outlined in Figure 9. Register live range information is kept in register bit vectors per VLIW instruction. The floater operations are used to shorten register live ranges and are described in [11]. To efficiently fill in the 3 branch delay slots of the TriMedia, the scheduler uses backtracking. The jumps are scheduled optimistically, and if the scheduler does not manage to fit the remaining operations in the branch delay slots, it backtracks. Note that spill and restore code is inserted on the fly during operation scheduling. Remarkably, spill and restore operations may trigger scheduling of extra copy operations.

```
schedule_operation(oper, tree) {
   cluster_list = build_ordered_cluster_list(oper, tree);
   for (instr = i_min; instr->cycle <= i_max; instr = instr->next) {
      for (cl = cluster_list->head; cl; cl = cl->next) {
         if (!assign_oper_to_slot(oper, cl, instr))
            continue;
         if (!schedule_floaters(oper, cl, instr) {
            unschedule_floaters(oper);
            continue; }
         if (!schedule_copies(oper, cl, instr)) {
            unschedule_copies(oper);
            unschedule_floaters(oper);
            continue; }
         if (!assign_register(oper, cl, instr))
            if (!schedule_spill_restore(oper, cl, instr)) {
               unschedule_copies(oper);
               unschedule_floaters(oper);
               continue; }
         early_jump = too_optimistic_jumps(tree);
         if (early_jump)
            /* unschedule & restart scheduling from early_jump */
            backtrack (early_jump, tree);
         return TRUE;   /* successfully scheduled operation oper */
      }
   }
   return FALSE; /* failed to schedule operation oper */
}
```

Figure 9. Scheduling algorithm

build_ordered_cluster_list(oper, tree) orders cluster assignments in *cluster_list* based on the following cost function:

$$C = c_c N_c + c_{rf} N_{liveregs}/N_{regs} + c_{slots} N_{opers}/N_{slots}, \qquad (1)$$

where C is the cost of the cluster assignment,
N_c – number of copies required,
$N_{liveregs}$ – number of live registers in the cluster,
N_{regs} – total number of registers in the architecture,
N_{opers} – number of operations scheduled in the cluster,
N_{slots} – number of issue slots in the cluster,
c_c, c_{rf}, c_{slots} – term coefficients, $c_c>0$, $c_{rf}>0$, $c_{slots}>0$.

The term coefficients were tuned for optimal performance. Minimization of cluster copies plays the major role in our cluster assignment. The scheduler assigns an operation to the cluster that requires the fewest inter-cluster data transfers by examining predecessors *and* successors of the operation. If we considered only predecessors *p1* and *p2* of operation *o1* from Figure 10, assignments of *o1* to cluster 1 and 2 would seem to need only one copy operation each. However, including successor *s1* into consideration indicates that assignment of *o1* to cluster 1 will require later another copy operation from *p3*. In fact, one can analyze even bigger neighborhoods of the data flow graph around the operation being scheduled to count required copies. However, our experiments showed no substantial benefit from considering larger neighborhoods.

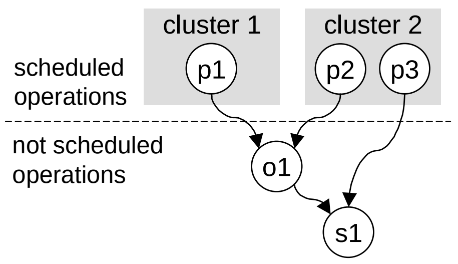

Figure 10. Accounting for future copy operations

As a secondary criterion, the scheduler dynamically balances the pressure on the RFs and the number of operations in the clusters. In particular, the scheduler tends to choose the cluster with the smallest ratio of the scheduled operations to the total number of issue slots in the cluster. Under these heuristics the scheduler's performance is rather high, yielding only 5-10% longer schedules than hand-optimized ones.

Our scheduler performs register allocation within a decision tree of basic blocks. The registers used to transport values between the decision trees (globals) [11] are allocated prior to instruction scheduling. In our experiments these global values were distributed among the clusters in a round-robin fashion. The round-robin distribution proved to be better than assigning all globals to one cluster, since the latter unbalances cluster assignments of operations towards the cluster with globals. Note that for the broadcast model our scheduler broadcasts globals such that they can be read later without penalty in all clusters. Performing such an optimization for the other ICC models may result in many ICC operations at the end of decision trees and, hence, extend the schedule length, especially, when the global variables have to be sent to many clusters. Therefore, the optimization of accesses to globals was implemented only for the broadcast model.

The ICC models supported by the scheduler require various model-specific optimizations. For example, in the extended results model a result of an operation besides being stored in the local RF may also be copied to a remote RF. However, if some consumers of this result reside in the clusters where the result was not copied to, the scheduler has to add and schedule an extra operation that will send the same result to the other cluster(s). Another important model's peculiarity is present in the extended operands. In this model “reuse” of inter-cluster transferred values is quite cumbersome, since the operation being scheduled immediately consumes the sent value as an operand without storing it locally. Therefore, in this model our scheduler does not reuse the sent value. On top of that, if in the extended operand model two operands of the same operation require inter-cluster transfers from different clusters but the inter-cluster bandwidth allows only one transfer, the same technique is used as described above for the extended result model.

3.3 Measuring the composition of the cycle count overhead

This section presents the techniques we used to measure separate contributions of the four major factors (excluding cache effects) to the cycle count overhead for the five ICC models. The baseline unicluster architecture is an 8-issue slot VLIW with the TriMedia operation set. All operations are pipelined and can contain a guard (predicate), two operands and one result. The distribution of the function units among slots was made such that the derived clustered architectures contained equal functionality in all their clusters, see Table 3.

Table 3. Baseline unicluster architecture

Function Units	Issue slots	Latency
alu, shifter	1,2,3,4,5,6,7,8	1
imul, fmul	1,3,5,7	3
load/store	2,4,6,8	3
branch	1,3,5,7	4

The two-cluster architectures used in our experiments have slots 1, 2, 3, and 4 in cluster one and slots 5, 6, 7, and 8 in cluster two. Each of the two clusters possesses 64 registers. The four-cluster architectures have slots 1 and 2 in cluster one, slots 3 and 4 in cluster two, slots 5 and 6 in cluster three, and slots 7 and 8 in cluster four. Each of the four clusters contains 32 registers.

To incrementally evaluate individual contributions of the four factors to the total cycle count overhead, we built specially configured clustered architectures, see Table 4. Execution of a benchmark on a specially configured architecture produces cycle count overhead due to certain factors only, while the other factors are suppressed by unbound resources. For example, the special architecture *latency* features a huge RF, unrestricted inter-cluster bandwidth and no ICC model constraints. Unrestricted inter-cluster bandwidth is implemented by instantiating sufficient extra issue slots with copy function units, which allow each operation to read (with a delay) a guard and operands from a remote cluster. Besides, the dedicated issue slots do not constrain scheduling freedom. So, the only factor contributing to the cycle count overhead of architecture *latency* relative to the unicluster is the inter-cluster transfer's latency of one cycle. The measured individual contributions of the three basic factors are reported in Table 5 for the two cluster and four cluster machines.

Table 4. Specially configured architectures

special architecture	ICC latency (cycles)	ICC bandwidth*	total registers	ICC model
latency	1	unbound	1024	Dedicated slots
bandwidth	1	1	1024	Dedicated slots
RFpressure	1	1	128	Dedicated slots
copy	1	1	128	Copy operations
dedslots	1	1	128	Dedicated slots
extresults	1	1	128	Extended results
extoperands	1	1	128	Extended operands
broadcast	1	1	128	Broadcasting

* - per cluster

The contributions of the five ICC encoding models were evaluated on the special architecture *dedslots*, *copy*, *extoperands*, *extresults*, and *broadcast*. Each of these architectures has the three basic factors and implements a corresponding ICC model. The measured contributions of the model constraints are presented in Table 6 for the two cluster machine and Table 7 for the four cluster machine. In our experiments the *copy* architecture requires one copy operation per inter-cluster transfer. In architecture *extresults* one operation per cluster per VLIW instruction can send a value to a remote cluster, and in architecture *extoperands* one operation per cluster can read from a remote RF. In architecture *broadcast* one operation per cluster per VLIW instruction can broadcast its result to all clusters.

Table 5. Three basic contributions for the two and four cluster machines

	2 cluster 8 issue slot machine			4 cluster 8 issue slot machine		
	latency	bandwidth	RF pressure	latency	bandwidth	RF pressure
viterbi	6.0%	5.8%	0.0%	23.5%	0.0%	0.0%
peaking	6.8%	0.1%	0.0%	11.4%	1.3%	0.0%
median	7.4%	7.1%	0.0%	14.2%	13.4%	0.0%
mpeg2enc	1.2%	5.1%	5.1%	7.6%	7.7%	10.4%
natmot	8.1%	3.3%	0.9%	12.2%	6.4%	4.5%
dvc_dec	3.6%	3.0%	0.1%	7.1%	4.4%	3.2%
renderer	5.3%	0.9%	0.0%	8.8%	3.3%	0.0%
transform	3.9%	7.0%	0.0%	9.1%	6.6%	2.5%
average	**5.3%**	**4.0%**	**0.8%**	**11.7%**	**5.4%**	**2.6%**

Table 6. Added contributions and the total cycle count overhead of the ICC models, 2 clusters

	dedicated slots		copy operations		extend. operand		extended results		broadcast	
	added	total	added	total	added	total	added	total	added	total
viterbi	0.0%	11.8%	11.8%	23.6%	5.9%	17.7%	0.0%	11.8%	-11.7%	0.1%
peaking	0.0%	6.9%	0.2%	7.1%	0.9%	7.7%	0.0%	6.9%	-1.6%	5.3%
median	0.0%	14.4%	16.5%	30.9%	16.7%	31.2%	6.6%	21.1%	-3.8%	10.6%
mpeg2enc	0.0%	11.4%	4.5%	15.8%	0.9%	12.2%	1.4%	12.8%	-6.9%	4.4%
natmot	0.0%	12.2%	2.3%	14.5%	8.8%	21.0%	0.7%	12.9%	-7.1%	5.1%
dvc_dec	0.0%	6.7%	11.0%	17.7%	5.4%	12.1%	2.0%	8.7%	-2.2%	4.6%
renderer	0.0%	6.2%	1.2%	7.4%	2.5%	8.7%	-0.1%	6.1%	-4.7%	1.5%
transform	0.0%	10.9%	1.9%	12.8%	6.0%	16.9%	-0.3%	10.5%	-8.7%	2.2%
average	**0.0%**	**10.1%**	**6.2%**	**16.2%**	**5.9%**	**15.9%**	**1.3%**	**11.4%**	**-5.8%**	**4.2%**

Table 7. Added contributions and the total cycle count overhead of the ICC models, 4 clusters

	dedicated slots		copy operations		extend. operand		extended results		broadcast	
	added	total	added	total	added	total	added	total	added	total
viterbi	0.0%	23.5%	35.1%	58.6%	5.9%	29.4%	5.8%	29.4%	-5.91%	17.6%
peaking	0.0%	12.6%	3.5%	16.1%	3.8%	16.4%	0.0%	12.6%	-7.14%	5.5%
median	0.0%	27.6%	46.5%	74.0%	23.8%	51.4%	3.3%	30.9%	-13.43%	14.1%
mpeg2enc	0.0%	25.7%	27.9%	53.6%	9.4%	35.1%	6.9%	32.6%	-19.41%	6.3%
natmot	0.0%	23.1%	17.9%	41.0%	9.4%	32.5%	1.8%	24.9%	-15.73%	7.4%
dvc_dec	0.0%	14.6%	64.7%	79.4%	14.0%	28.6%	16.0%	30.7%	-6.77%	7.9%
renderer	0.0%	12.2%	12.0%	24.2%	2.7%	14.9%	-0.5%	11.7%	-8.67%	3.5%
transform	0.0%	18.3%	9.0%	27.3%	3.3%	21.6%	1.1%	19.4%	-9.93%	8.4%
average	**0.0%**	**19.7%**	**27.1%**	**46.8%**	**9.0%**	**28.7%**	**4.3%**	**24.0%**	**-10.87%**	**8.8%**

To illustrate how the total cycle count is composed of the individual contributions consider execution of benchmark *mpeg2enc* on the two-cluster machine. Extra latency, limited bandwidth and increased register pressure overheads sum up to 1.2%+5.1%+5.1%=11.4%, see Table 5. Suppose we compile this benchmark for the extended result model, then the model's overhead will be 1.4%, see Table 6. So, the total cycle count overhead comes to 11.4%+1.4%=12.8% relative to the unicluster.

The dedicated slot model yields always 0.0% overhead because it introduces no additional overhead beyond the three basic factors. Therefore, negative numbers for other models merely mean better performance than the dedicated slot model. Remarkably, the scheduler found a better schedule for benchmarks *renderer* and *transform* in the extended result model than in the dedicated slot model. We attribute that to the shortcoming of the cluster assignment that has a relatively narrow view on the data flow graph, considering only direct successors and predecessors of the operation being scheduled.

Note that the benchmarks with high ILP, for example, DVC decoder and Viterbi suffered from the constraints of the copy operation the most. Apparently, occupation of valuable issue slots by copy operations in these benchmarks severely constraints scheduling of dense code.

4 Discussion of the results

Figure 11 and Figure 12 show the cycle count overhead of the considered ICC models averaged among the benchmarks and the contributions of the four factors to the total cycle count overhead.

Our results clearly demonstrate that the encoding of ICC in the ISA strongly influences performance of the clustered VLIW processors. For example, the broadcast model, especially, in the four-cluster architecture, utilizes the inter-cluster bandwidth better than the other models. Indeed, in our four-cluster machine four broadcasts to all RFs may take place, whereas the other models require four times three inter-cluster copies for the same transfer. Moreover, broadcasting can escape high penalty from the copies to and from globals as explained in Section 3.2, which significantly reduces the number of copies at the beginning and the end of decision trees. Evidently, this effect drastically decreases the contributions of the basic factors latency and bandwidth. Therefore, in our experiments the broadcast model outperforms other ICC models, see Figure 11 and Figure 12. Nevertheless, the higher RF pressure of the broadcast model may compromise its high performance by extra data cache stalls due to spilling.

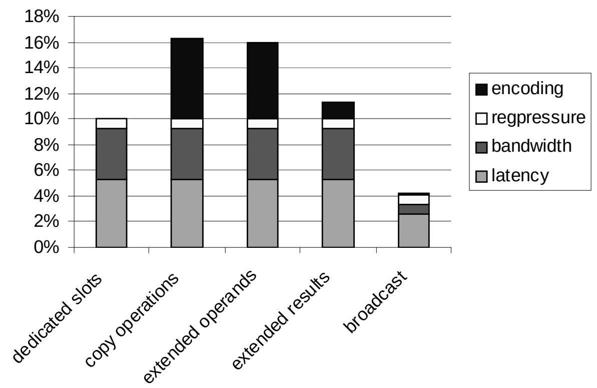

Figure 11. Total cycle overhead, 2 clusters

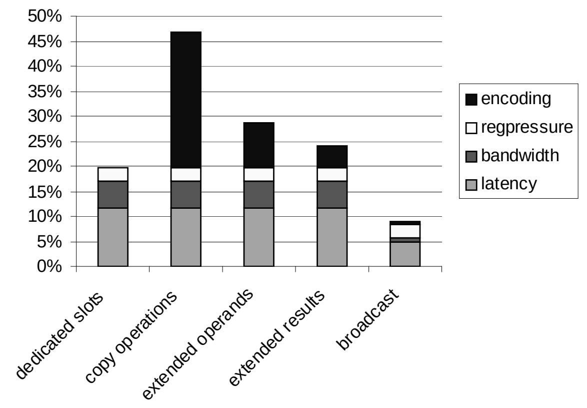

Figure 12. Total cycle overhead, 4 clusters

The copy operation model yields the highest cycle count overhead. In the dense code for our four cluster machine the total cycle count overhead reached 46.8% with respect to the unicluster architecture, 56% of which are caused by the copy operation constraint. Therefore, other ICC models (i.e. the extended results model) outperform the copy operation model at the same hardware cost. Hence, we doubt efficiency of the send-receive model described in Section 2.1 for high ILP code. Interestingly, the extended operands model substantially suffers from absence of copy "reuse", because, evidently, many copies have to be duplicated for different consumers of the same value. Although the dedicated issue slots model has comparatively high implementation complexity, see columns 2, 3, and 4 in Table 1, from a cycle count point of view it performs very well. If the instruction scheduler can not properly handle increased register pressure in the broadcast model, the dedicated slots model may deliver better performance. Note that in the two-cluster machine the extended results model could have comparable performance to broadcasting, if we would implement the optimization for global variables described in Section 3.2. Furthermore, our compiler heuristics for formation of decision trees could be adapted for the non-broadcasting models to decrease the number of globals, which may reduce the performance advantage of the broadcast model.

The contribution of the extra inter-cluster latency in the total cycle count overhead was significant in all our experiments. Therefore, further pipelining the ICC with inter-cluster latency of more than one cycle can become advantageous only in some distant future IC technologies, if the wire delays become totally overwhelming. Moreover, inter-cluster pipelining may be beneficial to exclude completely for processors that run at lower clock speed and need fewer registers than media processors. In all our clustered machines inter-cluster bandwidth was reduced to the minimum of one transfer per cluster per VLIW instruction. Hence, increasing the bandwidth may help reduce the large contribution of the limited bandwidth, unless the consequently increased number of RF ports limits the RF access delay.

5 Related work

Many prior studies in clustered VLIW architectures based their research on one inter-cluster communication model in the form of copy operations. This paper, however, analyzes several inter-cluster communication models in terms of the cycle count and implementation complexity. A significant amount of research was put in improving the scheduling algorithms for clustered VLIWs [12][13][14]. According to our study, though, choosing a proper ICC model can eliminate the constraint of the copy operation ISA, which considerably helps operation scheduling. This in turn can decrease the execution cycle count overhead by up to 56% or more if the application contains higher ILP than our benchmarks.

A similar discrimination and evaluation of the factors of the cycle count overhead was carried out for clustered Transport Triggered Architectures [16]. However, due to differences between the TTA and VLIW architectures they did not measure the ICC model overheads, which according to our research can dominate in VLIWs.

In the ICC model proposed in [17] a small Caching Register Buffer (CRB) is attached to the register file of each cluster to store copies of remote registers. ICC in the CRB scheme is carried out by the special broadcast operation sendb. According to our study this operation would block scheduling of regular operation in dense code just as the copy operation. Of course, the frequency of sendb operations will be much lower than copies, since one sendb operation implements a number of copies. On top of that, as proved by our broadcasting model, broadcasting features better utilization of the inter-cluster bandwidth than other models. Nevertheless, the CRB model decreases the register pressure on the architecture-visible registers, which in our experiments did not dominate the total overhead. Furthermore, implementation of a small inter-RF cache is costly in hardware.

Our measurements of the cycle count overhead were based on the execution cycle count, not on the schedule length like in [13][14]. Furthermore, we experimented with optimized full applications with high ILP rates. We believe that the dense code reflects the preferred high utilization of the machine and, hence, should be used for design space explorations of microprocessors.

6 Conclusions

This paper analyzes five inter-cluster communication models for clustered VLIWs, including *copy operations, dedicated issue slots*, *extended results*, *extended operands*, and *broadcasting*. On top of that, we identify and evaluate different factors of the cycle count overhead of clustered VLIW architectures. This evaluation reveals that encoding inter-cluster communication as copy operations causes up to 56% of the total cycle count overhead in our four cluster 8 issue slot VLIW. Therefore, some models free of the copy operation constraint (e.g. extended results) outperform the copy operation model at the same hardware cost. In our experiments broadcasting outperformed the other models, bringing the cycle count overhead of a two-cluster machine down to 4.2% with respect to the unicluster, due to efficient inter-cluster bandwidth utilization and cluster assignment of global variables.

References

[1] S. Rathnam, G. Slavenburg, “An architectural overview of the programmable multimedia processor, TM-1”, *41st IEEE International Computer Conference*, pp. 319-326, Santa Clara CA, 1996.

[2] S. Rixner, W.J. Dally, et al. “Register organization for media processing”, *6th International Symposium on High-Performance Computer Architecture*, 8-12 January 2000, Toulouse, France, 1999.

[3] V. Zyuban, P.M. Kogge, “The Energy Complexity of Register Files”, *International Symposium on Low-Power Electronics and Design*, pp. 305-310, Monterey, USA, August 1998.

[4] J. Janssen, H. Corporaal, “Partitioned Register Files for TTAs”, *28th Annual International Symposium on Microarchitectures*, Michigan, November 1995.

[5] R. Ho, K. Mai, and M. Horowitz, “The Future of Wires”, *Proceedings of the IEEE*, April 2001.

[6] M. Levy, “ManArray devours DSP code”, *Microprocessor report*, October 2001.

[7] TI TMS320C64xx DSPs. http://www.ti.com.

[8] P. Faraboschi, G. Desoli, et al. “Lx: A technology platform for customizable VLIW embedded processing”, *27th Annual International Symposium on Computer Architecture*, Vancouver Canada, June 2000.

[9] S. Sudharsanan, P. Sriram, et al. “Image And Video Processing Using Majc 5200”, *International Conference on Image Processing*, Vancouver Canada, 2000.

[10] http://www.sun.com/microelectronics/MAJC.

[11] J. Hoogerbrugge, L. Augusteijn, “Instruction scheduling for TriMedia”, *The Journal of Instruction-Level Parallelism*, February 1999.

[12] E. Ozer, S. Banerjia and T. Conte, “Unified assign and schedule: a new approach to scheduling for clustered register file microarchitectures”, *31st Annual International Symposium on Microarchitectures*, pp. 308-315, Dallas Texas, November 1998.

[13] V. S. Lapinskii, M.F. Jacome, and G.A. de Veciana, “High-Quality Operation Binding for Clustered VLIW Datapaths”, *Design and Automation Conference*, Las Vegas USA, June 2001.

[14] P. Mattson, W.J. Dally, et al. "Communication Scheduling", *International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, USA, 2000.

[15] http://public.itrs.net. ITRS roadmap.

[16] S. Roos, H. Corporaal, et al., “Clustering on the Move”, *4th International Conference on Massively Parallel Computing Systems*, Ischia Italy, April 2002.

[17] K. Kalias, M. Franklin, K. Ebcioglu, “A Register File Architecture and Compilation Scheme for Clustered ILP Processors”, *International Conference Euro-Par*, Paderborn, Germany, 27-30 August 2002.

[18] K. Kailas, K. Ebcioglu, et al., “CARS: A New Code Generation Framework for Clustered ILP processors”, *7th International Symposium on High Performance Computer Architecture*, pp. 133-134, Nuevo Leone Mexico, January 2001.

[19] J. Janssen, “*Compiler Strategies for Transport Triggered Architecture*”, PhD thesis, 2001, TU Deflt, The Netherlands.

[20] J. M. Codina, J. Sanchez, et al., “A Unified Modulo Scheduling and Register Allocation Technique for Clustered Processors”, *International Conference on Parallel Architecture and Compilation Techniques (PACT)*, Barcelona, Spain, September 2001.

[21] W.W. Hwu, S.A. Mahlke, et al., “The Superblock: an Effective Technique for VLIW and Superscalar Compilation”, *The Journal of Supercomputing*, vol. 7, pp. 229-249, May 1993.

[22] J.A. Fisher, “Trace Scheduling: a Technique for Global Microcode Compaction“, *IEEE Transactions on Computers*, vol. C-30, pp. 478-490, July 1981.

[23] P. Y. T. Hsu, E. S. Davidson, "Highly Concurrent Scalar Processing," *13th Annual International Symposium on Computer Architecture*, pp. 386-395, Tokyo, Japan, June 1986.

[24] W.A. Havanki, S. Banerjia, T. M. Conte, “Treegion scheduling for wide-issue processors”, *4th International Symposium on High-Performance Computer Architecture*, Las Vegas, USA, February 1998.