Inter-cluster Communication in VLIW Architectures

A. S. TERECHKO
NXP Semiconductors, Eindhoven
and
H. CORPORAAL
Technical University of Eindhoven

The traditional VLIW (very long instruction word) architecture with a single register file does not scale up well to address growing performance demands on embedded media processors. However, splitting a VLIW processor in smaller clusters, which are comprised of function units fully connected to local register files, can significantly improve VLSI implementation characteristics of the processor, such as speed, energy consumption, and area. In our paper we reveal that achieving the best characteristics of a clustered VLIW requires a thorough selection of an *Inter-cluster Communication (ICC) model*, which is the way clustering is exposed in the Instruction Set Architecture. For our study we, first, define a taxonomy of ICC models including *copy operations*, *dedicated issue slots*, *extended operands*, *extended results*, and *multicast*. Evaluation of the execution time of the models requires both the dynamic cycle count and clock period. We developed an advanced instruction scheduler for all the five ICC models in order to quantify the dynamic cycle counts of our multimedia C benchmarks. To assess the clock period of the ICC models we designed and laid out VLIW datapaths using the RTL hardware descriptions derived from a deeply pipelined commercial TriMedia processor. In contrast to prior art, our research shows that fully distributed register file architectures (with eight clusters in our study) often underperform compared to moderately clustered machines with two or four clusters because of explosion of the cycle count overhead in the former. Among the evaluated ICC models, performance of the *copy operation* model, popular both in academia and industry, is severely limited by the copy operations hampering scheduling of regular operations in high ILP (instruction-level parallelism) code. The *dedicated issue slots* model combats this limitation by dedicating extra VLIW issue slots purely for ICC, reaching the highest 1.74 execution time speedup relative to the unicluster. Furthermore, our VLSI experiments show that the lowest area and energy consumption of 42 and 57% relative to the unicluster, respectively, are achieved by the *extended operands* model, which, nevertheless, provides higher performance than the *copy operation* model.

Categories and Subject Descriptors: C.1.4 [**Computer Systems Organization**]: Processor Architectures—*Parallel architectures*; H.4.3 [**Hardware**]: Input/Output and Data Communications—

Authors' address: Andrei Terechko, NXP Semiconductors, High Tech Campus 32 (WDC) 5656 AE Eindhoven, The Netherlands; email: andrei.terechko@nxp.com; Henk Corporaal, Faculty of Electrical Engineering, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands; email: h.corporaal@tue.nl.

Interconnections (subsystems); D.3.4 [**Software**]: Programming Languages—*Processors (compilers, code generation)*

General Terms: Design, Experimentation, Performance, Languages

Additional Key Words and Phrases: Instruction-level parallelism, VLIW, intercluster communication, optimizing compiler, instruction scheduler, cluster assignment, register allocation, pipelining, clock frequency

ACM Reference Format:

Terechko, A. S. and Corporaal H. 2007. Inter-cluster communication in VLIW architectures. ACM Trans. Architec. Code Optim. 4, 2, Article 11 (June 2007), 38 pages. DOI = 10.1145/1250727.1250731 http://doi.acm.org/10.1145/1250727.1250731

1. INTRODUCTION

Convergence of *digital video, vision, 3D graphics, still image, networking, security, natural languages,* and *audio* in embedded multimedia devices demands high performance and low power consumption. On top of this, constantly emerging communication and content standards, along with requirements for device customization and in-field upgrades, necessitate high-level programmability. To meet these great demands embedded media processors rely heavily on optimizing compiler technology and hardware parallelism in the forms of pipelining, ILP (instruction-level parallelism), vector parallelism, data-level parallelism, and task-level parallelism [Fisher et al. 2004]. In this paper, we investigate ILP exploitation, in which modern compilers for sequential high-level languages are most mature. However, ILP extraction techniques can naturally coexist with other parallelization methods.

Previous studies of *ILP limits* [Lam and Wilson 1992; Lee et al. 2000; Liao and Wolfe 1997] indicate availability of potentially high operation concurrency in (media) applications, spanning the range of a few tens up to hundreds of independent operations. However, the ILP rates, extractable either statically by efficient fast compilers or dynamically by (superscalar) hardware, are much lower. Note, that fast compilation is crucial for quick time-to-market including numerous "patch-compile-execute" cycles. Despite active ILP research in the past 25 years, automatically extracted operation concurrency of full applications never neared the high potential. Extracting high ILP in future media applications is limited by the growing control intensity of the code (e.g., the context-based adaptive arithmetic coding from the H.264 video coding standard). On top of that, the rapidly widening gap between processor and memory speeds further mitigates the achievable ILP. Indeed, because of the increase of processor stall cycles (e.g., for cache misses) the number of instructions executed per cycle (IPC) decreases. Furthermore, feeding a wider ILP processor with data will be a major challenge for the bandwidth of slowing down memories. To match capabilities of realistic compilers and hardware trends, we confine our study to media processors with the issue width of 8. Thanks to our commercial optimizing C compiler, this amount of hardware parallelism is sufficient to sustain the ILP rates of 4.8 to 7.5 concurrent operations on complex applications. In fact, the achieved ILP is higher because of subword parallelism in SIMD operations.

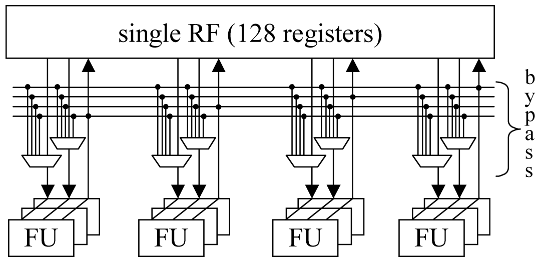

Fig. 1. Unicluster VLIW architecture.

Many embedded media processors exploit ILP using the *VLIW (very long instruction word) architecture* [Fisher 1981; Fisher et al. 2004] resulting from its high performance through parallel execution of operations and relatively low power and die area. These competitive characteristics are conditioned on the absence of costly hardware mechanisms to recover parallelism ILP from sequential programs, as it is done in superscalars. The classical VLIW datapath contains a number of parallel function units (FUs), a multiported register file (RF) and, if the processor is pipelined, 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. For example, if the FUs in Figure 1 have pipeline registers at their inputs, the bypass network can forward a result produced by one FU to another even before this result is committed to the RF. This way pipelined back-to-back execution of data-dependent operations is implemented. Note the multiple FUs behind the five VLIW issue slots in Figure 1, which designate (slightly) different functionalities supported by the issue slots.

Unfortunately, the large multiported RF and the bypass network hamper the ILP scalability of the processor, which only aggravates with advance of VLSI technologies [Ho et al. 2001; Agarwal et al. 2000; ITRS Technology Working Group 2005]. According to our VLSI layout experiments, a multiported RF for a unicluster VLIW with full connectivity to eight issue slots becomes impractically large and slow. Hence, increasing hardware ILP and, at the same time, sustaining or boosting the clock frequency becomes impossible for unicluster architectures. Therefore, (commercial) media processors (e.g., TMS320C6xxx, Equator BSP, HP/ST Microelectronics ST2xx, Analog Devices Tiger Sharc, BOPS Manarray) address this issue by splitting the monolithic architecture in smaller *clusters* (see Figure 2). Each cluster contains a local RF fully interconnected with cluster's FUs. Inter-cluster communication (ICC) is typically exposed in the ISA (instruction-set architecture) in the spirit of the VLIW approach. The smaller clusters reach significantly higher clock frequency (e.g., 1 GHz for the TI TMS320C6xxx DSPs) while preserving competitive low power consumption and chip area.

Note the two pipeline registers on the ICC paths in Figure 2. If each cluster occupies significant area or there are many clusters in the processor, the ICC wires span across long distances on the chip, substantially compromising the clock speed. This effect becomes especially prominent in the forthcoming IC

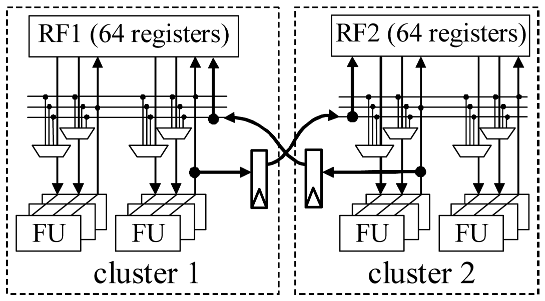

Fig. 2. Two-cluster VLIW architecture.

technologies [Ho et al. 2001; Agarwal et al. 2000]. To achieve 1 GHz the popular TI TMS320C64xx processor features intercluster pipeline registers (similar to our two ICC registers in Figure 2) in order to combat the long ICC wire delays. On the other hand, smaller and/or slower media processors can afford not instantiating the pipeline registers and, thus, decrease the execution cycle count. Our research concentrates on pipelined ICC paths aiming at high frequency high-end media processors.

The existing scientific literature mainly focused on one ICC model (e.g., copy operations) in the processor ISA. In contrast, the goal of our study is to characterize and systematically compare *several* intercluster communication models based on their execution time, area, and energy efficiency. Our scientific intuition behind this goal is that choosing a proper ICC model in the ISA can bring substantial performance and energy efficiency improvements. We strive to do research by realistic experiments via IC layout exercises instead of analytical VLSI models and employing a deeply pipelined VLIW architecture, derived from a commercial media processor. Low ILP code is a dangerous driver for processor's design space exploration, because it underutilizes parallel hardware leading to poor design decisions. Therefore, we employ full optimized C benchmark applications instead of out-of-the-box code with low ILP.

This paper is organized as follows. First, we introduce our taxonomy of intercluster communication models in VLIW ISAs in Section 2. Section 3 elaborates on our instruction scheduler capabilities. VLSI implementation characteristics derived from our layout exercises of five ICC models constitute Section 4. Analysis of the execution time of compiled C benchmarks for our models is presented in Section 5. Furthermore, this section includes energy consumption figures obtained from RTL power simulations. In Section 6 we put our research in perspective with prior studies. Finally, Section 7 concludes with recommendations for selecting the best ICC model and promising future research directions.

2. TAXONOMY OF INTERCLUSTER COMMUNICATION MODELS

Taxonomy of ICC models and corresponding VLSI implementations is vast [Parcerisa et al. 2005; Rixner et al. 1999; Zalamea et al. 2003; Gibert et al. 2005; Gangwar et al. 2003; Terechko et al. 2003a]. Before detailing the evaluated ICC models, in the beginning of this section, we briefly traverse the design space of intercluster communication mechanisms. For example,

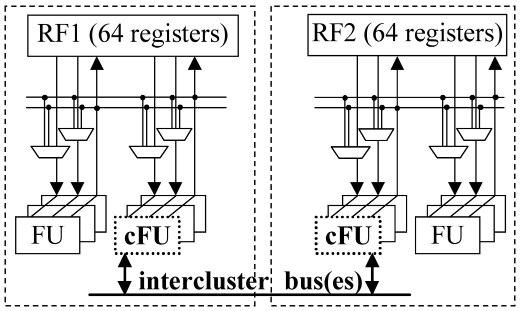

Fig. 3. Bus-based two-cluster architecture.

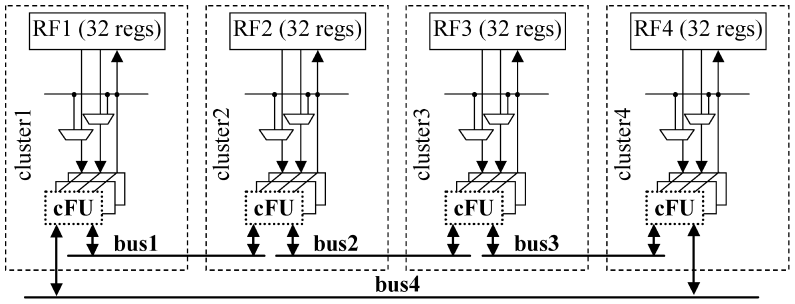

Fig. 4. Partially connected four-cluster VLIW with bus-based ICC.

there exist point-to-point and bus-based clustered VLIW processors. *Point-to-point* networks of clusters contain dedicated connections among clusters (see Figure 2). If some point-to-point connections are merged, we can derive the popular *bus-based* ICC model [Faraboschi et al. 2000; Gangwar et al. 2005; Lapinskii et al. 2002; Bekooij 2004] (see Figure 3). According to Bekooij [2004], a global bus can significantly simplify scheduling for a partially connected VLIW. However, as concluded in Gangwar [2005] and Parcerisa et al. [2005], the (global) bus substantially limits the clock frequency. In this study, we focus on point-to-point networks of clusters, but some of our results are also applicable to bus-based clustered architectures. For example, some ICC constraints incurring a cycle count overhead of a clustered VLIW are independent of the actual implementation of the interconnect (bus-based or point-to-point).

We also differentiate *fully* and *partially connected* networks of clusters. In the fully connected VLIW, each cluster has a direct connection to all others. This naturally speeds up ICC and simplifies the job of the instruction scheduler. On the other hand, the scalability of the fully connected VLIW is limited. This issue is overcome by partially connected and tiled architectures [Lee et al. 2000; Roos et al. 2002; Colavin and Rizzo 2003] (see Figure 4). The VLIW clusters in Figure 4 are organized in a ring fashion, so, for example, clusters 1 and 3 have no direct connection between each other. Unfortunately, partially connected clustered architectures are notorious for complex instruction scheduling, struggling with deadlock avoidance, which occurs when a copy path between two clusters cannot be scheduled [Roos et al. 2002]. Our work focuses on fully connected deadlock-free networks. Moreover, as we expressed in Section 1, the achievable ILP of media applications is modest, and, thus, the benefit of extreme scalability is arguable for media applications (e.g., control-intensive video codecs). Note, how clustering simplifies the bypass network and reduces the number of ports on the registers (see Figures 1, 3, and 4).

The intercluster 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 intercluster data transfers. Moreover, the number of intercluster transports per instruction should not exceed the intercluster bandwidth. These hardware restrictions require explicit specification of the intercluster communication in the VLIW code and, consequently, the VLIW ISA. Naturally, the compiler for a clustered VLIW processor gets complicated by this exposure of the hardware restrictions in the ISA. There exist *many ways to express intercluster communication in the ISA*. As this paper shows, the ICC model, to a large extent, determines the execution time of a clustered processor. In this section, we define and qualitatively analyze five models of ICC. To introduce each ICC model, we only sketch its architecture with simplified pipelines. The actual pipelined microarchitecture used for our experiments is further detailed in Section 4. Table I (see later) summarizes hardware complexity characteristics of the considered models.

Each description of an ICC model is accompanied by a *scheduled VLIW assembly snippet* to show intercluster transport in this model. Below is an example of code showing two data-dependent VLIW instructions of a four-issue-slot unicluster 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 (if an operation has multiple results), and symbol → separates operands from results. In the examples for the ICC models (shown in the following sections), sequences of operations in the first one-half of the columns belong to one cluster and in the other one-half, to the other cluster; index in square brackets identify the cluster. Each presented instance of the ICC models has the ICC bandwidth of a single ICC transfer per cluster per VLIW instruction. The latency of the inter-cluster transfers in the examples is one cycle. Note that in the unicluster code example below, there is merely a single cycle delay between two data-dependent operations `op1` and `op2`.

```
  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. An example code with intercluster transport by means of *copy operations* is presented below. The value of `r3` from RF1 in cluster 1 is

Table I. Complexity Characteristics of the Inter-cluster Communication Models[a]

Inter-cluster Communication Model	Extra #Read Ports	Extra #Write Ports	#Muxes in Bypasses	#Inputs Per Bypass Multiplexer	VLIW Instruction Size Increase Relative to Unicluster (Bits)
Unicluster	0	0	$2N_{slots}$	$N_{slots}+1$	0
Copy operations	0	B_w	$2N_{slots}$	$N_{slots}/C+B_w+1$	$-N_{operands}log_2(C)$
Dedicated issue slots	B_r	B_w	$2N_{slots}+B_wC$	$N_{slots}/C+B_w+1$	$B_wC[log_2(C-1)+2N_{bits}]-N_{operands}log_2(C)$
Extended operands	B_r	0	$2N_{slots}+B_wC$	$N_{slots}/C+B_w+1$	$B_rC[log_2(C)]-N_{operands}log_2(C)$
Extended results	0	B_w	$2N_{slots}$	$N_{slots}/C+B_w+1$	$B_wC[log_2(C-1)+N_{bits}]-N_{operands}log_2(C)$
Multicast	0	B_w	$2N_{slots}$	$N_{slots}/C+B_w+1$	$B_wC[M(N_{bits}+1)]-N_{operands}log_2(C)$

[a]Explantion of Symbols C, number of clusters;
N_{slots}, total number VLIW issue slots (in all clusters);
$N_{operands}$, number of source and destination operands in all issue slots of a single VLIW instruction;
N_{bits}, number of bits per (source or destination) operand field;
B_w, B_r, number of processor words a cluster can write to or read from the other clusters per cycle (bandwidth);
M, number of destinations of a multicast operation. In the general case it equals to C-1.

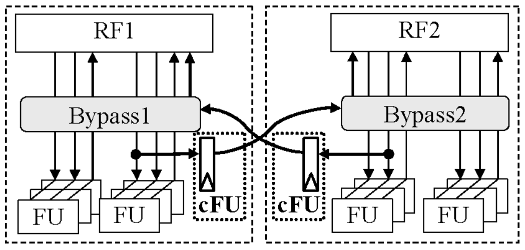

Fig. 5. Copy operations.

transferred to `r1` of `RF2` in cluster 2. The intercluster data transfer 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  ;
```

Figure 5 depicts an implementation of the copy operation of the ICC model. 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 intercluster pipeline register (see Figure 5). 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 one extra write port on the RFs per intercluster path and has a rather simple bypass network compared to other models (see columns 2, 3, 4, and 5 in Table I).

This encoding of ICC implies that some VLIW issue slots will be occupied with intercluster copy operations. In the dense high ILP code, 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.

A bus-based variant of this ICC model is used in the ISA of ST Microelectronics and Hewlett Packard ST2xx [Faraboschi et al. 2000]. In fact, Lx requires two copy operations per intercluster transfer: `send` and `receive`. The `send` initiates the data transfer in the source cluster and the `receive` obtains the data in the destination cluster. In fact, this scheme removes the extra write port on the RFs and, consequently, simplifies the bypass network. On the other hand, instead of a single copy operation for each transfer, this scheme needs two, which negatively impacts the code size and scheduling freedom for other operations. Remarkably, the send and receive model can easily be extended to support multicast (Section 2.5). A multicast would be carried out by a single `send` operations, putting a value on the bus, and multiple `receive` operations in several clusters, picking up the value. This technique, however, adds significant complexity in register allocation and scheduling.

2.2 Dedicated Issue Slots

In this model, the VLIW instruction is extended with extra *issue slots dedicated* to intercluster communication (see Figure 6). For example, each processing

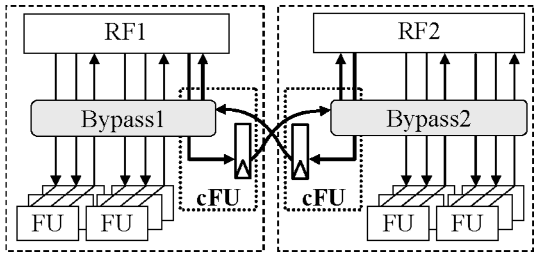

Fig. 6. Dedicated issue slots.

element (cluster) of BOPS's ManArray [Levy 2001] 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 very different performance and VLSI implementation characteristics.

Implementation of this model is relatively expensive. Extra dedicated issue slots lead to expansion of the VLIW instruction, complicating the instruction decoder and instruction fetch unit. Moreover, this ICC model needs two extra RF ports per dedicated slot and the number of multiplexers in the bypass network is comparatively high (Table I). Below is the code for the machine, shown in Figure 6, with two dedicated issue slots for intercluster transport in slots 3 and 4. In fact, the two extra slots make the total issue width equal to six. The intercluster 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  |  *   ;
```

Dedicated issue slots cannot issue regular operations, because they have insufficient read ports for dual-source operand operations. If we added an extra read port (possibly sacrificing the achievable clock frequency) and allow scheduling regular operations in the dedicated slots, our wider VLIW machine would implement the copy operation model. In the view of a possible clock frequency decrease and the limited scheduling freedom of intercluster communication in the copy operation model, we did not consider the wider VLIW configuration with dedicated issue slots that can start regular operations.

2.3 Extended Operands

The source operands in this ICC model are extended with cluster identification. For example, the Texas Instruments VelociTI architecture extends some of the operands with cluster-id fields. These fields specify the RF where the values should be read from. VelociTI restricts the intercluster bandwidth to one intercluster read per VLIW instruction per cluster. This model allows using a value

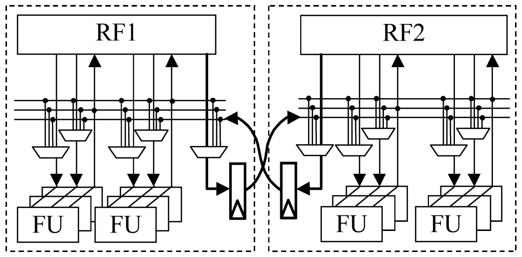

Fig. 7. Extended operands.

from a remote RF without storing it in the local RF (see Figure 7), 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 complicated. Note that this model has more multiplexers in the bypass than the unicluster.

The code below illustrates the extended operand model. The first argument of the operations is extended with specification of the RF. `op1`, thus, reads the local value of `r1` from `RF1` and writes to `r3` in `RF1`. In the mnemonics of `op1`, we could have omitted [1]. However, in the binary encoding of the operation, there will be a bit indicating what RF the value should be read from. Two cycles later, `op2`, in the second cluster, consumes the result of `op1` (`r3` from `RF1`) without storing it in `RF2`. A downside of this ICC model is that the hardware should detect and initiate the intercluster transfer rather early in the pipeline, which may complicate the bypass logic in deep pipelines. Moreover, the cluster-id extension is always fixed to the VLIW instruction with the corresponding operand, which limits the scheduling freedom of intercluster 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, Section 1. In this ICC 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 intercluster moves rather than copies. Therefore, if the result of an operation is required in both the local and remote clusters, the instruction scheduler must add an extra operation to the data-flow graph (DFG). In the code example below, the result of `op1` is moved to register `r1` in cluster 2 without being stored in cluster 1. Note that according to our mnemonics, `op2` writes to `r3` in `RF2` specified by `r3[2]`. Obviously, [2] could have been omitted in the assembly, but not in binary encoding of the operation.

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

The hardware implementation of this model is comparable to that of the copy operation. Furthermore, in contrast to the extended operands, the pipeline of the extended results model can initiate the ICC transfer relatively late, in or after the execute stage.

2.5 Multicast

To specify multicasting, the operation's destination field is extended with (multiple) register- and cluster-ids. This way the ICC value can be sent to a selected number of remote clusters within a single transfer, which relative to broadcasting, reduces RF pressure by not duplicating registers in clusters where the value will not be used. In terms of implementation complexity, the *multicast* model is similar to the *extended results* and *copy operations* models (Table I). Conceptually, this model resembles the `sendb` operation from the CRB scheme [Kailas et al. 2002, 2001], however, various realizations of this model, for example, in the binary instruction format, can be devised. 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` until the third VLIW instruction.

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

An important trade-off between the code size and register allocation freedom in the scheduler is whether to extend the multicast operation with both cluster- and register-ids in the remote RFs or only cluster-ids (and write to the same destination register-id in all specified clusters). After our initial experiments showed the high impact of scheduling freedom on the final performance of the clustered processor, we selected the multicast with both register- and cluster-ids for our performance evaluation.

A peculiar instance of the multicast model is *broadcasting* to shared register addresses used 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 our notion of clustering. However, for example, the Sun MAJC architecture [Sudharsanan et al. 2000] avoids a shared RF by replicating the "shared registers" in all clusters. The contents of the replicated registers are kept synchronized (see Figure 8).

The FUs always read the "shared registers" from the local copy, whereas the writes to the "shared registers" are broadcast 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 broadcast value from its copy of the shared registers after the delay of the intercluster transfer. Besides the "shared registers," this model allows local RFs that are accessible only within one cluster. The FUs are fully connected to the local RF of a cluster and the local copy of the global RF.

From the implementation point of view, replication of the registers costs extra area. To lessen the number of write ports on the replicated RFs, we can restrict

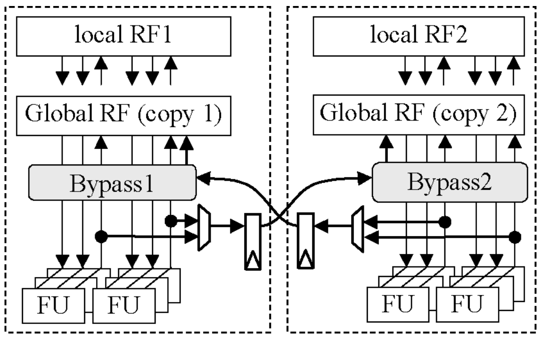

Fig. 8. Shared register addresses.

the number of writes to the "shared registers" per instruction. In Figure 8, for example, only one operation per cluster can write to a "shared register" via two 2-to-1 multiplexers. Nevertheless, broadcasting to all clusters is never power efficient.

In the example code below, the register address space is split, like in Sun 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`. `op2` is, consequently, delayed until the third instruction by the scheduler.

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

In contrast to our previous publication [Terechko et al. 2003a], broadcast does not outperform other models. We attribute this to the unrealistic modeling of allocation of global values (alive on the scheduling units' boundaries), which were assumed in Terechko et al. 2003a to be accessible, with no penalty from the global register file. However, our new benchmarks showed that the globals do not always fit in the "shared" RF. Furthermore, broadcasting has a higher register pressure, since every ICC data transport duplicates the values in all clusters. The latter effect led to overutilization of the registers during spilling, which our instruction scheduler could not efficiently cope with. Since multicast is potentially superior, in terms of performance, than broadcasting, we used only the multicast model in our performance evaluation.

In Table I, we summarize the main characteristics of all the presented ICC models. Interestingly, the instruction-size column indicates a trade-off between the shrunk operand sizes and extra ICC fields. Note, that in Table I, we did not include possible instruction-size expansion because of increased VLIW headers (e.g., in the *dedicated issue slots* model), because of high dependency on a concrete binary encoding of the VLIW instructions. Validation of an intercluster transfer within a multicast can be encoded by using a constant register as the destination (instead of the shown +1), indicating that the corresponding intercluster transport is disabled. Note, that `log`$_2$`(C)` in the last column equals to the number of bits reduced in each operand as a result fewer addressable registers in a cluster.

3. COMPILATION FOR ICC MODELS

Although clustering boosts higher clock speed, it also incurs overhead in the number of execution cycles. We recognize the following factors of the cycle count overhead:

1. extra *latency* of intercluster data transfers
2. limited intercluster *bandwidth*
3. higher *register pressure*
4. intercluster communication *model constraints*
5. extra *cache stall cycles* because of code size overhead and higher register pressure

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 intercluster 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 partly compensate each other, which leads to the static code size deviation within ±5% with respect to the unicluster. Having measured 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 to share the cache among all clusters. Having considered the complexity of the trade-offs, we conclude that evaluation of data memory for clustered processors deserves a separate study, such as Gibert et al. [2002, 2005] and, therefore, we assume an ideal shared memory (no cache misses) in the remainder of this paper.

We measured the cycle count overhead using the commercial *state-of-the-art TriMedia C/C++ compiler* TCS 4.6. For our study we enhanced the TriMedia's unicluster instruction scheduler to schedule the intercluster communication in all the five ICC models. The scheduling unit is the guarded decision tree of basic blocks [Hsu and Davidson 1986; Hoogerbrugge and Augusteijn 1999; Havanki et al. 1998]. The guarded decision tree is an acyclic control-flow graph without join points and side entries, which is a more general case than hyper and superblocks. Furthermore, our instruction scheduler employs an advanced intra-function analysis for cluster assignment of global values described in Terechko et al. [2003b]. Inspired by Özer et al. [1998], Janssen [2001], Kailas et al. [2001] and Codina et al. [2001], 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 [Kailas et al. 2001; Özer et al. 1998] and yields significantly denser code [Janssen 2001].

3.1 Instruction Scheduling

Our scheduling algorithm is operation-based. First, the scheduler computes operation priorities according to the heuristic presented by Hsu and Davidson [1986]. It then, composes a list of ready-to-schedule operations that have no dependencies on not yet, scheduled operations. Next, from the ready-to-schedule list, the scheduler selects an operation with the highest priority and schedules it according to Algorithm 1. Function `build_ordered_cluster_list()` from Algorithm 1 builds an ordered list of possible cluster assignments 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 a 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, and c_c, c_{rf}, c_{slots} term coefficients, $c_c > 0$, $c_{rf} > 0$, $c_{slots} > 0$.

This cost function gives higher costs to cluster assignments requiring more copy operations, with higher RF pressure and issue slot utilization. Essentially, it stimulates load balancing of operations among the clusters and minimization of expensive intercluster transfers. The term coefficients were experimentally fine-tuned and equal to 1, 1, and 0.9 for c_c, c_{rf}, c_{slots}, respectively. If the RF pressure exceeds a certain threshold (85% of registers or fewer than five free registers in our experiments), then coefficient c_{rf} is automatically multiplied by 5. The coefficients were kept the same for all the ICC models, since we discovered no consistent improvements from tailoring them to each model.

Algorithm 1 Instruction scheduling algorithm

```
1: schedule_operation(oper, tree) {
2:     cluster_list = build_ordered_cluster_list(oper, tree);
3:     for (instr->cycle = i_min; instr->cycle <= i_max; instr = instr->next) {
4:         for (cl = cluster_list->head; cl; cl = cl->next) {
5:             if (!assign_oper_to_slot (oper, cl, instr))
6:                 continue;
7:             if (!schedule_floaters(oper, cl, instr) {
8:                 unschedule_floaters(oper);
9:                 continue; }
10            if (!schedule_copies(oper, cl, instr)) {
11:                unschedule_copies(oper);
12:                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 */
}
```

Register live range information is kept in register bit vectors per VLIW instruction. To shorten register live ranges, the scheduler employs floater operations using function `schedule_floaters()` in Algorithm 1, as described in Hoogerbrugge and Augusteijn [1999]. Copy operations are scheduled in `schedule_copies()` by adding new operation nodes to the data-flow graph and, subsequently, scheduling them just like regular operations. Note, that if scheduling of the operation in question fails, the corresponding copy operations are unscheduled and removed from the data-flow graph. To efficiently fill in the eight branch delay slots of the TriMedia, the scheduler uses backtracking. The jumps are scheduled optimistically in an early cycle, and, if the scheduler does not manage to fit the remaining operations in the branch delay slots (checked by `too_optimistic_jumps()` in Algorithm 1), it backtracks using the `backtrack()` function. If no available register is found for operation's results in `assign_register()`, spill and restore code is inserted on the fly by `schedule_spill_restore()`. Remarkably, spill and restore operations may trigger scheduling of extra copy operations.

Minimization of intercluster traffic plays a major role in our cluster assignment by having a high contribution to the cost with the $c_c N_c$ term in Formula (1). In contrast to prior research, our scheduler assigns an operation to the cluster that requires the fewest intercluster data transfers by examining predecessors *and successors* of the operation. If we considered only predecessors `p1` and `p2` of operation `o1` from Figure 9, 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 later require another copy operation from `p3`. In fact, one can analyze even larger 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, which, on the other hand, increase the compilation time.

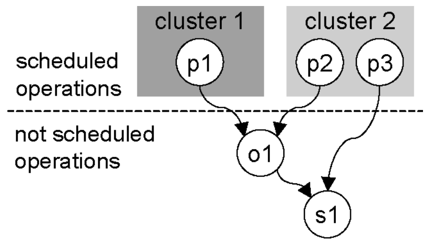

Fig. 9. Accounting for future copy operations.

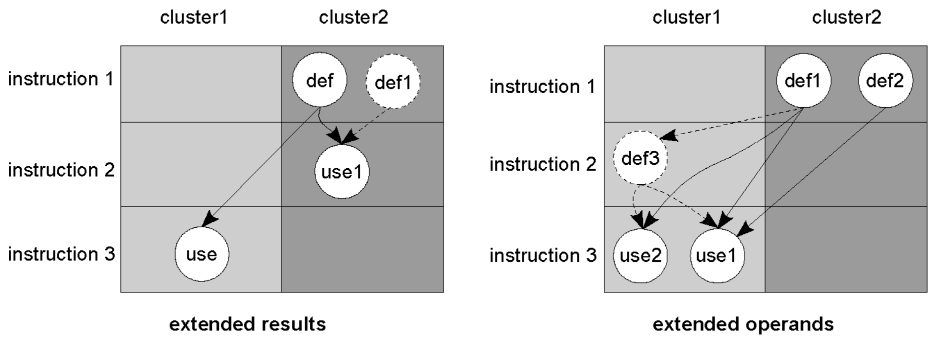

Fig. 10. Model-specific scheduling.

3.2 Model-Specific Instruction Scheduling

The ICC models require various scheduling techniques specific to each ICC model. For example, in the extended results model, a result of an operation may be copied to a remote RF. However, if some consumers of this result reside in clusters, where the result was not copied to, the scheduler has to add and schedule a compensation operation, transferring the same result to the other cluster(s). For example, in Figure 10 the `def` operation has to feed both `use` and `use1`. If `use` is scheduled in cluster 1, we need to add compensation operation `def1` to feed `use1` sitting in the other cluster 2. Compensation operation `def1`, in this example, can be merely a duplicate of operation `def`. If the same code was to be scheduled for the dedicated issue slots model, the result of `def` will also be available in cluster 2 and `def1` will not be required. Therefore, this effect in the extended results model leads to unfortunate performance loss resulting from the occupation of regular issue slots.

Another important model's peculiarity is present in the extended operands model. If two source operands of the same operation `use1` (see Figure 10, right side) require intercluster transfers from operations `def1` and `def2` residing in a different cluster and the intercluster communication bandwidth is restricted to one transfer per cluster per cycle, then our scheduler has to add compensation code `def3`. `use1`, then, reads one operand from a local register produced by `def3`, and the other operand using an extended operand's cluster identification. On top of that, in the extended operands model "reuse" of intercluster transferred values is quite cumbersome, since the operation being scheduled immediately consumes the sent value without storing it locally. At least two options resolve this for operation `use2` in Figure 10: to `use` compensation code `def3` to feed `use2` from a local register or to postpone `use2` until the next instruction. The first option is more expensive, because `def3` occupies a regular issue slot and postpones (not shown in the figure) both `use1` and `use2` to satisfy the intercluster read delay for `def3`. Our scheduling experiments with high ILP code showed that the second approach of postponing `use2` to the next instruction outperforms the first one.

For scheduling the multicast operation, we employ the same technique as described in Kailas et al. [2002] for scheduling operation `sendb`. According to this technique, our scheduler attaches new intercluster data transfers to the already scheduled ones.

The ICC models expose different degree of scheduling freedom. For example, in the *dedicated-issue slots* model, the ICC transport can be scheduled in any time slot between the producer and consumer, whereas, in other models, it is typically fixed to the cycle of the producer or consumer. The presented model-specific scheduling constraints will further impact the final schedules. In conclusion, we can expect that *the scheduling freedom provided by the models will influence the cycle count overhead* and the overall VLIW performance. Note, that our performance and energy measurements in subsequent sections are sensitive to the quality of the compiler and the instruction scheduler, in particular. According to comparisons of our schedule quality against a vast random search [Terechko et al. 2003b], our results are only few percentage off, relative to the lowest achieved bounds. Further compiler tuning and optimizations for particular ICC models and global values can increase the presented benefits from clustering.

4. MICROARCHITECTURE AND LAYOUT EXPERIMENTS

Although there exist deep studies of the cycle count overhead of clustered ILP processors [Faraboschi et al. 2000; Kailas et al. 2001; Colavin and Rizzo 2003], few studied the clock period thoroughly. A lower clock period of a processor can be achieved by various techniques: pipelining, clustering, RTL optimizations, VLSI circuit optimizations, etc. Therefore, to measure the added benefit of clustering solely and stay realistic, we used the *deep 16-stage pipeline* of the optimized-for-speed VLIW media processor TriMedia TM5250 [Halfhill 2004] as the starting point. The pipeline features a two-stage register file access, a dedicated stage for bypassing (data forwarding), and a fast single-cycle ALU design. Figure 11 illustrates our microarchitecture and pipeline of a single cluster for the *copy operation* model. To transfer a value to another cluster, the value is, first, read from the local RF in stages `READ1` and `READ2`. It is then fed into the bypass network of, say, source operand `src1` of `slot2`. In the execute stage, it is *transferred to remote clusters*, which takes a complete cycle. Upon arrival to the remote cluster, the value is multiplexed with values sent from other clusters and registered, see block ICC in the figure. This multiplexer allows fixing the number of RF ports dedicated to ICC, despite the growth of the number of clusters, keeping the ICC bandwidth constant (one ICC read or one ICC write in our example). In the next cycle, the value is forwarded to the bypasses of

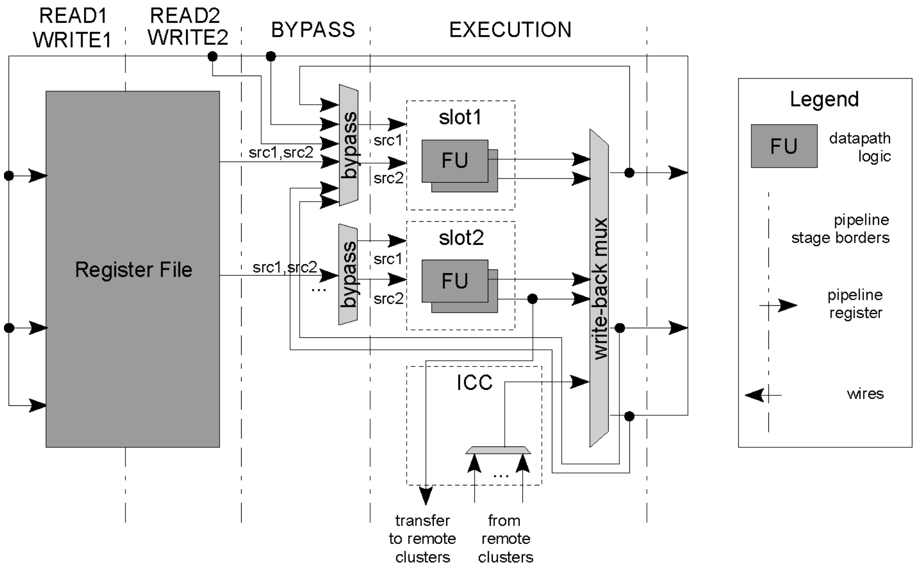

Fig. 11. Microarchitecture of the copy operation model (single cluster, two issue slots per cluster).

the destination cluster and to the `write-back multiplexer`, just as any other local result. Forwarding the value into the bypass network allows operations in flight to consume their operands that are not yet committed to the RF. Note that only the bypasses for the first issue slot are fully shown in Figure 11, to simplify the drawing. The register file in this model has one extra write port for intercluster communication.

The implementation of the *copy operations*, *extended results* and *multicast* differs by few logic gates in the instruction decoding and ICC interfaces, whereas the major datapath blocks (RF, bypass network, write-back bus multiplexing, etc.) are the same. We have neglected differences between these ICC models and, consequently, these models have the same VLSI properties in our evaluation. The hardware for the dedicated slots and extended operands was implemented differently, but with the same pipeline structure as depicted in Figure 11. The microarchitecture and pipeline of the *dedicated issue slots* model is presented in Figure 12. One can notice the extra register file read port and an accompanying bypass multiplexer required for the dedicated issue slot, complicating the hardware implementation. Figure 13 presents the extended operands microarchitecture. This model benefits from no extra RF write ports relative to the unicluster and, thus, a simple write-back multiplexer.

To quickly create RTL (register transfer level) descriptions for various clustered VLIW datapaths, we built an *RTL generator*. The generated RTL code was placed and routed by Cadence tools in standard cell CMOS 130-nm technology. To characterize VLSI properties of the presented ICC models, we laid out eight-issue-slot 32-bit VLIW data paths for the unicluster and clustered machines in two, four, and eight-cluster configurations. On top of the deep pipelining, we manually applied various RTL timing optimizations (bit slicing, substituting multiplexer gates with faster logic gates, logic redesign, register duplication

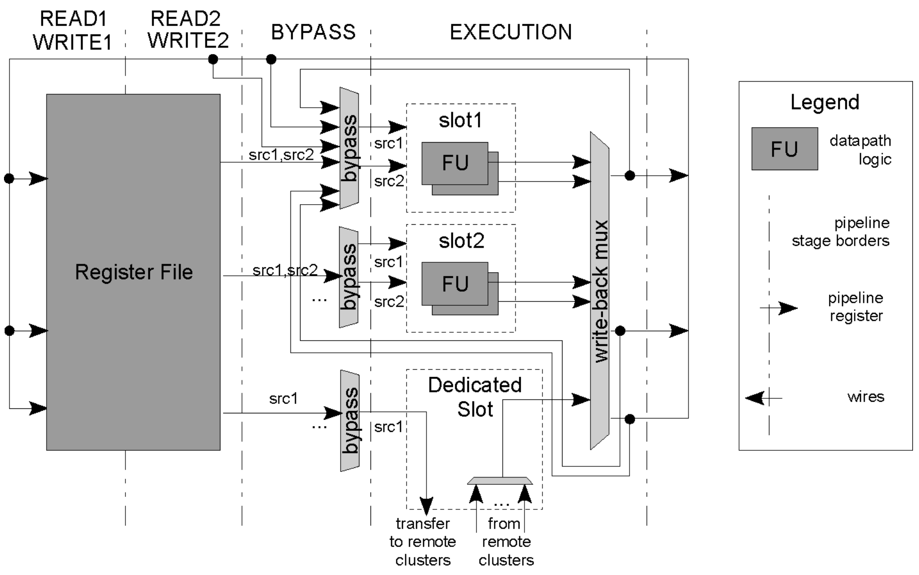

Fig. 12. Microarchitecture of the dedicated slots model (single cluster, two issue slots per cluster).

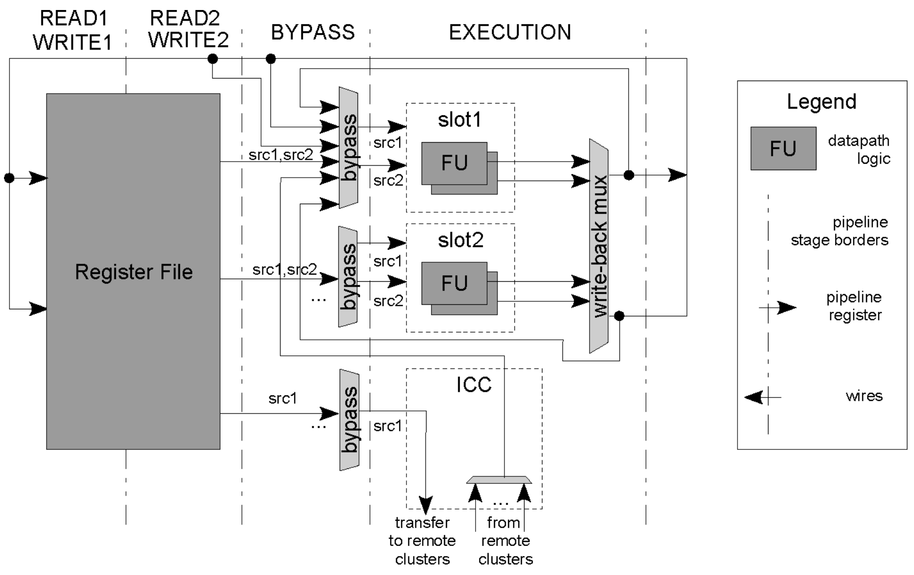

Fig. 13. Microarchitecture of the extended operands model (single cluster, two issue slots per cluster).

to reduce fan-out, pipeline retiming moving timing critical logic to early/later stages). The outcome of our optimizations was a well-balanced pipeline with the critical path in the FU bypasses, except for the unicluster, where the critical path lied in the big register file. Figure 14 demonstrates the cluster layout approach used for our evaluations, showing the floorplan of an eight-cluster VLIW machine. Each cluster was allocated to a rectangular region on the

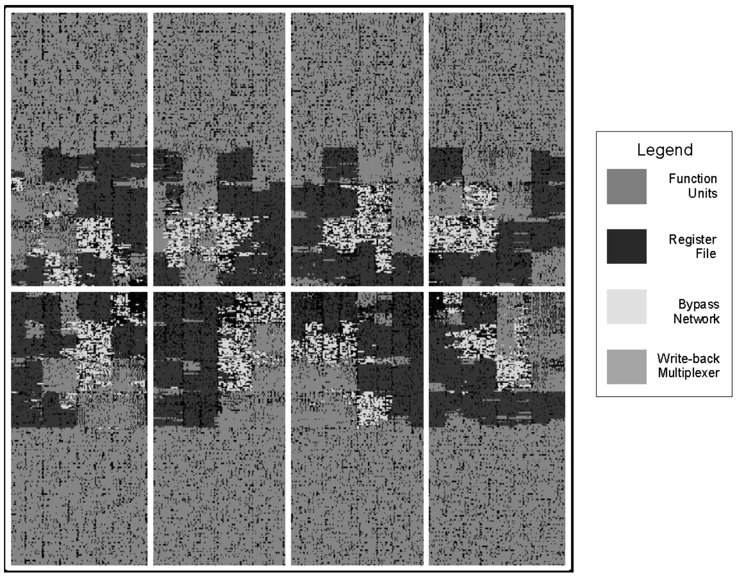

Fig. 14. Layout of an eight-cluster eight issue slot VLIW data path with extended results.

Table II. Microarchitecture Parameters of the Evaluated VLIW Machines[a]

Inter-cluster Communication Model	Copy Operations; Extended Results; Multicast	Dedicated Slots	Extended Operands
Total issue slots	8	$8 + C$	8
Slots per cluster	$8/C$	$(8 + C)/C$	$8/C$
ICC bandwidth (read:write)	0 : 1	1 : 1	1 : 0
Register ports (read:write)	$2 \bullet 8/C : 1 \bullet 8/C + 1$	$2 \bullet 8/C + 1 : 1 \bullet 8/C + 1$	$2 \bullet 8/C + 1 : 1 \bullet 8/C$
Bypass complexity (#inputs: #bypass trees)	$3 \bullet (8/C + 1) + 1 : 2 \bullet 8/C$	$3 \bullet (8/C + 1) + 1 : 2 \bullet 8/C + C$	$3 \bullet (8/C) + 1 : 2 \bullet 8/C + C$

[a]C, number of clusters.

floorplan, which guided the cell placement tool. In this case, the place-and-route tool has the freedom of selecting different layouts for each cluster, which is demonstrated by the layout in Figure 14.

Before discussing the layout results, Table II summarizes the microarchitecture parameters of our VLIW machines. The machines contained 128 registers, in total, evenly distributed among the clusters. Each issue slot had four or five FUs, except for the dedicated slots with a single ICC FU. The intercluster communication latency was one cycle. The presented ICC bandwidth, RF ports, and bypass complexity are per cluster.

Figure 15 presents the clock frequencies of the layouts after detailed routing and extraction of parasitic capacitances. The clock frequency of the unicluster reached only 200 MHz, being severely limited by the slow shared RF and bypass network. As expected, the layouts of smaller clusters reach higher clock frequency. Figure 15 shows a significant clock frequency leap from 200 MHz

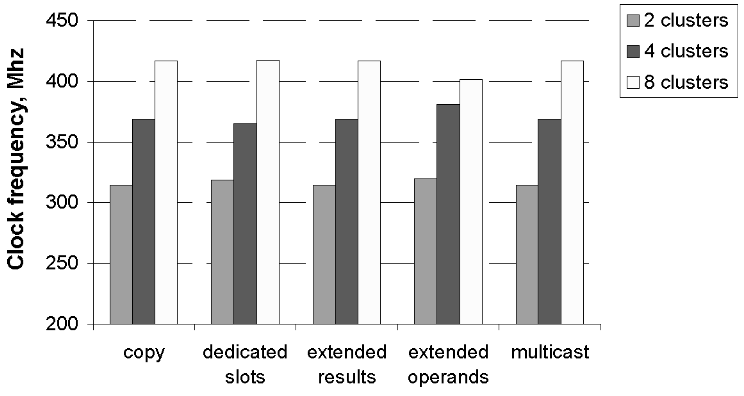

Fig. 15. Clock frequencies of the clustered VLIW datapaths (unicluster: 200 MHz).

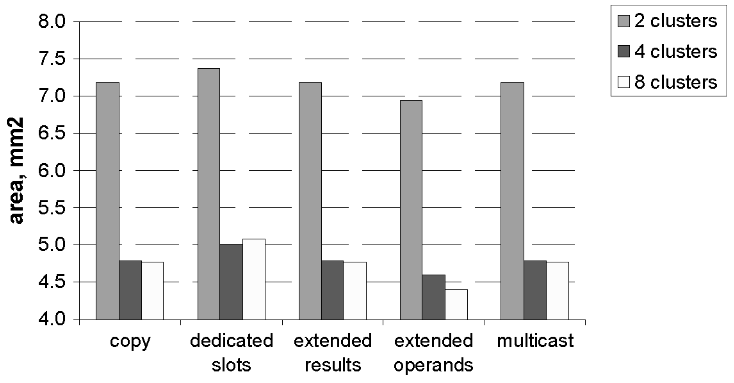

Fig. 16. Area of the clustered VLIW datapaths (unicluster: 10.6 mm^2).

unicluster to about 320 MHz for the two-cluster machines, indicating that even modest clustering may provide substantial speedups. Furthermore, the clock frequency variation among the ICC models is low, suggesting that the cycle count overhead will play the decisive role for execution time of the ICC models. Note that the copy operations, extended results, and multicast models have the same physical characteristics in our evaluation.

Processor area is an important cost factor, especially, in a chip multiprocessor, where one processor is replicated several times. In Figure 16, we present the area of the laid out clustered VLIW datapaths. Because of the huge monolithic RF of 5.4 mm^2, the unicluster was the biggest VLIW datapath occupying a total area of 10.6 mm^2. Interestingly, the two cluster configurations are still penalized by a comparatively large area. The main reason for this penalty is high routing complexity of the RFs that demanded a low row utilization. The four-cluster VLIWs, on the other hand, are compact.

According to Figure 16, the eight cluster machines do not provide (significant) area advantage over the four cluster data paths. First, this effect is caused by diminishing returns from clustering. Indeed, reducing the number of ports

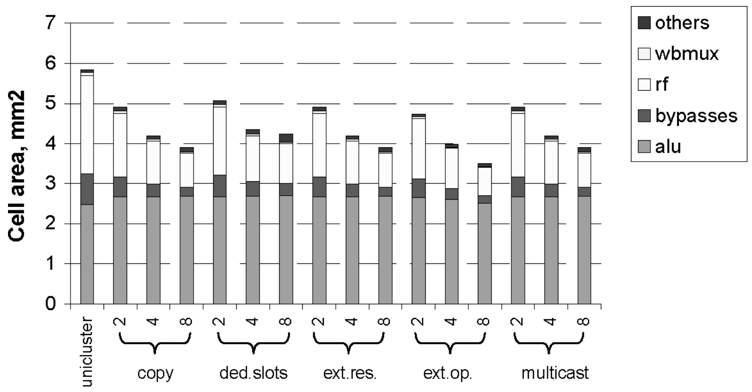

Fig. 17. Cell area breakdown of the clustered VLIW data paths.

on the registers only influences the RF multiplexer structures and not the constant area of register flip-flops. Furthermore, other components of the data path grow with clustering. For example, the ICC multiplexer, which reaches for the eight cluster machine the size of a 7-to-1 32-bit multiplexer, is instantiated eight times in the data path. Finally, our experiments favored speed over area optimizations, especially in the eight-cluster machines, where we wanted to obtain the highest clock frequency to evaluate performance limits of clustering.

Figure 17 shows the cell area breakdown of the VLIW data paths. Note that the cell area does not include area required for routing and, therefore, it is always smaller than the final layout. Especially for wire-limited RF designs, the gap between the cell area and layout area is large. Figure 17 shows that the major area savings come from reducing register files, denoted as `rf`. Although area of bypasses also benefits from clustering, its contribution to the total VLIW data path area is limited. Interestingly, the `alu` area varies across the layouts mainly because of the buffer insertion for speed optimizations performed by the synthesis tool.

5. PERFORMANCE AND ENERGY EVALUATIONS

In this section, we evaluate the ICC models in terms of execution time, energy consumption, and performance density. These three metrics are important factors in selecting proper ICC models for three embedded processor platforms—single-core media processors, mobile processors, and chip multiprocessors. Speeding up the *execution time* often dominates the design of a fast single-core media processor. Low *energy consumption* is crucial for mobile battery-operated processors, because it largely influences the operation time of a mobile device. *Performance density* is a measure quantifying computational capabilities of a processor per IC area unit (e.g., square millimeter). This measure is important for evaluation of efficiency of a processor architecture in a chip multiprocessor system, where a single processor core is replicated. Indeed, if the performance density of an architecture is low, replication of the processor will not efficiently utilize the given IC area. Therefore, the fastest single core architecture may be not the best match for a chip multiprocessor. Note, however,

Table III. Multimedia Benchmark Characteristics

Benchmark	Category	Weight	ILP	Number of DFGs	Number of C Functions	Lines of C Code
dpl2	Audio	0.5	5.1	45	12	2410
downmix	Audio	0.2	0.7	18	6	290
dtsdec	Audio	0.5	1.2	652	85	53782
mlpdec	Audio	0.5	1.2	242	35	2387
filmdetect	Video	1	6.3	24	4	1777
majorityselect	Video	1	6.2	22	5	676
median	Video	1	3.8	23	4	367
mpeg2vdec	Video	1	4.8	106	10	4555
sharpen	Video	0.2	2.6	19	4	345
rgb2cmyk	Video	0.2	7.5	12	2	209
autcor_pulse	Filter	0.2	2.4	48	6	422
fft_pulse	Filter	0.2	2.2	63	7	370
viterbi	Filter	0.2	0.7	50	6	792
mpeg2dec	Video	0	1.9	1055	114	8680
pegwit_encrypt	Encryption	0	1.4	758	97	5691
pegwit_decrypt	Encryption	0	1.4	756	97	5691
rasta	Speech	0	1.5	1441	114	7161
unepic	Image	0	1.0	725	49	3524
adpcm_decode	Audio	0	3.0	41	5	741

that on top of performance density, other issues, such as ease of programmability, extractable degree of thread-level parallelism in the applications, and predictability requirements play important roles in choosing between a single- and multicore processor architectures. However, we believe that these issues are orthogonal to our study.

5.1 Benchmarks

Our benchmark suite consisted of multimedia C applications hand-optimized for the Philips TriMedia VLIW media processor, which features a rich SIMD operation set [Van de Waerdt et al. 2005]. Furthermore, we included six nonoptimized C applications (mpeg2dec, pegwit_encrypt, pegwit_decrypt, rasta, unepic, adpcm_decode) from the Mediabench suite to observe clustering effects in out-of-the-box code. To cover a significant area of the application domain, the benchmarks were chosen from different media application categories (see Table III). Using unweighted means in benchmarking ignores relative importance of the benchmarks in a real workload [Smith 2006]. Therefore, our benchmarks were assigned weights based on the industrial practice of processor benchmarking for the TriMedia architecture. The weights indicate the performance requirements of the applications in a realistic workload of a media processor. For example, the performance-demanding video codec mpeg2vdec (weight 1.0) influences architectural decisions more than the small FFT filter fft_pulse (weight 0.2). Audio applications have medium performance requirements and, hence, have the weight of 0.5.

The optimized benchmarks underwent optimizing source-to-source transformations to increase ILP. Furthermore, the code was enhanced with 32-bit SIMD

Table IV. Baseline Unicluster Architecture

Function Units	Issue Slots	Latency
simple alu	1,2,3,4,5,6,7,8	1
alu, shifter	1,2,3,4,5,6,7,8	2
imul, fmul	1,3,5,7	6
dsp alu	2,4,6,8	3
fp alu	2,4,6,8	6
branch	1,3,5,7	9
load/store	1,3,5,7	5

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 eight issue slot VLIW machine. Note, that SIMD operations are equivalent to 2 to 5 RISC operations and, therefore, the extracted ILP rates of the optimized code are understated. This makes the actual ILP for the optimized code much higher than presented in Table III. In total, the optimization of these applications brought 10x–20x speedups with respect to the initial source code. We believe that in the embedded domain it is heavily optimized benchmarks, reflecting high utilization of the target machine, that should be used to evaluate compiler/architecture performance, as opposed to nonoptimized mostly sequential code. In the low ILP code (as used in evaluations in some other publications) the compiler does not get confronted with complex and dense data-flow graphs, and some negative effects from clustering are mitigated (e.g., cycle count explosion in deeply clustered VLIWs). Therefore, despite the fact that we included several nonoptimized benchmarks from the Mediabench suite to observe clustering effects in low ILP code, we assign weight 0 to them to avoid biased architectural decisions.

5.2 Cycle Count Performance

To reduce long simulation time required for our design space exploration, we utilized our scheduler's capability to exactly calculate the execution cycle count based on basic block execution frequencies, as described in Hekstra et al. [1999]. The execution frequencies were obtained from a single simulation run on a unicluster; the cycle counts for all other models were merely calculated based on the new instruction schedules for the models. The baseline unicluster architecture for our experiments is an eight-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 clusters contained equal functionality among each other (see Table IV). In the eight-cluster machines, though, some function units (e.g. `load/store, imul`) were available only in one-half of the clusters.

Dynamic cycle counts of the benchmarks are presented in Tables V, VI, and VII, where 100% is the cycle count of the unicluster. The cycle count primarily illustrates the overhead incurred by clustering according to the first four factors introduced in the beginning of Section 3. Each table with cycle counts is accompanied by the *weighted average* of the cycle counts according to the

Table V. Cycle Count Overhead for Two-cluster VLIW Machines

	Weights	Copy (%)	Dedicated Slots (%)	Extended Results (%)	Extended Operands (%)	Multicast (%)
`dpl2`	0.5	137.9	109.3	117.3	110.8	109.3
`downmix`	0.2	101.8	101.8	101.8	101.8	101.8
`dtsdec`	0.5	102.8	102.0	102.0	102.1	102.0
`mlpdec`	0.5	101.4	101.2	101.2	101.3	101.2
`filmdetect`	1	111.5	111.2	111.2	111.5	111.2
`majorityselect`	1	103.6	103.2	103.3	103.3	103.2
`median`	1	106.7	106.7	106.7	107.2	106.7
`mpeg2vdec`	1	121.1	105.9	108.8	111.3	105.9
`sharpen`	0.2	107.2	107.2	107.2	113.3	107.2
`rgb2cmyk`	0.2	182.2	108.5	110.6	110.8	108.5
`autcor_pulse`	0.2	107.6	103.8	103.8	103.8	103.8
`fft_pulse`	0.2	108.6	110.7	110.7	110.7	110.7
`viterbi`	0.2	100.0	100.0	100.0	100.0	100.0
`mpeg2dec`	0	103.5	101.7	102.1	102.8	101.7
`pegwit_encrypt`	0	101.1	101.0	101.0	101.2	101.0
`pegwit_decrypt`	0	101.0	101.0	101.0	101.2	101.0
`rasta`	0	104.1	103.9	104.0	104.4	103.9
`unepic`	0	103.2	103.1	103.1	103.3	103.1
`adpcm_decode`	0	104.5	104.5	104.5	104.5	104.5
W. AVERAGE		112.8	105.9	107.0	107.2	105.9

Table VI. Cycle Count Overhead for Four-cluster VLIW Machines

	Weights	Copy (%)	Dedicated Slots (%)	Extended Results (%)	Extended Operands (%)	Multicast (%)
`dpl2`	0.5	192.1	119.9	122.5	124.5	121.1
`downmix`	0.2	108.0	108.0	108.0	108.0	108.0
`dtsdec`	0.5	108.1	103.8	103.9	105.4	103.7
`mlpdec`	0.5	104.2	102.8	102.8	103.1	102.8
`filmdetect`	1	132.4	113.4	116.1	118.6	115.5
`majorityselect`	1	130.0	103.2	103.6	103.6	103.2
`median`	1	122.3	106.8	107.0	107.9	106.8
`mpeg2vdec`	1	170.7	115.0	119.5	121.1	111.0
`sharpen`	0.2	121.9	119.6	119.6	119.6	119.9
`rgb2cmyk`	0.2	277.4	114.8	138.1	150.7	119.0
`autcor_pulse`	0.2	112.7	106.4	106.4	107.1	106.2
`fft_pulse`	0.2	113.3	111.5	108.8	111.6	106.5
`viterbi`	0.2	100.8	100.7	100.7	100.7	100.7
`mpeg2dec`	0	107.3	103.3	102.7	104.0	102.6
`pegwit_encrypt`	0	103.2	101.6	101.6	101.9	101.6
`pegwit_decrypt`	0	102.8	101.5	101.5	101.7	101.5
`rasta`	0	111.7	106.9	107.8	107.4	106.8
`unepic`	0	110.6	106.7	108.2	107.5	106.7
`adpcm_decode`	0	120.4	115.9	115.9	118.1	115.9
W. AVERAGE		138.0	109.5	111.5	113.0	109.3

Table VII. Cycle Count Overhead for Eight-cluster VLIW Machines

	Weights	Copy (%)	Dedicated Slots (%)	Extended Results (%)	Extended Operands (%)	Multicast (%)
dpl2	0.5	208.5	138.9	160.2	149.6	164.2
downmix	0.2	108.0	108.0	108.0	108.0	108.9
dtsdec	0.5	115.3	109.0	110.6	110.8	113.6
mlpdec	0.5	115.0	112.2	111.6	112.6	119.2
filmdetect	1	198.3	118.3	151.8	123.8	168.1
majorityselect	1	211.1	106.5	119.8	107.1	144.9
median	1	126.1	124.6	124.3	125.3	133.8
mpeg2vdec	1	224.1	130.0	147.7	141.1	151.3
sharpen	0.2	137.2	133.2	133.5	139.2	134.7
rgb2cmyk	0.2	362.2	144.3	251.7	176.0	272.9
autcor_pulse	0.2	128.7	113.8	119.2	117.5	116.7
fft_pulse	0.2	124.4	111.7	116.1	120.0	111.5
viterbi	0.2	104.6	104.4	104.4	104.4	102.2
mpeg2dec	0	114.4	106.1	110.1	107.3	112.2
pegwit_encrypt	0	121.6	107.3	110.7	107.3	112.0
pegwit_decrypt	0	122.9	110.5	112.1	109.4	118.3
rasta	0	121.9	110.1	115.2	112.2	116.3
unepic	0	118.9	111.9	113.4	112.0	116.6
adpcm_decode	0	138.6	131.8	156.7	134.1	131.8
W. AVERAGE		174.9	119.8	134.5	124.9	144.2

following formula:

$$W\_AVERAGE = \frac{\sum_{i \in all\_benchmarks} weight_i \cdot cycle\_count_i}{\sum_{i \in all\_benchmarks} weight_i} \quad (2)$$

Later, in Figure 18, we combine the weighted averages of cycle count overheads for all models and all cluster configurations.

The tables above show that the ICC models have distinctly different cycle counts, which is mainly conditioned on the corresponding architectural constraints of the ICC models. Mediabench benchmarks are much less sensitive to clustering, showing low overhead numbers. This is mainly caused by low utilization of the VLIW (ILP), which could have exaggerated the final performance benefits from clustering, unless the weights were 0.0. Noteworthy, some benchmarks (e.g., rgb2cmyk and dpl2) from the above tables show a drastic cycle count overhead. Their cycle counts are mainly defined by small high ILP loop, which get severely penalized by clustering.

Figure 18 presents cycle count overheads of clustered machines relative to the unicluster. Two series of overheads are presented—the low bandwidth of one intercluster transfer per cluster (denoted as BWL) and the doubled high bandwidth of two intercluster transfers per cluster (denoted as BWH). Among the ICC models, the *copy operation* clearly performs the worst. The main factor of this performance degradation is that copy operations reside in regular VLIW slots, which interfere with scheduling of regular operations. No surprise that the *dedicated issue slots* model performs the best in terms of cycle count. This

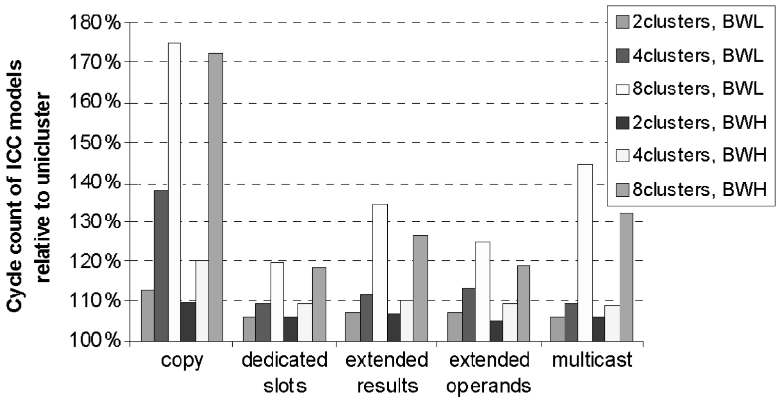

Fig. 18. Weighted averages of the cycle count overheads for low and high ICC bandwidth networks.

architecture boasts a very modest cycle count overhead for all cluster configuration as a result of extra resources solely dedicated to intercluster communication. Figure 18 clearly shows that the growing number of clusters results in a significant cycle count overhead, especially for the eight-cluster configurations. The latter suffers from insufficient issue slots per cluster, forcing a lot of expensive intercluster communication. On top of that, the distributed RF architectures with eight clusters lack registers because of value duplications, which badly hurts performance in case of register spilling.

For bandwidth sensitivity analysis, we increased both the input and output bandwidth from one to two transfers per cycle per cluster. Note, that the eight-cluster configurations of the copy operation, extended results, and multicast models do not naturally support the output bandwidth higher than the number of slots in each cluster (i.e., one). Therefore, these clustered VLIW configurations could only benefit from the higher input bandwidth. Figure 18 shows the copy operation model benefits the most from the increased bandwidth (e.g., for the four-cluster VLIW the overhead drops from 38 to 21%). This is caused by increased scheduling freedom for intercluster communication, which in the low-bandwidth case was severely limited by a single VLIW issue slot per cluster. To further benefit from this effect (especially, in the eight-cluster VLIW), a hybrid model with a higher bandwidth can be formed by combining the copy operations model, for example, with extended operands. Furthermore, Figure 18 shows that the eight-cluster VLIWs profit more from higher bandwidth than the two-cluster configurations, which is conditioned on the presence of more intensive ICC in deeply clustered machines requiring higher bandwidth. In general, our bandwidth sensitivity analysis shows only a moderate overhead reduction. According to Table I, the increased bandwidth incurs a higher number of RF ports, which may negatively contribute to the clock frequency of the complete VLIW datapath. In view of the modest cycle count reduction, our subsequent experiments are carried out with the minimum bandwidth, aiming at achieving the highest clock frequency.

Although multicast is a very promising mechanism, in our experiments it performs well only on two- and four-cluster machines. Multicast allows to

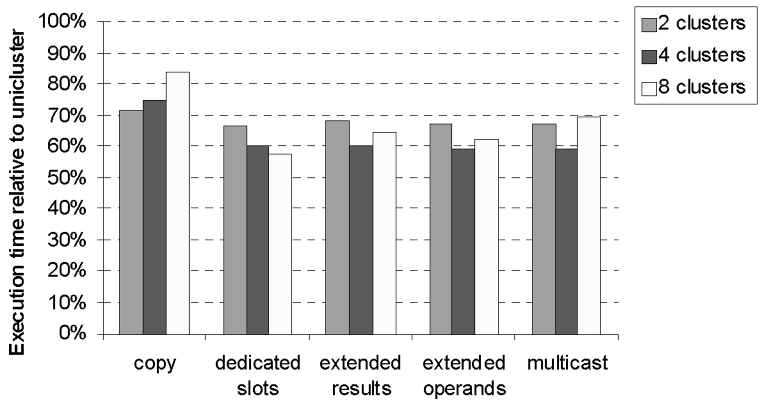

Fig. 19. Execution time relative to the unicluster.

simultaneously carry out multiple intercluster data transports. However, as explained in our microarchitecture description section of the model, the incoming ICC writes get multiplexed onto a single write port of the RF. This obviously limits the number of destinations of a multicast. Furthermore, our scheduler does not employ any advanced techniques to optimize the scheduling of multicasts, which further confines the performance of this model in our experiments. In fact, this leads, in the eight-cluster architectures, to a paradox that the potentially less powerful extended results model outperforms the multicast. We discovered, that this effect is caused by the greedy nature of multicasting in our scheduler, which quickly saturates the small RFs in the eight-cluster machine, causing an explosion of spill/restore code.

5.3 Execution Time, Performance Density, and Energy Consumption

The execution time being a product of the cycle count (evaluated in Section 3) and clock period (measured in Section 4) is shown in Figure 19. First, we observe that clustering obviously drastically reduces the execution time by up to 42%. Note that the popular *copy operation* model performs the worst and, remarkably, increasing the number of clusters beyond two results in slower VLIW machines. The extra resources for intercluster communication in the *dedicated issue slots* do not hamper the clock frequency and, thus, this model performs the best. Moreover, the *dedicated slots* model has a positive trend of increasing performance with more aggressive clustering; however, the execution speedup is diminishing for higher number of clusters. Therefore, the *dedicated slots* model is the best candidate for a single processor core IC targeting at high-performance applications. Other models (*extended results*, *extended operands*, and *multicast*) perform similarly to each other. Noteworthy for these models, clustering above four clusters degrades the performance because of a high cycle count overhead in the eight-cluster machines.

Figure 19 presents performance characteristics measured for clustered VLIW architectures with point-to-point ICC networks. In fact, the presented ICC models can be implemented with either point-to-point or *bus-based*

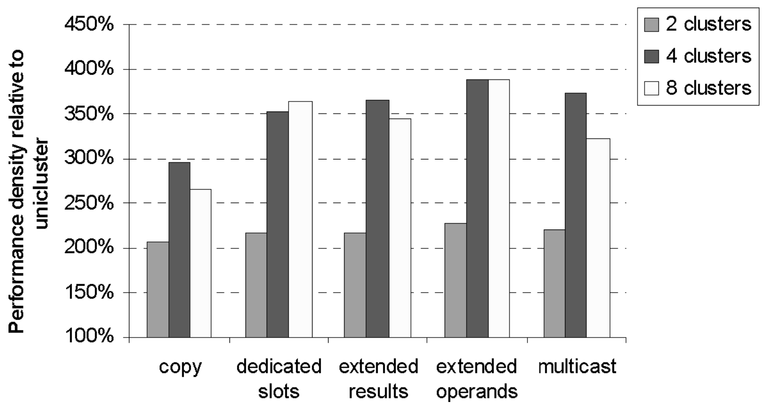

Fig. 20. Performance densities relative to the unicluster.

interconnects. The point-to-point interconnect in our experiments occupied a negligible area of less than 0,1%, which does not motivate for the employment of a bus-based model from the area standpoint. From the architecture perspective, the intercluster transport is more constrained in a bus-based system than in a point-to-point network, because of wire sharing in the former. At the same time, the bus model does not decrease the number of RF and bypass ports, and, consequently, no clock frequency improvement in the datapath is achieved. Therefore, the bus-based interconnect is unlikely to improve performance of clustered VLIW architectures, unless the point-to-point interconnect becomes not routable. We expect inter-cluster routability issues to appear only in massively parallel architectures with many clusters, which do not match our media application domain. Note, that Gangwar et al. [2005] and Parcerisa et al. [2005] also report, that the bus-based model underperforms compared to point-to-point networks.

In Figure 20 we present performance densities of the intercluster communication models. In our evaluation performance, density is the reciprocal of the product of execution time and area: $PD = 1/(E \cdot A)$, normalized to the performance density of the unicluster. Hence, the performance density of the unicluster is 100%. The *extended operands* model is a clear winner, achieving a 3.9x better performance density than the unicluster. Therefore, this architecture is a good candidate for forming a chip multiprocessor. Note, that the *dedicated issue slots* architecture had the highest absolute performance, but due to the area penalty, it is less efficient as a basis of a chip multiprocessor. Interestingly, the eight-cluster machines often perform worse than the four-cluster VLIWs as a result of the high cycle count overhead in the former.

Energy consumption (or the product of power dissipation and execution time) of a clustered VLIW processor is determined by a complex tradeoff. On one hand, the smaller clusters dissipate less power. On the other hand, the program takes longer to execute on a clustered machine (running at the same clock frequency) and, consequently, causes higher switching activity than the unicluster. To quantify for this tradeoff, we conducted power simulations of all

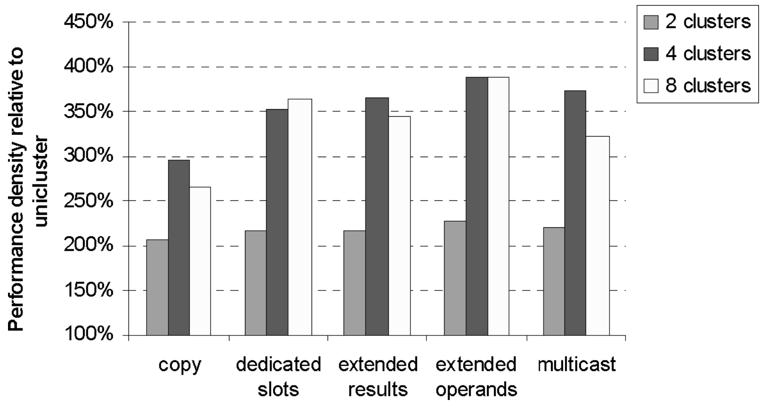

Fig. 21. Energy consumption relative to the unicluster (fixed program; shortest execution time).

our (clustered) VLIW machines at the RTL level using Synopsys Power Compiler V-2004.06 under worst-case conditions of the process, voltage 1.2 V, and temperature 125°C in CMOS 130-nm technology. The total of static and dynamic power dissipation was then multiplied by the execution time from Figure 19, yielding energy consumption of the clustered machines shown in Figure 21. Thus, Figure 21 presents energy for all VLIWs required to execute a set of *fixed programs* in a *shortest possible time*. Burd and Brodersen [2002] term this mode as a "maximum throughput" (of operations). The power consumption of the 200 MHz unicluster in our measurements reached 234 mW, including 1.7 mW of leakage power. Figure 21 shows that for most of the ICC models the lower power dissipation of smaller clusters outweighs the higher switching activity rates. In other words, despite being faster, most of the clustered VLIWs consume less energy than the corresponding unicluster. However, the *copy operation* model in the eight-cluster configuration turns out to consume substantially more energy than the unicluster because of the exploded cycle count, yet again emphasizing the importance of choosing a proper ICC model. Furthermore, all fully distributed RF architectures (with eight clusters in our experiments) consume more energy than modestly clustered machines, clearly indicating that extensive clustering is not energy-efficient.

Figure 21 shows energy consumption for the VLIWs executing the same program in a shortest possible time. However, for real-time media processing (e.g., video decoding) there is no need to process media faster than the real time (e.g., to decode frames faster than the video frame rate). In fact, we can trade the speed surplus of clustered VLIWs for energy. Indeed, if we assume that the unicluster is fast enough for real-time processing, we can measure energy consumption of clustered VLIWs required to execute the *fixed program* in a *fixed execution time*. Burd and Brodersen [2002] term this mode as a "fixed throughput" (of operations). Let the fixed execution time be equal to the execution time of our program on the uniprocessor. Subsequently, we can reduce the clock frequency of the clustered VLIWs, since they do not have to run faster. However, to save energy, we should also reduce the supply voltage V_{dd} [Burd and Brodersen 2002]. Note that reducing V_{dd} limits the highest achievable clock frequency.

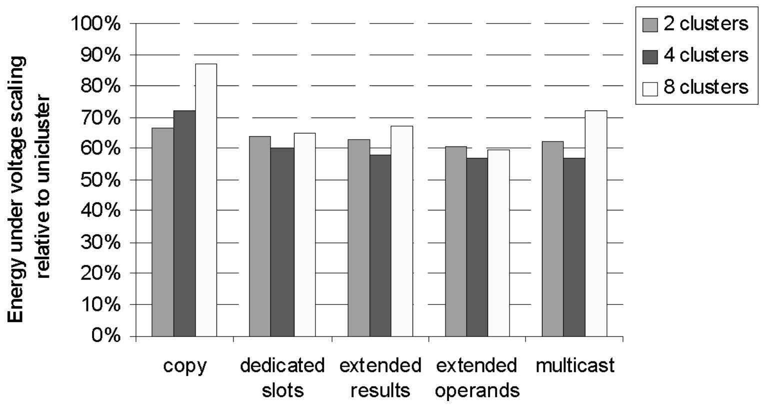

Fig. 22. Voltage scaling effect on energy consumption (fixed program; fixed execution time).

Consider the following energy equation:

$$Energy = Power \cdot ExecutionTime \approx (C_{eff} \cdot V_{dd}^2 \cdot f)\left(\frac{N_{cycles}}{f}\right) = C_{eff} V_{dd}^2 N_{cycles} \qquad (3)$$

where C_{eff} is the effective switched capacitance, V_{dd} is the supply voltage, and $N_{cycles}$is the number of cycles required to execute our fixed program. In Equation (3), we neglected leakage and short-circuit power dissipation, since it was less than 1% of the total power dissipation for our technology. C_{eff} and N_{cycles} will remain unchanged if the V_{dd} is scaled. Therefore, based on simulated energy from Figure 21 and Equation (3), we can calculate energy consumption of the clustered VLIWs running at 1.0 V relative to the unicluster at 1.2 V (see Figure 22). Using CMOS 130-nm technology data sheets for voltage and temperature derating factors, we verified that the clock frequencies (required to achieve our fixed execution time on clustered processors) were indeed achievable at the lowest 1.0 V supply voltage of our technology. Note that we cannot drop voltage for the unicluster, since, otherwise, it will run at a lower clock frequency and will not manage to execute the fixed program in our fixed execution time [Chandrakasan and Brodersen 1995].

With voltage scaling, all clustered machines consume significantly less energy than the unicluster. Furthermore, the energy savings now reach a substantial factor of 1.75 times for the four-cluster VLIWs of the *extended operands* models. Therefore, this ICC model is a good candidate for battery-operated embedded processors in a mobile device. In general, the presented reduction of energy consumption could be further improved if our microarchitecture and RTL are tuned for low power rather than for speed. For example, the hardware can be extended with clock gating or power shut-down of unused clusters, function units, and registers.

6. RELATED WORK

Many prior studies in clustered VLIW architectures based their research on one intercluster communication model, typically, in the form of copy operations.

Our study, in contrast, analyzes a *large taxonomy of intercluster communication models* in terms of execution time and VLSI characteristics. This research extended our previous publications in the following directions. Instead of the hypothetical 64-bit CPU64 architecture [van Eijndhoven et al. 1999] used in our previous studies, which has never been implemented in VLSI, we employed the 32-bit ISA and the deep VLIW pipeline from the commercial Philips TriMedia TM5250 VLIW core [Halfhill 2004]. Furthermore, on top of the VLSI layouts of the *copy operations* ICC model from Terechko et al. [2005] we designed and laid out hardware for the other four ICC models in CMOS 130-nm IC technology. Moreover, we conducted experiments to quantify the energy consumption trade-off for all the evaluated clustered VLIW machines. Instead of the register-hungry *broadcast* ICC model from Terechko [2003a], we employed the more promising *multicast* ICC model. Finally, cluster assignment heuristics of global values presented in Terechko et al. [2003b] have been extended from the *dedicated issue slot* machine to *extended results*, *extended operands*, *copy operations*, and *multicast* ICC models. In general, all the above-mentioned extensions enabled a comprehensive comparison of all the models within the same ISA, compiler optimizations, and hardware template.

6.1 Alternative Clustered Architectures

In partitioned ILP architectures [Janssen 2001], the register file is split in a similar way to the clustered architectures presented in our work. However, the bypass network remains shared among all issue slots, which may negatively impact the clock frequency of the processor. On top of clustering the data path of an ILP processor, there has been substantial work published on *clustering the memory hierarchy* [Gibert et al. 2002, 2003, 2005]. The ideas span the range of distributed caches with interleaved addressing [Gibert et al. 2002] to compiler-managed L0 buffers [Gibert et al. 2003] to cache coherence support [Gibert et al. 2005]. Furthermore, several elaborate concepts are developed for *hierarchical* clustered RF architectures and instruction-scheduling algorithms for them [Zalamea et al. 2003, Rixner et al. 1999, Kapasi et al. 2002].

6.2 Inter-cluster Communication Models

Gangwar et al. [2003] further developed our initial taxonomy along with advanced-scheduling algorithms for clustered VLIWs. In 2003, they expand into *hybrid ICC models*, combining several simpler ICC models. Furthermore, on top of our point-to-point ICC model evaluations, they studied *bus-based ICC models* in more detail Gangwar et al. [2005], concluding that it underperforms relative to the point-to-point models. An important auxiliary study performed in Gangwar et al. [2005] was the layout-based analysis of pipeline registers in the ICC paths. The outcome was that the pipeline registers are essential for achieving high clock frequencies for VLIW machines with more than two clusters. For example, VLIW machines with a high number of clusters may require a higher ICC bandwidth to sustain higher performance. In 2003, Gangwar et al. quantify the impact of the bandwidth parameter on clustered VLIW performance. Furthermore, Codina et al. [2001] proposed the *memory-based*

ICC mechanism, which can be used if the regular ICC means, such as a bus or a point-to-point network, is overloaded. Another interesting option to reduce intercluster communication has been presented by Aletà et al. [2003], where, instead of communicating data, it is recomputed in the destination cluster, which results in substantial performance speedups.

In the multicast ICC model from Kailas et al. [2001, 2002], a special broadcast operation `sendb` was proposed to communicate between clusters, which is scheduled the same way as our *multicast*. Unfortunately, no feasibility study demonstrated clock frequency implications of their caching register buffer, and, hence, no conclusion was drawn on whether this ICC model provides substantial performance improvement. We feel that scheduling heuristics for `sendb` and our *multicast* presented by Kailas et al. can be improved to enable more effective use of this operation, because potentially the *multicast* operation can better utilize the ICC bandwidth than the other models by triggering several intercluster transfers of the same value.

In the early 1980s, Fisher pioneered the design of clustered VLIW processors with the Multiflow Trace mini-supercomputer. Later, Fisher et al. [2004] accumulated an impressive expertise on clustered VLIWs, including advanced instruction-scheduling algorithms and cycle count evaluations using optimized benchmarks. Their effort resulted in a commercial design of the ST2xx VLIW processor, but investigation of diverse ICC models was not pursued. Furthermore, there exist studies of *partially connected* clustered ILP processors [Colavin and Rizzo 2003; Roos et al. 2002; Veredas et al. 2005], which did not gain popularity because of deadlock issues in the scheduling process and register allocation.

6.3 Instruction Scheduling for Clustered VLIW Architectures

Related studies focusing on integrating cluster-assignment heuristics into existing optimizing ILP compilers can be categorized into *acyclic* [Ellis 1985; Özer et al. 1998; Kailas et al. 2001; Chu et al. 2003; Nagpal and Srikant 2004] and *cyclic* scheduling algorithms [Codina et al. 2001; Aletà et al. 2002, Zalamea et al. 2003; Lapinskii et al. 2002; Gibert et al. 2005]. Cyclic algorithms typically apply a software pipelining technique to cyclic data-dependency graphs (loops), while acyclic schedulers concentrate on global scheduling involving code motion beyond control points in acyclic data-dependency graphs. To combat the notorious phase-coupling problem, some instruction schedulers merge cluster assignment, register allocation and instruction scheduling in a *single phase* [Kailas et al. 2001; Codina et al. 2001]. The research group at the Universitat Politècnica de Catalunya greatly contributed to innovations on (software pipelined) instruction scheduling and microarchitecture for clustered ILP processors [Aletà et al. 2002; 2003; Codina et al. 2001; Gibert et al. 2005; Parcerisa et al. 2005; Zalamea et al. 2003]. For example, in Parcerisa et al. [2005], they present a comprehensive analysis of ICC models for superscalar processors. Remarkably, just as Gangwar, they also conclude that point-to-point ICC networks (considered in our publication) outperform the bus-based implementations. Unfortunately, there exist very few studies on optimization of cluster assignment of values that are alive across scheduling units (e.g. regions) [Kailas et al. 2001], whereas our prior research [Terechko et al. 2003b] indicates potentially high performance gains in this area.

6.4 Characterization of VLSI Properties of Clustered VLIW Architectures

From the *VLSI implementation* perspective, our study resorts to actual layout implementation of the complete data paths, in contrast to many prior research works judging clock frequency improvements based on analytical models of register files presented in Rixner et al. [1999]. Our layout experiments demonstrate that the bypass network often is slower than the easily pipelined RF and, hence, the clock frequency of the distributed RF architectures was often exaggerated in prior works that neglected the bypasses. Furthermore, we focus on the standard cell design popular in embedded computing, instead of full custom designs of PC and server processors [Palacharla et al. 1997].

7. CONCLUSIONS

Our experiments demonstrate that clustering can bring a substantial 1.74 execution time speedup for a deeply pipelined wide eight-issue slot VLIW processor and drastically reduce the energy consumption and area by a factor of 1.75 and a factor of 1.81, respectively. However, to achieve these improvements from clustering a processor architect has to thoroughly select a proper *Inter-cluster Communication* model. For example, we show that copy operations of the popular *copy operation* ICC model severely hamper scheduling of regular operations, and, therefore, aggressive clustering only worsens the performance. On the other hand, the *dedicated issue slots* model, where ICC is carried out by dedicated VLIW issue slots, performs the best, and consistently boosts performance with clustering. If the code size of this model is acceptable for a particular design, then this model appears to be the best choice for fast single-core VLIW processors. In contrast to prior art, our research shows that fully distributed RF architectures (in our study with eight clusters) often underperform compared to modestly clustered (two to four clusters) machines resulting from explosion of the cycle count overhead in the former. Interestingly, the lower power dissipation of smaller clusters in the trade-off for energy consumption outweighs the higher cycle counts, resulting in lower energy dissipation of most of the evaluated clustered VLIWs relative to the unicluster. Moreover, the higher speed of the clustered VLIWs can be traded for even lower energy consumption with the help of voltage scaling. However, extensive clustering turns out to be not energy efficient. The energy savings are the highest for the frugal *extended operands* model, making it the best candidate for battery-operated VLIW processors. Furthermore, the *extended operands* model is a good building block for chip multiprocessor systems thanks to its high performance density.

Another important conclusion we draw is that only realistic experimentation unveils crucial aspects of clustering. For example, revealing the explosion of the cycle-count overhead for the *copy operation* model and distributed RF architectures was conditioned on the usage of optimized full C applications for our instruction-scheduling experiments instead of out-of-the-box code with low ILP. Furthermore, only the VLSI layout experiments demonstrated that the bypass network in clustered VLIWs is often slower than the register files, which are easy to pipeline.

Future research includes exploration of binary instruction formats for the ICC models with related code size, clustering of memory hierarchy, and advanced scheduling algorithms for the promising *multicast* model.

ACKNOWLEDGMENTS

The authors thank E. Le Thénaff, M. Garg, J.T.J. van Eijndhoven, and E.J.D. Pol for their valuable contributions to the presented research.

REFERENCES

AGARWAL, V., HRISHIKESH, M. S., KECKLER, S. W., AND BURGER, D. 2000. Clock Rate versus IPC: The end of the road for conventional microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, Vancouver, Canada, IEEE Computer Society Press, Los Alamitos, CA. 248–259.

ALETÀ, A., CODINA, J. M., SANCHEZ, F. J., GONZÁLEZ, F. J., AND KAELI, D. R. 2002. Exploiting pseudo-schedules to guide data dependence graph partitioning. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Charlottesville, VA. IEEE Computer Society Press, Los Alamitos, CA. 281–290.

ALETÀ, A., CODINA, J. M., GONZÁLEZ, F. J., AND KAELI, D. R. 2003. Instruction replication for clustered microarchitectures. In *Proceedings of the 36th Annual International Symposium on Microarchitecture*, San Diego, CA. IEEE Computer Society Press / ACM Press, New York. 326–335.

BEKOOIJ, M. 2004. Constraint Driven Operation Assignment for Retargetable VLIW Compilers. PhD thesis, ISBN 90-74445-60-8, Technical University of Eindhoven, Eindhoven, The Netherlands.

BURD, T. D. AND BRODERSEN, R. W. 2002. *Energy Efficient Microprocessor Design*. Kluwer Academic Publishers, Novell, MA.

CHANDRAKASAN, A. P. AND BRODERSEN, R. W. 1995. Low power digital CMOS design. Kluwer Academic Publishers, Novell, Massachusetts.

CHU, M., FAN, K., AND MAHLKE, S. 2003. Region-based hierarchical operation partitioning for multicluster processors. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, San Diego, California. ACM Press, New york. 300–311.

CODINA, J. M., SÁNCHEZ, J., AND GONZÁLEZ, A. 2001. A unified modulo scheduling and register allocation technique for clustered processors. In *Proceedings of the 10th International Conference on Parallel Architecture and Compilation Techniques*, Barcelona, Spain. IEEE Computer Society Press, Los Alamitos, CA. 175–184.

COLAVIN, O. AND RIZZO, D. 2003. A scalable wide-issue clustered VLIW with a reconfigurable interconnect. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, San Jose, CA. ACM Press, New York. 148–158.

ELLIS, J. R. 1985. *Bulldog: A Compiler for VLIW Architectures*. MIT Press, Cambridge, MA.

FARABOSCHI, P., BROWN, G., FISHER, J. A., DESOLI, G., HOMEWOOD, F. 2000. Lx: A technology platform for customizable VLIW embedded processing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, Vancouver, Canada. IEEE Computer Society Press, Los Alamitos, CA. 203–213.

FISHER, J. A. 1981. Trace scheduling: A technique for global microcode compaction. *IEEE Transactions on Computers*, 478–490.

FISHER, J. A., FARABOSCHI, P., AND YOUNG, C. 2004. Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools. Morgan Kaufmann. San Francisco, CA.

GANGWAR, A., BALAKRISHNAN, M., AND KUMAR, A. 2003. Impact of Inter-cluster Communication Mechanisms on ILP in Clustered VLIW Architectures, In *Proceedings of the 2nd Workshop on Application Specific Processors*, San Diego, CA.

GANGWAR, A., BALAKRISHNAN, M., RANJAN PANDA, P., AND KUMAR, A. 2005. Evaluation of bus based interconnect mechanisms in clustered VLIW architectures. In *Proceedings of the Design Automation and Test in Europe*, Munich, Germany, IEEE Computer Society Press/ACM Press, New York. 730–735.

GIBERT, E., SÁNCHEZ, J., AND GONZÁLEZ, A. 2002. Effective instruction scheduling techniques for an interleaved cache clustered VLIW processor. In *Proceedings of the 35th International Symposium on Microarchitecture*, Istanbul, Turkey, IEEE Computer Society Press/ACM Press, New York. 123–133.

GIBERT, E., SÁNCHEZ, J., AND GONZÁLEZ, A. 2003. Flexible compiler-managed L0 buffers for clustered VLIW processors. In *Proceedings of the 36th Annual International Symposium on Microarchitecture*, San Diego, CA, USA, IEEE Computer Society Press/ACM Press, New York. 315–325.

GIBERT, E., SÁNCHEZ, J., AND GONZÁLEZ, A. 2005. Distributed data cache designs for clustered VLIW processors. *IEEE Transactions on Computers, 54*, 10, 1227–1241.

HALFHILL, T. R. 2004. Best media processor: TriMedia TM5250. *Microprocessor Report*, 2/9/04, http://www.mpronline.com.

HAVANKI, W. A., BANERJIA, S., AND CONTE, T. M. 1998. Treegion scheduling for wide-issue processors. In *Proceedings of 4th International Symposium on High Performance Computer Architecture*, Las Vegas, NV. IEEE Computer Society Press. 266–276.

HEKSTRA, G. J., LA HEI, G. D., BINGLEY, P., AND SIJSTERMANS, F. W. 1999. TriMedia CPU64 design space exploration. In *Proceedings of the International Conference on Computer Design*, Austin, USA, IEEE Computer Society Press, Los Alamitos, CA. 599–606.

HO, R., MAI, K., AND HOROWITZ, M. 2001. The future of wires. *Proceedings of the IEEE, 89*, 4, 490–504.

HOOGERBRUGGE, J. AND AUGUSTEIJN, L. 1999. Instruction scheduling for TriMedia. *The Journal of Instruction-Level Parallelism, 1*, http://www.jilp.org/.

HSU, P. Y. T. AND DAVIDSON, E. S. 1986. Highly concurrent scalar processing. In *Proceedings of 13th Annual International Symposium on Computer Architecture*, Tokyo, Japan. 386–395.

ITRS TECHNOLOGY WORKING GROUPS. 2005. *International Technology Roadmap for Semiconductors (ITRS)*. The ITRS Technology Working Groups. http://www.itrs.net/.

JANSSEN, J. 2001. *Compiler Strategies for Transport Triggered Architecture*. PhD thesis, Technical University of Deflt, The Netherlands.

KAILAS, K., EBCIOGLU, K., AND AGRAWALA, A. K. 2001. CARS: A new code generation framework for clustered ILP processors. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture*, Nuevo Leone, Mexico. IEEE Computer Society Press, Los Alamitos, CA. 133–134.

KAILAS, K., FRANKLIN, M., AND EBCIOGLU, K. 2002. A register file architecture and compilation scheme for clustered ILP processors. In *Proceedings of the 8th International Conference Euro-Par*, Paderborn, Germany, Lecture Notes in Computer Science, Springer. 500–511.

KAPASI, U. J., DALLY, W. J., RIXNER, S., OWENS, J. D., AND KHAILANY, B. 2002. The imagine stream processor. In *Proceedings of the 20th International Conference on Computer Design*, Freiburg, Germany. IEEE Computer Society, Los Alamitos, CA. 282–288.

LAM, M. S. AND WILSON, R. P. 1992. Limits of control flow on parallelism. In *Proceedings of the 19th International Symposium on Computer Architecture*, Queensland, Australia. ACM Press, New York. 46–57.

LAPINSKII, V. S., JACOME, M. F., AND DE VECIANA, G. A. 2002. Application-specific clustered VLIW datapaths: Early exploration on a parameterized design space. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 21*, 8, 889–903.

LIAO, H. AND WOLFE, A. 1997. Available parallelism in video applications. In *Proceedings of the 30th International Symposium on Microarchitecture*, Research Triangle Park, North Carolina. ACM Press/IEEE Computer Society Press, New York. 321–329.

LEE, W., BARUA, R., FRANK, M., SRIKRISHMA, D. 1998. Space-time scheduling of instruction-level parallelism on a RAW machine. In *Proceedings of 8th International Conference on Architectural*

Support for Programming Languages and Operating Systems, San Jose, CA. ACM Press, New York. 46–57.

Lee, H. H. S., Wu, Y. and Tyson, G. S. 2000. Quantifying instruction-level parallelism limits on an EPIC architecture. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, Austin, TX. IEEE Computer Society Press, Los Alamitos, CA. 21–27.

Levy, M. 2001. ManArray devours DSP code. *Microprocessor report*, 10/8/01-01, http://www.mpronline.com/.

Mahlke, S. A., Lin, D. C., Chen, W. Y., Hank, R. E. and Bringmann, R. A. 1992. Effective compiler support for predicated execution using the hyperblock. In *Proceedings of the 25th Annual International Symposium on Microarchitecture*, Portland, Oregon. IEEE Computer Society/ACM Press, New York. 45–54.

Nagpal, R., and Srikant, Y. N. 2004. Integrated temporal and spatial scheduling for extended operand clustered VLIW processors. In *Proceedings of the International Conference on Computing Frontiers*, Ischia, Italy. ACM Press, New York. 457–470.

Özer, E., Banerjia, S., and Conte, T. 1998. Unified assign and schedule: a new approach to scheduling for clustered register file microarchitectures. In *Proceedings of the 31st IEEE/ACM Annual International Symposium on Microarchitecture*, Dallas, Texas. IEEE Computer Society Press, Los Alamitos, CA. 308–315.

Palacharla, S., Jouppi, N. P., and Smith, J. E. 1997. Complexity-effective superscalar processors. In *Proceedings of the 24th International Symposium on Computer Architecture*, Denver, CO. 206–218.

Parcerisa, J.-M. L, Sahuquillo, J., González, and A., Duato, J. 2005. On-chip interconnects and instruction steering schemes for clustered microarchitectures. *IEEE Transactions on Parallel and Distributed Systems, 16*, 2, 130–144.

Rixner, S., Dally, W. J., Khailany, B., Mattson, P., Kapasi, U. J., Owens, J. D. 1999. Register organization for media processing. In *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, Toulouse, France. IEEE Computer Society, Los Alamitos, CA. 375–386.

Roos, S., Corporaal, H., and Lamberts, R. 2002. Clustering on the Move. In *Proceedings of the 4th International Conference on Massively Parallel Computing Systems*, Ischia, Italy, IEEE Computer Society Press, Los Alamitos, CA.

Smith, J. E. 2006. Benchmarking: Science? Art? Neither? In *2006 SPEC Benchmark Workshop,* Austin, Texas. http://www.spec.org/workshops/2006/.

Sudharsanan, S., Sriram, P., Frederickson, and H., Gulati, A. 2000. Image and video processing using Majc 5200. In *Proceedings of the International Conference on Image Processing*, Vancouver Canada, IEEE Computer Society Press, Los Alamitos, CA. 122–125.

Terechko, A. S., Le Thénaff, E., van Eijndhoven, J. T. J., and Corporaal, H. 2003a. Inter-cluster communication models for clustered VLIW processors. In *Proceedings of the 9th Symposium High Performance Computer Architectures*, Anaheim, CA. IEEE Computer Society Press, Los Alamitos, CA. 354–364.

Terechko, A. S., Le Thénaff, E., and Corporaal, H. 2003b. Cluster assignment of global values for clustered VLIW processors. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, San Jose, CA. ACM Press, New York. 32–40.

Terechko, A. S., Garg, M., and Corporaal, H. 2005. Evaluation of speed and area of clustered VLIW processors. In *Proceedings of 18th International Conference on VLSI Design* in conjunction with *the 4th International Conference on Embedded Systems Design*, Kolkata, India, IEEE Computer Society Press, Los Alamitos, CA. 557–563.

van Eijndhoven, J. T. J., Vissers, K. A., Pol, E. J. D., Struik, P., Bloks, R. H. J., van der Wolf, P., Vranken, H. P. E., Sijstermans, F. W., Tromp, M. J. A., and Pimentel, A. D. 1999. TriMedia CPU64 architecture. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Austin, Texas. IEEE Computer Society Press, Los Alamitos, CA. 586–592.

van De Waerdt, J.-W., Vassiliadis, S., Das, S., Mirolo, S., Yen, C., Zhong, B., Basto, C., van Itegem, J.-P., Amirtharaj, D., Kalra, K., Rodriguez, P., and van Antwerpen, H. 2005. The TM3270 media-processor. In *Proceedings of 38th Annual IEEE/ACM International Symposium on Microarchitecture*, Barcelona, Spain, IEEE Computer Society Press/ACM Press, New York. 331–342.

VEREDAS, F. J., SCHEPPLER, M., MOFFAT, W., AND MEI, B. 2005. Custom implementation of the coarse-grained reconfigurable ADRES architecture for multimedia purposes. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, Tampere, Finland. IEEE Computer Society Press, Los Alamitos, CA. 106–111.

ZALAMEA, J., LLOSA, J., AYGUADE, E., AND VALERO, M. 2003. Hierarchical clustered register file organization for VLIW processors. In *Proceedings of 17th International Parallel and Distributed Processing Symposium*, Nice. IEEE Computer Society Press, Los Alamitos, CA. 77a.

Received March 2006; revised July 2006 and October 2006; accepted October 2006