The CURE: Cluster Communication Using Registers

VIGNYAN REDDY KOTHINTI NARESH, Qualcomm Technologies Incorporated
DIBAKAR GOPE and MIKKO H. LIPASTI, University of Wisconsin Madison

VLIW processors typically deliver high performance on limited budget making them ideal for a variety of communication and signal processing solutions. These processors typically need large multi-ported register files that can have side effects of increased cycle time and high power consumption. The access delay and energy of these register files can also become prohibitive when increasing the register count or the access ports, thus limiting the overall performance of the processor. Most prior art circumvent this problem by using multiple clusters with private register files, to lower the access delay and reduce energy consumption. However, clustering artifacts, like increased inter–cluster communication operations and spill-recovery code, result in a performance penalty.

This paper proposes CURE — a novel technique to considerably reduce the negative effects of clustering. CURE augments the ISA to expose the communication registers to the compilers to increase availability of architectural register state to all functional units. The inter–cluster communication operations are integrated into regular ALU and memory operations to improve instruction encoding efficiency. We also propose a new code scheduling heuristic to handle the ISA changes, and to realize the improvements in processor's performance and energy consumption. Our quantitative analysis estimates that CURE, when compared to the baseline 8–issue uni–cluster processor, boosts average performance by 61% while reducing the average register dynamic energy by 77%.

Q1 CCS Concepts:

Q2 Additional Key Words and Phrases:

ACM Reference Format:
Vignyan Reddy Kothinti Naresh, Dibakar Gope, and Mikko H. Lipasti. 2017. The CURE: Cluster Communication Using Registers. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 124 (August 2017), 19 pages.
https://doi.org/10.1145/3126527

1 INTRODUCTION

Very long instruction word (VLIW) processors are widely used, despite the sparse research activity. Modern digital signal processors (DSPs), like the CEVA X series [20] or the Qualcomm Hexagon DSP [10] or Tensilica's BBE series [33], use VLIW architecture to achieve performance targets within restricted area and power budgets. Many other older, yet relevant DSPs [15, 16, 18, 23] and server processors [25] have been using VLIW architectures. These processors are ubiquitous in modem and multimedia applications such as computer vision, augmented reality, object recognition, high definition audio, sensor processing, image enhancement, machine learning and baseband

This article was presented in the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES) 2017 and appears as part of the ESWEEK-TECS special issue.
Vignyan completed this work at University of Wisconsin - Madison.
Q3 Authors' addresses:

https://doi.org/10.1145/3126527

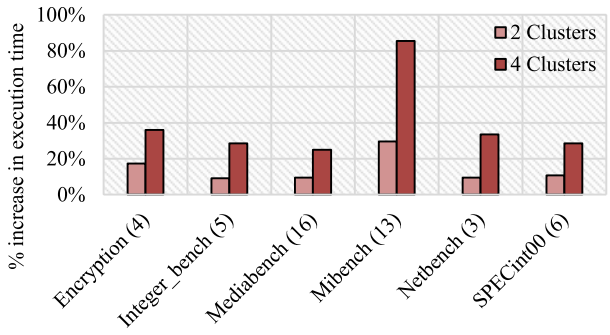


Fig. 1. Increase in execution time of two and four cluster processors when compared to a uni-cluster processor; all processors are running at same frequency.

processing (wired/wireless) that require real-time performance level [10, 20]. While found in many other devices, a mobile phone is probably the most common device with at least one VLIW DSP.[1]

VLIW processors rely on compilers to extract and express instruction level parallelism (ILP) of a program. With most of the processor components simplified, scalability of these processors is limited primarily by the large multiported register file (RF) [34]. The RF needs sufficient ports to read and write operands of all the instructions that can be scheduled in a given cycle. Typically, an n-wide VLIW processor requires *2n* read ports and *n* write ports. In this paper, an m-read, n-write RF is represented as mRnW. Additionally, compile time scheduling requires many architected registers to lower the spill-recovery code. Thus a large multiported RF, that is likely to limit the scalability, is an essential part of a high performance VLIW processor [39].

In this literature, the access delay, individual access energy, and area are defined as *characteristics of importance* (COI) for a RF. For a fixed number of entries in an RF, adding access ports dramatically increases the COI due to (1) increased the load capacitance on the storage bit cells, requiring larger bit cells and sense amplifiers to achieve comparable delays. (2) increased bit and word lines complicate routing and quadratically increases the area of the RF [42].

Modern VLIW processors limit the COI of the register file, and improve scalability by decentralizing execution units into clusters [5, 11, 19]. TI TMS320C6x [23], Equator MAP1000 [15], ADI TigerSharc [18] or HP/ST Lx [16] are some commercial implementations of such decentralized/clustered processors. These cluster processors divide the functional units and RFs amongst the clusters in a roughly symmetric fashion. These processors, with the divided RF, can run at higher frequencies and can improve performance over the uni-cluster counterparts.

However, the functional units in these processors have limited access to the architected registers. Inter-cluster communication (ICC) instructions are required to communicate architected registers from one cluster's RF to another cluster's RF. These ICC operations can also lead to additional spill-recovery code due to increased register pressure. Scheduling of useful instructions can also be delayed due to these additional operations. Increasing ICC channel count can alleviate some of scheduling limitation but our experiments show that instruction overhead had trivial reduction. Note that increasing ICC channels quickly diminishes the benefits of decentralized RF, as it would be synonymous to implementing multiple access ports outside the RF.

Figure 1 shows the increase in execution cycles for different clustered processors compared to an equivalent uni-cluster baseline. In this figure, the cycle time benefits of clustering are ignored

[1]Billions of baseband VLIW DSP radios in smart phones worldwide consume up to 125MW of power at the wall plug, or about 400K tons coal/year.

to observe the overheads. Details of the evaluation for this figure are presented in Section 4. A couple of interesting observations can be made from the Figure 1. First, wider VLIW processors are definitely useful. If the programs were not scalable beyond 4-wide cluster, then the execution times on the uni-cluster and the two-cluster processors would have been similar. Second, the overheads of clustering are significant and increase with degree of clustering.

This paper presents CURE — a technique to improve performance of decoupled VLIW processors. CURE is a hardware-software technique with enhancements in hardware, instruction set architecture (ISA) and compiler to increase availability of architected register values to all clusters. Increased availability reduces ICC instructions and spill–recovery instructions resulting in a more efficient code schedule. In this literature, we describe two architectural variants of CURE - CURE-C and CURE-X. The hardware enhancements focus on register file organization and is different for CURE-C and CURE-X. Assuming a two cluster machine as baseline, CURE-C uses two communication register banks each of which is written exclusively by one cluster and read exclusively by the other cluster. CURE-X, on the other hand, uses a single communication register bank and uses network coding concept to share access ports between the clusters. Enhanced ISA allows specifying the use of communication registers within the instruction bundle, which reduces the number of explicit communication instructions in the program. The compiler is augmented with new code scheduling heuristic that enables using the hardware and ISA features to generate improved program binaries.

The primary contributions of this paper are as follows:

(1) Hardware microarchitectures of CURE-C and CURE-X are presented to enable efficient inter–cluster communication using explicit communication register banks.
(2) ISA is changed to integrate communication operations within the instruction bundles, which enable efficient code generation by reducing explicit communication instructions.
(3) New code scheduling heuristic in the compiler generates high performance code that can use the ISA and microarchitectural enhancements.
(4) A quantitative analysis of CURE and its comparison with uni–cluster and two–cluster baselines is presented in this paper. The code for two-cluster baseline is generated by CARS [26], a state-of-the-art code generation algorithm.[2]

Unlike the prior techniques where values between instructions are communicated via duplicated register files, architectural register banks, scratch pad buffer or stack cache, communication between different clusters is achieved without any explicit communication instructions. In our analysis of clustering artifacts with the two-cluster baseline, we found that explicit ICC operations hurt performance the most. On an average, when compared to a two-cluster baseline, CURE reduces these explicit ICC operations by 15.3x and boosts performance by 25% while reducing the RF dynamic energy by 26%. Compared to a uni-cluster baseline, CURE runs 61% faster while consuming 77% lower RF dynamic energy. The configurations of baseline and CURE designs are presented in Table 2.

The rest of the paper is organized as follows. In Section 2, the hardware enhancements of CURE-C and CURE-X are detailed. The software enhancements are explained in Section 3. Section 4 details the evaluation and analysis of CURE. Some of the related work is presented in Section 5. Section 6 concludes the paper.

[2]The evaluation with BUG [14] code generation algorithm resulted in lower performance consistently and is excluded for conciseness.

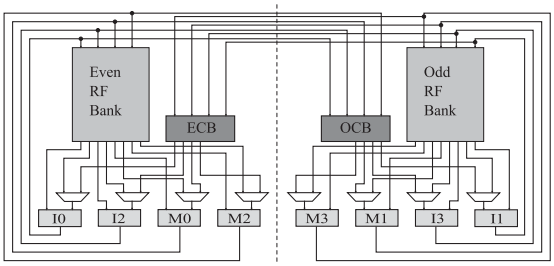


Fig. 2. Register file architecture of CURE-C.

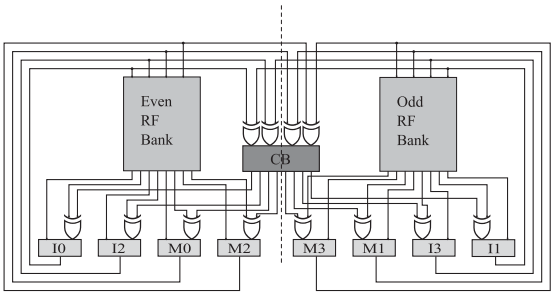


Fig. 3. Register file architecture of CURE-X.

2 HARDWARE CHANGES

Since CURE is targeted for VLIW processors, the primary objective is to achieve speedy hardware that can operate using low energy. While dependent on the target processor, this technique should be attractive in most VLIW designs. The uni-cluster baseline in this paper has eight functional units and a 16R8W 128-entry RF. A comparable two-cluster baseline has two 8R4W 64-entry register banks with an ICC network. CURE architecture is compared and contrasted against these baseline processors throughout this paper.

2.1 CURE-C

Figure 2 shows a representative architecture of the CURE-C proposed in this paper. Like the two-cluster baseline, CURE-C has two clusters, each with a private register bank. These private banks are referred to as primary banks and are marked as "even" and "odd" in the Figure 2. Each of those has a 64-entry 8R4W RF.

A communication register bank (CRB), named as ECB for the even cluster and OCB for the odd cluster in the Figure 2, is added to each cluster and facilitate high bandwidth buffered communication. Each CRB is 4R4W and has statically variable size. Only values generated by the other cluster can be written into a CRB. In the example presented in Figure 2, ECB gets values produced by the odd cluster and OCB gets values generated by the even cluster.

2.2 CURE-X

Figure 3 shows a representative architecture of CURE-X used in this paper. A coded bank (CB) is added in the middle and is accessible by the functional units of both clusters. This 8R4W, statically variable size, CB acts as an ICC channel and gets its values by storing the EXOR'ed values of primary bank write backs. This amplifies the write port bandwidth and effective storage space of

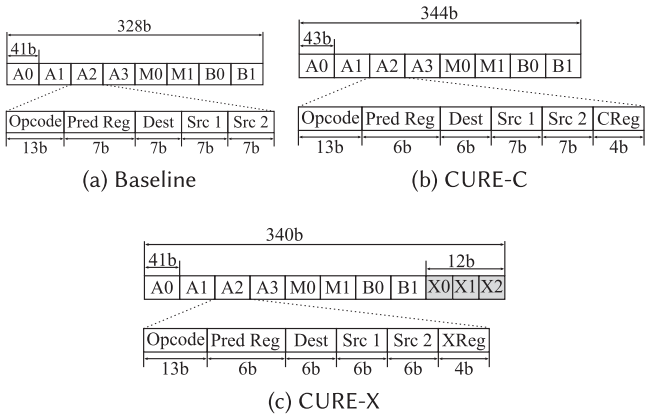


Fig. 4. Instruction bundle and operation formats. Ax = ALUs, Mx = memory units, Bx = branch units.

the coded bank. These simultaneous write backs into the primary registers are now "paired" in the coded bank. If an instruction requires an operand that is not present in it's cluster, the coded bank then provides an encoded value of the operand. The required operand can be decoded by EXORing this encoded value with the accessible paired register value.

2.3 Additional Hardware Details

Shared architectural features of CURE-C and CURE-X are explained in this section. Figure 2 and Figure 3 show that only one input of a functional unit can be provided by the CRB or the CB. This places additional constraints on the code schedule generation during compilation. This limitation is optional and does not affect the functionality of either techniques. While additional EXOR gates and multiplexers can enable operand delivery to both ports, a more constrained design is considered for analysis in this paper. This makes design explanation more comprehensible, and also imposes more constraints on the compiler.

In lieu of the ICC network, CURE-C and CURE-X add (1) Additional register banks — CRB or CB add area, leakage power and dynamic access energies. However, the access delay of the RF does not degrade as long as the size of CRB/CB does not exceed the size of primary bank. The size of the CRB/CB is a key factor in deciding the overhead COI for these RF architectures. (2) The rank of 2:1 Multiplexers in CURE-C or the rank of EXOR gates in CURE-X add to the delay, leakage power, dynamic energies and area of the RF.

The compiler is aware of CURE RF and orchestrates the functionality, resulting in a simple and attractive hardware. The software implementation also enables more sophisticated management of the communication registers. If required, the compiler specifies communication registers to be read to get the operands. It can also specify the output communication registers in the instructions. The instruction set is augmented with these additional fields to enable CURE.

2.4 ISA Changes

CURE ISA modifications require additional ISA bits to specify the read and write indices of the CRB/CB in the instruction bundle. To analyze the overhead of introducing additional ISA bits, we consider the baseline instruction format similar to Intel IA-64 [25] and Figure 4(a) shows this format. Each instruction bundle has eight instructions with four ALU, two memory and two branch operations. With the assumption of 128 general purpose registers and 128 predication registers, each register specifier takes seven bits.

Figure 4(b) shows the updated instruction bundle and operation formats envisioned to support CURE-C with 16 entry CRBs. Assuming each CRB has 16 entries, an operation requires four bits to specify the output CRB register. Clustering in CURE-C reduces the register index bits down to six. Since source registers need an additional bit to specify the CRB register when necessary, only two bits are available after clustering. Therefore, to support CURE-C, two additional bits are added per operation resulting in 16 additional bits per instruction bundle.

Figure 4(c) shows the updated instruction bundle and operation formats envisioned to support CURE-X with 16 coded registers. Assuming a CB of size 16, each operation requires four bits to specify the source coded register. The four unused bits per operation, gained from clustering, can now be re-purposed for specifying the source coded register. Although specifying source coded register does not increase the size of an instruction bundle, we still require 12 additional bits per instruction bundle to specify the destination coded registers. The 12 additional bits stems from the requirement of accommodating three write-backs into the coded bank from six operations (four ALU and two memory operations) in a bundle.

When compared to the baseline uni-cluster machine, CURE-C and CURE-X require 4.9% and 3.7% more bits to specify an instruction. Since the total number of instructions are different for CURE, the uni-cluster baseline and the two-cluster baseline, this relative memory footprint of CURE has to be evaluated quantitatively. This evaluation is presented in Section 4.

In addition to the ISA modification, the ISA extension includes a special XMOVE operation. An XMOVE operation reads the physical register from the primary bank and writes into the appropriate CRB in CURE-C. In CURE-X, XMOVE operation supplies the read physical register as one of the XOR write inputs to the coded bank. XMOVE is an ICC operation in CURE designs. In the two-cluster baseline, the ICC operations copy the register values from one private register bank to another. Although this enables re-usability of the copied value across multiple instructions, it also increases register pressure and can lead to spill-recovery code. Since XMOVE only writes CRB/CB, increased ICC operations in CURE can not cause spill-recovery code.

Since CURE ISA specifies communication registers to be written by an instruction bundle, ICC operations are embedded and limit the explicit XMOVE operations. This reduction in XMOVEs and the unaffected spill-recovery code due to XMOVEs gives CURE a distinct advantage over the two-cluster baseline.

2.5 CURE-C Vs CURE-X

The differences in CURE-C and CURE-X architectures results in different characteristics and trade offs. Since there are two CRBs, each containing half the read ports compared to the CB, the combined COI of two CRBs in CURE-C have similar COI to the CB in CURE-X. However, the architecture results in different energy characteristics.

The CURE-X design provides several advantages over the CURE-C design due to (1) merging of writes imply fewer total writes, which lowers energy consumption. (2) Simpler physical layout as the write path wires meet at the EXOR gates and not cross each other. (3) The CURE-X ISA changes requires less overhead per instruction word, saving the code size.

In CURE-X, an operand value located in remote cluster is typically obtained by reading the primary bank and the CB simultaneously and EXORing these two values. On the other hand, CURE-C architecture can supply value from either the primary bank or the CRB which can limit the additional read energies. However, there are energy components due to pre-charging bit lines and leakage currents that still contribute to the overall energy. Additionally, since the write energies of a register file are usually higher than the read energies, CURE-C is likely to consume more energy than CURE-X designs. Section 4 presents a more detailed comparison on these two architectures. Overall, from the hardware perspective, CURE-X is a more appealing design compared to CURE-C.

3 SOFTWARE MODIFICATIONS

The CURE architecture adds an additional resources—the CRBs in CURE-C or the CB in CURE-X—that the compiler must manage in addition to its traditional tasks of allocating registers and scheduling code to the execution units. The interaction between these two code generation tasks depends on phase ordering, and is complicated by the need to introduce spill code when the allocator runs out of registers. The liveness range of a variable—needed to construct the conflict graph and assign variables to non-conflicting register—is determined by the time interval between its definition and its last use, both of which are dependent on code scheduling decisions, which in turn are circularly dependent on spill code introduced by an insufficient number of allocable registers.

When generating code for CURE-X, the temptation is to treat the CB as just another RF, and apply known-good heuristics (such as graph coloring [6]) to allocate coded registers. However, the task of allocating coded registers is even more intimately coupled to scheduling than is the case with conventional registers, as the pairwise binding of register values to CB entries is determined by the co-occurrence of paired register writes in the two clusters. Hence, the coded register allocator must know precisely which pair of instructions are scheduled to finish in the same cycle, must then bind those two register names to a single entry in the CB, and must associate the beginning of the live range of that coded register with that cycle, and the end of the live range with the latter of the last uses of either of the paired registers. This information, needed to populate the conflict graph used for conventional register allocation approaches, is not available until the final schedule has been created, which, when faced with a shortage of coded registers, must be modified to accommodate additional copy/move operations.

The closely-intertwined relationship between allocating coded registers and scheduling of instructions in a given VLIW cycle motivates unification of cluster assignment, instruction scheduling and register allocation into a single phase. This is similar to the state-of-the-art code generation proposal, CARS [26]. The CARS algorithm considers all resource constraints and also allocates output registers on-the-fly at each cluster scheduling step. This reduces register spills and avoids iterative re-scheduling steps. This state-of-the-art algorithm is implemented in our code generation framework and is used to generate code for the baseline processors.

In this work, CURE is applied to only the integer RF. So, "register", unless explicitly characterized, implies an integer register. The basic scheduling units for CURE are a set of basic blocks or hyperblocks. These scheduling units are selected for cluster scheduling strictly in topological order. Operations within a block are scheduled in top down fashion. At the beginning of each block, the CRBs in CURE-C and the CB in CURE-X are assumed to be empty to statically ensure that a communication register will be available when required during dynamic execution.

3.1 CURE Code-Generation Algorithm

Algorithm 1 presents our proposed code generation algorithm at high-level. First, a list of data-ready operations is formed for each cycle and an estimation of each operation's execution on program's critical path is done. An operation's criticality is estimated by adding it's height and depth. Here, depth is the earliest execution cycle of an operation, counting from the start of a data-flow graph, whereas height is the latest execution cycle of the operation, counting from the end of the data-flow graph. Both depth and height of an operation are derived under the premise of an infinite resource machine.

In order to determine the best cluster to schedule an operation, the following set of factors are used to compute the resource-constrained earliest schedule cycle for each cluster.

(1) When assigning a cluster, priority is given to the cluster with access to all the input operands from its primary bank. If an operation requires access to a CRB (ECB or OCB) in

ALGORITHM 1: CURE Algorithm (High level)

```
for all block ∈ Unscheduled_Code do
  CURE-C: CRB (ECB and OCB) ← φ
  CURE-X: CB ← φ
  while unscheduled operations exist in block do
    Ops ← Prioritized list of source-ready unscheduled operations
    for Op ∈ Ops do
      for cluster = 0 to max_clust_num do
        sched_cycle[cluster] = Resource-Constrained-Schedule(Op, cluster)
      end for
      if Op schedulable in current_cycle then
        Schedule Op on the cluster requiring minimum XMOVE ops
        Update all machine resources
        Update depth of dependent operations
        Allocate output registers
        CURE-C: "COPY" write-backs to CRB (ECB and OCB)
        CURE-X: "PAIR" simultaneous write-backs to CB
        Insert and schedule required XMOVE operations
        Ops ← Ops − Op
      end if
    end for
    CURE-C: W_CRB ← {Writes in this cycle}
    R_CRB ← {Overwritten registers}
    CRB_occ = CRB_occ + W_CRB − R_CRB
    CURE-X: WPS ← {Write pairs in this cycle}
    RS ← {Overwritten pairs}
    CB_occ = CB_occ + WPS − RS
    if current_cycle < min(depth of Ops) then
      Move to next cycle
    end if
  end while
end for
```

CURE-C or to the CB in CURE-X, other factors affect cluster assignment and Algorithm 2 details this process further.

(2) If the operation cannot access a source operand located in a remote cluster through the CRBs or the CB, an XMOVE operation has to be inserted on the cluster with access to the source register. So, source readiness for the dependent operation may be delayed depending on the earliest cycle in which this XMOVE operation can be scheduled.

(3) If machine resources like ALU, destination register and load/store queue entries are not available, the operations have to wait to be scheduled in the next cycle.

(4) When free registers are not available in the CRBs or in the CB, a simple replacement policy is used to reuse a communication register. This replacement policy for the communication registers is described later in this section.

CURE-C and CURE-X designs primarily differ in the way the communication registers are updated in response to the write-backs into the primary banks, as illustrated in Algorithm 1.

If an operation can be scheduled on either cluster in the same cycle, a cluster that does not require an XMOVE operation is selected. Alternatively, the instruction can be scheduled to a cluster

ALGORITHM 2: Source operand guided cluster assignment

```
for Op ∈ Ops do
  if Number_of_sources(Op) == 2 then
    RB1 ← REG_BANK(SRC1(Op))
    RB2 ← REG_BANK(SRC2(Op))
    CBP1 ← Present_in_Coded_Bank(SRC1(Op))
    CBP2 ← Present_in_Coded_Bank(SRC2(Op))
    if RB1 and RB2 are same then
      Assign Op to cluster with RB1
    else if (CBP1 == True)&&(CBP2 == True) then
      No priority to any cluster
    else if CBP1 == True then
      Assign Op to cluster with RB2
    else if CBP2 == True then
      Assign Op to cluster with RB1
    else
      No priority to any cluster
      Record an XMOVE operation
    end if
  else if Number_of_sources(Op) == 1 then
    RB ← REG_BANK(SRC(Op))
    CBP ← Present_in_Coded_Bank(SRC(Op))
    if CBP == True then
      Assign Op to the least assigned cluster
    else
      Assign Op to cluster RB
    end if
  else
    No priority to any cluster
  end if
end for
```

with lower pressure on primary register bank and machine resources. A delay in scheduling a non-critical operation due to resource-constraints may cause the dependent instructions to be on the critical path and may increase their scheduling priority.

Depending on the availability of operands in the CRBs or in the CB and cluster assignment of the operation, XMOVE operations are often required to be inserted in the code schedule. These XMOVE operations are inserted at appropriate slots in the schedule to ensure the availability of operands to the required operation in time. For CURE-X, an ideal schedule of these XMOVE operations considers a cycle that has empty instruction slots and lower coded register pressure while maximizing the information density of the CB. Furthermore, an XMOVE operation does not explicitly transfer operands from one cluster to the other. Instead it reads the physical register from the primary bank and writes it back to the same physical register. When this register is written back, it is paired into the CB. The algorithm traverses the schedule backwards from the current cycle searching for a schedule cycle with a free instruction slot and a replaceable coded register. If a free instruction slot was not found in this time frame, the dependent instruction is delayed. Finally, CRB_{occ} and CB_{occ} counters are updated each cycle to account for the changes in occupied registers in the CRBs and in the CB.

3.2 CURE Cluster Assignment

Algorithm 2 describes in detail how the dependency of the source operands of operations guide their dispatch to different clusters in the same schedule cycle in CURE-X design. *RB*1 and *RB*2 find the primary banks of the two source operands of an operation, whereas *CBP*1 and *CBP*2 check their availability in the CB. The following scenarios are addressed.

(1) If all sources are present in the same primary bank, the operation has to be scheduled on the cluster with access to this bank. If an operation has only one source and that is available in the CB, then remaining factors considered in computing the resource-constrained-schedule cycle decides it's cluster assignment.
(2) If neither sources are available in the CB, record an XMOVE operation for one of the sources. This XMOVE operation will be scheduled such that the source will be available in the CB when required by the consumer.
(3) If all the sources are not available in the CB, the operation is scheduled on a cluster with access to the source operand not having a copy in the CB. If all sources are available in the CB, then no cluster gets any priority, other factors guide it's cluster assignment and scheduling.
(4) If there are no sources to be read from the RF, then other factors drive its cluster assignment as described before.

Similarly the presence of the source operands in the primary banks and in the CRBs in CURE-C and their dependency drive the assignment of operations to specific clusters. We leave the clustering details of CURE-C design due to space constraints and similarity to the clustering algorithm illustrated for CURE-X in Algorithm 2.

The CB is updated each cycle with the new pairs of physical registers overwriting old pairs. Two important actions during updates are as follows.

(1) *Selecting a coded register to write:* If free coded registers are available, they are preferred over replacing an active coded register. Once the number of free registers falls below a threshold, a priority scheme based on usage of the paired physical registers is used to select a replacement register. If either of the paired registers is not referenced even once, the coded register is given the lowest replacement priority. Once both the paired registers are referenced, the coded register gains replacement priority quickly unless it is used in successive cycles. The coded register with highest replacement priority is used to write the new physical register pair.
(2) *Referential integrity:* If a physical register, say Rx, is written multiple times in a relatively small window, it can be paired up with multiple physical registers. Each pair has a different version of Rx. Such a scenario has to be avoided to ensure Referential integrity. To ensure this, any coded register having an instance of either of the physical registers being written, are freed.

4 EVALUATION

Trimaran [7] infrastructure is used to implement and evaluate CURE. *ELCOR* module in *Trimaran* generates code schedule for a target processor. This module is extended with CARS and CURE algorithms for code generation. *SIMU* module in *Trimaran* simulates the target processor and generates various event counts that are used to estimate performance and energy consumption. The module is enhanced to simulate CURE architecture.

All the benchmark suites distributed with the Trimaran package were used for performance and energy evaluation. The benchmarks and their execution times in our baseline uni–cluster

Table 1. Benchmarks and the Uni–Cluster Baseline Performance

	Baseline stats	
Benchmark Suite	# of benchmarks	Average Cycles
Encryption	4	131,981,243
Integer_bench	5	10,988,051
Mediabench	16	39,553,393
Mibench	14	22,196,670
Netbench	3	291,342,168
SPECint2000	6	5,584,469,067

Table 2. Configurations of the Tested Processors

Parameter	uni–cluster	two–cluster	CURE
Clusters	1	2	2
Integer ALUs	4	2x2	2x2
Memory ALUs	4	2x2	2x2
RF (Integer)	128	2x64	2x64
CRB/CB Size	0	0	(8 to 64)
ICC BW	None	4	None
L1 I,D$	32KB, 4-way SA, 2 cycles		
L2 $	128KB, 8-way SA, 12 cycles		
off-chip mem	150 cycles		

configuration are reported in Table 1. These benchmark suites comprehensively represent various applications from mobile, security, network, multimedia and other real world applications. All benchmarks are verified to be functionally correct for both baselines and for all CURE configurations. A relatively fair representation of relative performance of a benchmark suite is the geometric mean of the relative performance of individual benchmarks in the suite. This metric is used to report the quantitative performance results for the benchmark suites. *gzip, vpr, mcf, parser, bzip2* and *twolf* with train input set, were the only functionally available SPEC2000 benchmarks for *Trimaran*.

4.1 Register File Characteristics

Fabmem tool was used to estimate the access delay, energy and area of the RFs evaluated in this paper [8]. This tool uses 45nm Nangate Technologies library to generate the configured RF netlist. *Fabmem* runs gate level simulations using HSPICE to estimate the timing delay, read and write energies, and the area of the configured RF. These estimates are used for evaluating RFs in CURE and the baseline processors. The additional delay, area and energy required for EXOR gates is accounted for CURE implementations.

A uni–cluster configuration sets the lower bound on code overhead, but is limited in the operating frequency due to RF restrictions. A clustered configuration, on the other hand, can operate at higher frequency but with significant code overhead. For this reason, we use a uni–cluster processor and a two–cluster processor to evaluate CURE's performance. Table 2 presents the different configurations used for our analysis. Performance and energy consumption were recorded with CRB/CB sizes of 64, 32, 16 and 8. These configurations are presented as CURE-C64, CURE-C32, CURE-C16 and CURE-C8 respectively for CURE-C configurations, and as CURE-X64, CURE-X32,

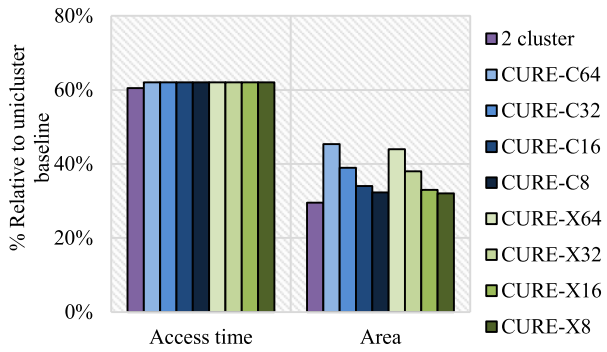


Fig. 5. Delay and area of different configurations relative to 128 entry 16R8W RF.

CURE-X16 and CURE-X8 for CURE-X configurations. The performance between the equivalent configurations of CURE-C and CURE-X are very similar across all the tested benchmarks.

Figure 5 presents a relative comparison of the access delay and area of different RF configurations normalized against a 128 entry 16R8W RF. In this analysis, the overhead of the ICC network in the two–cluster baseline is completely ignored. The added multiplexers or EXOR gates in CURE-C and CURE-X configurations, results in a small increase in access time over the two–cluster baseline. These observed delays are used in generating graphs in Figure 8(a) and Figure 8(b). As discussed in Section 2, 64 entry configurations of CURE have significant area overhead over the two–cluster baseline, but this reduces significantly when only 16 entries are considered. In this paper, 16 entry CURE configurations are championed as they have optimal COI in addition to optimal performance.

4.2 Power and Performance Analysis

Table 2 presents the different configurations used for our analysis. Local, intra-cluster bypass networks are assumed in CURE, similar to the conventional two cluster architecture [34]. Thus, enabling equivalent cycle time benefits for CURE [34]. All ICC is achieved through the communication registers. Any operations dependent on the values produced in the previous cycle on a different cluster are delayed by a cycle to get their values from the communication registers. The two cluster processor has an ICC bandwidth (ICC BW) set to four (four register values can be copied across in each cycle). Code generation for two–cluster processor is done by our implementation of CARS algorithm.

The dynamic energy consumption of RF is calculated using the activity counts from *Trimaran* and the RF parameters from *Fabmem*. The power analysis is done using the following equations.

$$E_{Read\_CURE-C} = RE_{Primary\_Bank} \times total\_reg\_reads + (E_{MUX/XOR} + RE_{CRB/CB}) \times total\_CRB/CB\_reads$$

$$E_{Write\_CURE-C} = WE_{Primary\_Bank} \times total\_reg\_writes + (WE_{CRB}) \times total\_CRB\_writes$$

$$E_{Write\_CURE-X} = WE_{Primary\_Bank} \times total\_reg\_writes + (WE_{XOR} + WE_{CB}) \times total\_CB\_writes$$

In these equations, *RE* and *WE* stand for the independent read and write energies of the RFs. The activity counters *total_reg_reads*, *total_reg_writes*,*total_CRB_reads*, *total_CRB_writes*, *total_CB_reads* and *total_CB_writes* already include the additional move operations inserted in the scheduled code. Figure 6 shows the RF power consumption relative to the baseline. The RF energy savings for the two cluster processor are between 60% to 75% depending on the benchmark suite.

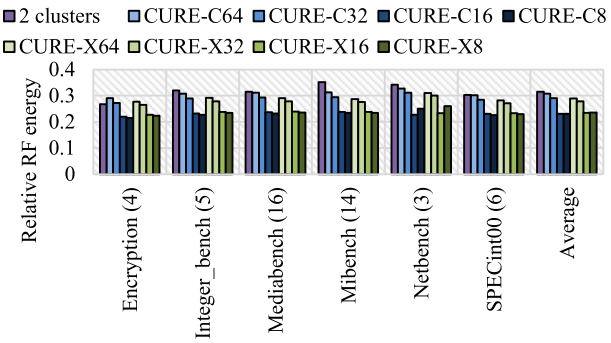


Fig. 6. Dynamic RF energy consumption relative to uni–cluster baseline.

Compared to that, both CURE-C16 and CURE-X16 design points consistently save more than 75% of the RF energy in all the benchmarks. Reducing the communication register count from 32 to 16 causes a significant dip in read and write energies. This results in a very power-efficient spot for CURE16 with almost similar number of XMOVEs as CURE with 32 communication registers. The RF power due to additional XMOVE instructions is included in this evaluation.

Two anomalies of the Figure 6 are the 64 entry CURE configurations in *Encryption* suite and 8 entry CURE configurations in *netbench*. *Encryption* suite has considerable number of CRB/CB reads. This coupled with higher read energy of the 64 entry CRB/CB causes higher energy consumption of the CURE-C64 and the CURE-X64 designs when compared to the two-cluster baseline. In *netbench*, unlike in other benchmarks, number of XMOVE operations increase significantly for CURE with eight entry CRB/CB, thus leading to higher energy than CURE with 16 entry CRB/CB.

The performance of CURE-C and CURE-X at comparable configurations was very similar. This is due to the fact that most XMOVE operations in CURE are due to the conservative static assumption on the dynamic availability of correct register values in the communication registers. To improve readability, the rest of this paper uses CURE to refer to CURE-C and CURE-X configurations.

Comparison of execution times and energy delay products of benchmarks for different configurations is done for two cases. (1) *Assume RF is not the critical path:* The cycle time of the processor does not improve if the RF is not on the critical path. The RF energy benefits still exist in the two cluster and CURE designs in both cases. (2) *Assume RF is the critical path:* If the RF is on the critical path, the access delay benefits of the RF directly benefit the cycle time of the processor.

In a practical scenario, RF is the critical path, but decentralizing makes other components, like the bypass networks, as the critical path. The intention of presenting these two scenarios is to show that CURE out-performs both baselines in both these scenarios.

In case the RF is not on the critical path, Figure 7(a) shows the relative execution time for the tested configurations. All CURE configurations have less than 3.5% increase in execution time relative to the uni–cluster baseline. In comparison, the two cluster processor suffers a 16% or higher increase in average execution time, while using the state-of-the art scheduling mechanism.

CURE suffers from performance loss in high ILP applications with long register lifetimes. Registers with long lifetimes get evicted from the communication bank due to size limitations. When they are accessed again in the program, additional XMOVE operations have to be inserted. The high amount of available ILP reduces the number of free slots available to insert the additional XMOVE operations, thus increasing the total schedule cycles of a block.

Except for CURE8 in *netbench*, the performance differences are trivial for different CURE configurations. However, the communication bank activity counts vary a lot and can be derived from

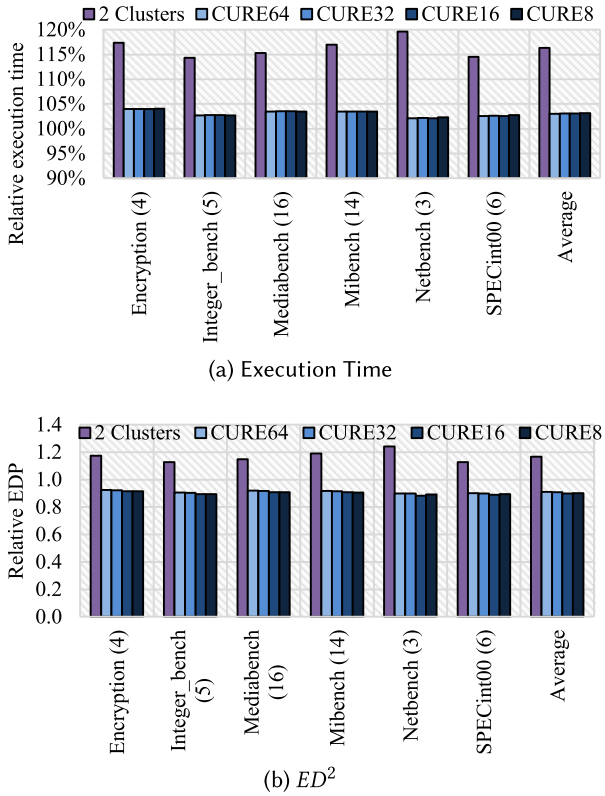
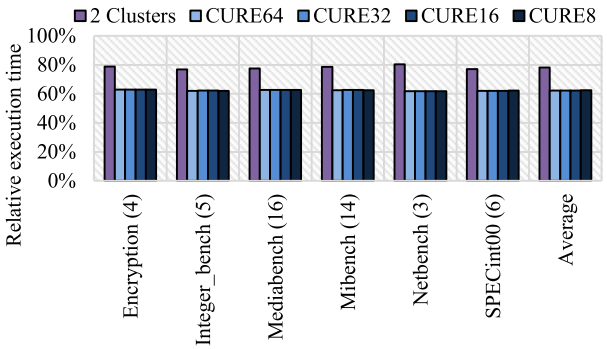


Fig. 7. Execution time and ED^2 relative to uni–cluster baseline; assuming RF is not the critical path.

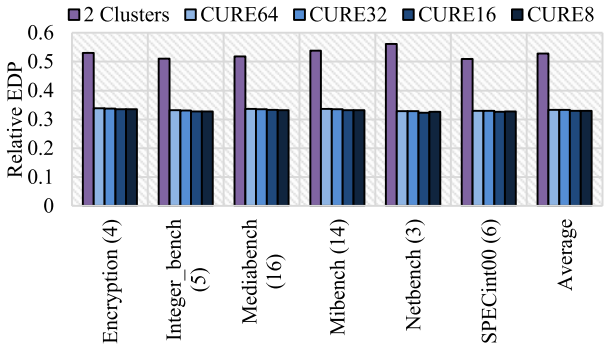
the Figure 6. The performance of CURE starts to degrade at CURE8 as the number of communication bank replacements trigger new XMOVE instructions and delay the code schedule. In this evaluation, CURE16 was the optimal configuration with best performance at lowest energy for all the benchmark suites.

Though the energy of the RF decreases significantly for two cluster and CURE configurations, it is only part of the processor energy. The energy benefits can be quickly offset by decreased performance. Figure 7(b) shows the energy delay product (ED^2) of different configurations assuming the energy consumed by CURE-X designs. Given the high activity of RF, we assume about 20% of the processor energy can be attributed to RF energy. If the RF is not on the critical path, the two cluster processor is a bad choice compared to the uni–cluster. CURE on the other hand is still a viable option with its relative energy delay product around 0.9 for the CURE16 design.

If we assume that RF access is the sole critical delay path in all the considered designs, a faster banked design like CURE enables higher frequency and improved performance. The relative execution times for different benchmark suites are presented in the Figure 8(a). CURE and the two-cluster design derive similar propagation delay benefits from their RF and bypass networks over the uni–cluster baseline. Considering the cycle time benefits, the ED^2 of different configurations is shown in Figure 8(b). The cycle time benefits are required for the two cluster processor to be favorable compared to a uni–cluster. However, CURE16 still has the best overall power and performance benefits in all the compared configurations.



(a) Execution Time



(b) ED^2

Fig. 8. Execution time and ED^2 relative to uni–cluster baseline; assuming that RF is the critical path.

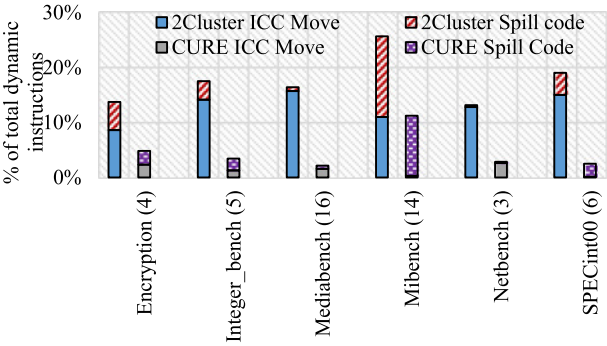


Fig. 9. Geometric mean of spill–recovery and ICC Move (ICM) instructions as percent of total dynamic instructions.

Clustering artifact reduction: Figure 9 shows the XMOVE and the spill–recovery code as a percent of the total dynamic instructions for the two–cluster and CURE16 configurations. As noted in Section 2, the implicit ICC operations in CURE reduce the number of explicit ICC operations from an average of 13% in two–cluster baseline to 1% in CURE. Note that two–cluster configuration has about 18% more overall instructions than CURE, which translates to a humongous 15.3x reduction in number of ICC operations from two–cluster baseline to CURE. In CURE,

the number of the explicit ICC operations, XMOVEs, indicate the communication bank pressure. Spill–recovery code is reduced from an average of 2.5% in the two–cluster configuration down to 1.6% in CURE. This translates to 84% lower spill–recovery instructions in CURE compared to the two–cluster baseline. High availability of registers, low ICC operations, lower spill–recovery code all result in a better code schedule that executes faster. These benefits were clearly highlighted in the *Rijndael* benchmark from *Mibench* where the two–cluster processor takes 150% more execution cycles than the uni-cluster baseline, while CURE16 takes only 7.3% more.

Instruction memory footprint: CURE-C and CURE-X have 4.9% and 3.7% respective increases in the instruction size than the baseline. However, CURE has 18% lesser average instructions than the two–cluster baseline, resulting in 14% and 15% reduction in average instruction memory footprint for CURE-C and CURE-X respectively. The effective impact of instruction size increase should be in the instruction-holding-caches, where less than 5% typically has trivial impact on the processor's performance and power.

Leakage power analysis: Leakage power was not reported by *Fabmem*. Assuming that leakage power is proportional to area, CURE16 RF has a small increase in leakage power over the two–cluster baseline. The area and leakage power increase of CURE16 RF is very small relative to the area and leakage power of the entire chip. Higher performance of CURE is likely to reduce the overall leakage energy of the chip.

Increasing two–cluster RF size: If the area overhead of CURE16 RF is allocated to the two–cluster baseline's RF, its configuration changes from 2×64 to 2×72. Ignoring the timing effects of bigger RF, the 2×72 RF provides a trivial, 0.02%, gain in performance over the 2×64 RF. Increasing the RF size has little impact on ICC operations count, resulting in this trivial performance gain with bigger RF.

ICC bandwidth scaling: In an experiment with two–cluster baseline, an increase in ICC bandwidth showed trivial performance benefits. This is attributed to the fact that increased ICC bandwidth although allows more registers to be communicated across clusters in a cycle, however it has little impact on the ICC instructions and spill–recovery code. Generally, ICC complexity has to be engineered so that the benefits of clustering (improved cycle time, power and area) are not lost.

5 PREVIOUS WORK

There are decades of research on addressing the problems of large multiported RFs in both superscalar and VLIW processors. While the solutions for superscalar processors are predominantly hardware-only, solutions for VLIW processors additionally rely on compiler in order to simplify hardware.

Clustered architectures: Clustering of the processor resources and dividing the RF into smaller, lower ported banks lowers the RF access time and power. In superscalar processors, the problem of bank conflicts was addressed by various techniques implemented in the hardware [13, 17, 22, 36, 37]. VLIW processors [23, 24] that use clustering are also referred to as limited connectivity VLIW architectures, implying availability of only a subset of architected registers to each cluster [5]. An extensive study on performance and scalability of various ICC networks has shown that performance of clustered processors is significantly lower than the uni-cluster processor when cycle time benefits are ignored [19, 28, 35]. However, the improved RF access delay and the bypass path delay can increase the processor frequency and lower the execution time [34]. A recent research by Zhao et al. [41] also explores using non-uniform size register partitions to save on RF energy.

RF caching or hierarchical RFs use lower ported full RF in conjunction with a smaller fully ported RF cache to improve cycle time and to limit RF power [4, 12, 38, 39]. A major challenge

with RF caching is the increased latency of memory operations that always interact with second level RF, and typically need another cycle to get the value into the first RF.

Bypass networks [4, 21] use the spatial locality of values to lower RF access ports in uni–cluster processors [31] or to solve bank conflicts in clustered processors [36].

Code generation for clustered processors has extensive research, and we only present a subset of the algorithms we tried in our evaluations. Bottom-up Greedy (BUG) algorithm [14] recursively traverses from the exit nodes to entry nodes of the input data precedence graph and assigns estimate of the functional unit and operand availability for each operation. A list scheduler inserts the communication operations where necessary and generates the final code schedule. Multi-flow TRACE compiler [27] extends BUG to assign independent dependence chains to different clusters. Restricted interconnect could result in sub-optimal usage of the clusters or over insertion of communication operations. Percolation Scheduling Compiler [32] for clustered VLIW processors generates code assuming a fully connected VLIW and then partitions the code across clusters while inserting necessary copy operations. Addressing the phase ordering effects, UAS [30] integrates phases of cluster assignment with code scheduling, while CARS [26] integrates register phase allocation also. An alternate scheme in [40], like CARS, integrates all phases of code generation and results in efficient code with benefits similar to that of CARS.

There are a number of modulo scheduling approaches, targeting loop-intensive codes for clustered VLIW architectures [1–3, 9]. URACAM [9] also performs clustering, scheduling, and register allocation in an unified stage, similar to that of CARS, and allows trading ICCs for memory pressure. [2, 3] builds on URACAM and improves the cluster assignment phase further through graph-partitioning techniques, resulting in more balanced workloads among the clusters and less ICCs. However, for two-cluster configurations, these techniques show marginal improvement in performance when compared to URACAM and CARS. So, in this research, CARS [26] is used to generate code for the two-cluster baseline. However, these acyclic code generation techniques can benefit further by using these graph-partitioning [2] and instruction replication [1] based modulo scheduling heuristics.

Coded Register File use is first advocated in CRAM [29] to address RF issues in superscalar processors. CRAM, while useful in superscalar processors, is ill suited for VLIW processors due to hardware complexities in coded bank management, and in scheduler changes.

6 CONCLUSIONS

In this paper, we present the CURE, a new design to address the problems associated with large multiported register files in VLIW processors. We propose use of communication registers in a clustered architecture to increase the availability of the registers to all the clusters. Communication registers act as an inter-cluster communication path to ease up the bypass networks as well. Two variants of CURE architecture are presented and their merits were evaluated. The hardware and software changes required to make the CURE work are detailed. We show that the performance and power benefits of the CURE make it desirable over uni-cluster and conventional multi-clustered VLIW processors.

REFERENCES

[1] Alex Aletà, Josep M. Codina, Antonio González, and David Kaeli. 2003. Instruction replication for clustered microarchitectures. In *MICRO-36*.

[2] Alex Aletà, Josep M. Codina, Jesús Sánchez, and Antonio González. 2001. Graph-partitioning based instruction scheduling for clustered processors. In *MICRO-34*.

[3] Alex Aletà, Josep M. Codina, Jesús Sánchez, Antonio González, and David Kaeli. 2002. Exploiting pseudo-schedules to guide data dependence graph partitioning. In *PACT*.

[4] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi. 2001. Reducing the complexity of the register file in dynamic superscalar processors. In *MICRO-34*.
[5] A. Capitanio, N. Dutt, and A. Nicolau. 1992. Partitioned Register Files For VLIWs: A preliminary analysis of tradeoffs. In *MICRO-25*.
[6] Gregory Chaitin. Register allocation and spilling via graph coloring. *SIGPLAN Not.* 39, 4.
[7] Lakshmi N. Chakrapani, John Gyllenhaal, Wenmei W. Hwu, Scott A. Mahlke, Krishna V. Palem, and Rodric M. Rabbah. 2004. Trimaran: An infrastructure for research in instruction-level parallelism. In *In Instruction-level Parallelism. Lecture Notes in Computer Science*. Springer-Verlag, www.trimaran.org.
[8] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg. 2011. FabScalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template. In *ISCA-38*.
[9] Josep M. Codina, Jesús Sánchez, and Antonio González. 2001. A unified modulo scheduling and register allocation technique for clustered processors. In *PACT*.
[10] L. Codrescu, W. Anderson, S. Venkumanhanti, M. Zeng, E. Plondke, C. Koob, A. Ingle, R. Maule, and R. Talluri. 2013. Qualcomm Hexagon DSP: An architecture optimized for mobile multimedia and communications. In *Hot Chips*.
[11] Osvaldo Colavin and Davide Rizzo. 2003. A scalable wide-issue clustered VLIW with a reconfigurable interconnect. In *CASES*.
[12] J.-L. Cruz, A. Gonzalez, M. Valero, and N. P. Topham. 2000. Multiple-banked register file architectures. In *ISCA-27*.
[13] Nam Duong and R. Kumar. 2009. Register Multimapping: A technique for reducing register bank conflicts in processors with large register files. In *SASP-7*.
[14] John R. Ellis. 1985. *Bulldog: a compiler for vliw architectures (parallel computing, reduced-instruction-set, trace scheduling, scientific)*. Ph.D. thesis.
[15] Equator. 1998. MAP1000 unfolds at Equator. In *Microprocessor Report*.
[16] Paolo Faraboschi, Geoffrey Brown, Joseph A. Fisher, Giuseppe Desoli, and Fred Homewood. 2000. Lx: a technology platform for customizable VLIW embedded processing. In *ISCA-27*.
[17] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic. 1997. The multicluster architecture: reducing cycle time through partitioning. In *MICRO-30*.
[18] Jose Fridman and Zvi Greenfield. 2000. The TigerSHARC DSP Architecture. *IEEE Micro*.
[19] A. Gangwar, M. Balakrishnan, P. R. Panda, and A. Kumar. 2005. Evaluation of bus based interconnect mechanisms in clustered VLIW architectures. In *DATE*.
[20] J. S. Gardner. 2012. CEVA Exposes DSP Six Pack. In *Microprocessor Report*.
[21] N. Goel, A. Kumar, and P. R. Panda. 2007. Power Reduction in VLIW Processor with Compiler Driven Bypass Network. In *VLSID-20*.
[22] A. Gonzalez, J. Gonzalez, and M. Valero. 1998. Virtual-physical registers. In *HPCA-4*.
[23] Texas Instrucments Inc. 1998. TMS320C62x/67x CPU and instruction set reference guide.
[24] Texas Instruments. 2010. TMS320C6745/C6747 Fixed/Floating- point digital signal processors (Rev.D).
[25] Intel. Intel Itanium Architecture Software Develorer‘s Manual: Intel Itanium Instruction Set. *www.intel.com* 3, 293–370.
[26] Krishnan Kailas and Ashok Agrawala. 2001. CARS: A new code generation framework for clustered ILP processors. In *HPCA*.
[27] P. Geoffrey Lowney, Stefan M. Freudenberger, Thomas J. Karzes, W. D. Lichtenstein, Robert P. Nix, John S. O'Donnell, and John C. Ruttenberg. 1993. The multiflow trace scheduling compiler. *The Journal of Supercomputing* 7 (1993), 51–142.
[28] R. Nagpal and Y. N. Srikant. 2007. Register file energy optimization for snooping based clustered VLIW architectures. In *SBAC-PAD-19*.
[29] V. R. K. Naresh, D. J. Palframan, and M. H. Lipasti. 2011. CRAM: Coded registers for amplified multiporting. In *MICRO-44*.
[30] Emre Özer, Sanjeev Banerjia, and Thomas M. Conte. 1998. Unified assign and schedule: a new approach to scheduling for clustered register file microarchitectures. In *MICRO-31*.
[31] I. Park, M. D. Powell, and T. N. Vijaykumar. 2002. Reducing register ports for higher speed and lower energy. In *MICRO-35*.
[32] Roni Potasman. 1992. *Percolation based compiling for evaluation of parallelism and hardware design trade-offs*. Ph.D.
[33] C. Rowen, D. Nicolaescu, R. Ravindran, D. Heine, G. Martin, J. Kim, D. Maydan, N. Andrews, B. Huffman, V. Papaparaskeva, S. Gal-On, P. Nuth, P. Patwardhan, and M. Paradkar. 2011. The World's Fastest DSP Core: Breaking the 100 GMAC/s Barrier. In *Hot Chips*.
[34] A. Terechko, M. Garg, and H. Corporaal. 2005. Evaluation of speed and area of clustered VLIW processors. In *VLSID-18*.

[35] A. Terechko, E. Le Thenaff, M. Garg, J. van Eijndhoven, and H. Corporaal. 2003. Inter-cluster communication models for clustered VLIW processors. In *HPCA-9 2003*.

[36] J. H. Tseng and K. Asanovic. 2003. Banked multiported register files for high-frequency superscalar microprocessors. In *ISCA-30*.

[37] S. Wallace and N. Bagherzadeh. 1996. A scalable register file architecture for dynamically scheduled processors. In *PACT*.

[38] R. Yung and N. C. Wilhelm. 1995. Caching processor general registers. In *ICCD*.

[39] J. Zalamea, J. Llosa, E. Ayguade, and M. Valero. 2000. Two-level hierarchical register file organization for VLIW processors. In *MICRO-33*.

[40] Javier Zalamea, Josep Llosa, Eduard Ayguad, and Mateo Valero. 2001. Modulo scheduling with integrated register spilling for clustered VLIW architectures. In *Micro-34*.

[41] Yingchao Zhao, C. J. Xue, Minming Li, and B. Hu. 2009. Energy-aware register file re-partitioning for clustered VLIW architectures. In *ASP-DAC*.

[42] V. Zyuban and P. Kogge. 1998. The energy complexity of register files. In *ISLPED*.

Q4 Received xxx; revised xxx; accepted xxx

Author Queries

Q1: AU: Please provide CCS 2012 Concepts per author guidelines and provide XML codes as well.
Q2: AU: Please provide Additional Key Words and Phrases.
Q3: AU: Please provide complete mailing and email addresses for all authors.
Q4: AU: Please provide article history dates.