Heuristics for Greedy Transport Triggered Architecture Interconnect Exploration

Timo Viitanen Heikki Kultala Pekka Jääskeläinen Jarmo Takala

Tampere University of Technology, Department of Pervasive Computing, Finland

{timo.2.viitanen, heikki.kultala, pekka.jaaskelainen, jarmo.takala}@tut.fi

Abstract

Most power dissipation in Very Large Instruction Word (VLIW) processors occurs in their large, multi-port register files. Transport Triggered Architecture (TTA) is a VLIW variant whose exposed datapath reduces the need for RF accesses and ports. However, the comparative advantage of TTAs suffers in practice from a wide instruction word and complex interconnection network (IC). We argue that these issues are at least partly due to suboptimal design choices. The design space of possible TTA architectures is very large, and previous automated and ad-hoc design methods often produce inefficient architectures. We propose a reduced design space where efficient TTAs can be generated in a short time using excecution trace-driven greedy exploration. The proposed approach is evaluated by optimizing the equivalent of a 4-issue VLIW architecture. The algorithm finishes quickly and produces a processor with 10% reduced core energy product compared to a fully-connected TTA. Since the generated processor has low IC power and a shorter instruction word than a typical 4-issue VLIW, the results support the hypothesis that these drawbacks of TTA can be worked around with efficient IC design.

Categories and Subject Descriptors C.3 [*Special-purpose and application-specific systems*]

General Terms Transport Triggered Architecture, Interconnection Network, Register File

Keywords TTA, VLIW, design space exploration, port sharing

1. Introduction

Very Long Instruction Word (VLIW) processors are widely used for digital signal processing. A typical bottleneck for VLIW processor scaling is the large, many-port *register file* (RF). There is a large body of literature on optimizing RF power. One main approach is to partition the RF into a multi-banked [2] or clustered [16] design; another is to exploit the fact that most values are short-lived and need not be saved to the RF. These can be transported through the *forwarding network* whose original purpose is to eliminate data hazards. Bypass aware compilers [19] eliminate RF accesses by scheduling them so that they are eliminated by the forwarding logic. For greater effect, the ISA can include finer-grained control over the forwarding network, leading to *exposed-datapath* architectures such as the *Transport Triggered Architecture* (TTA) [1], which may remove as many as 80% of RF writes and 60% of reads [9]. However, various sources report that much of the saved RF power is transposed into a complex, power-hungry *interconnection network* (IC) and a wide instruction word, reducing the appeal of this architecture [3, 9]. The code density drawback is particularly severe, given that memory access power is scaling less efficiently than computation logic.

We were unable to find an intrinsic reason in the TTA architecture for the reported code density and IC complexity drawbacks, and there is no comparative study on the subject. As noted in [1], the architecture is sufficiently flexible to represent the topologies of conventional RISC and VLIW processors. In this case, the IC should be no more complex than the forwarding network of the RISC or VLIW counterpart since there is no need for runtime bypass detection. Our working hypothesis which we investigate in this article is that the IC and code density drawbacks are, rather than intrinsic features of the architecture, artifacts of inefficient IC design.

TTA design is complicated by the large design space of possible interconnection network topologies. A moderately large TTA analyzed later in the paper has 2^{132} possible IC networks, many of which are inefficient or nonfunctional. Existing automated TTA design flows start from a *fully connected* IC where every bus is connected to every port. This type of IC exhibits a harmful kind of redundancy. For example, in a 8-bus fully-connected IC, any instruction may be rearranged in $8! = 40320$ different ways which all have the same effects but different encodings. Several research TTAs opt to use a fully-connected [5, 9] or nearly fully-connected [17] ICs.

In this paper, we attempt to lay out a practical TTA IC design algorithm, and to measure the effects of IC optimization on power consumption. This is approached by limiting the design space to a subset similar to VLIW processor ICs.

This paper makes the following contributions:

- We propose to simplify the TTA design space by starting from a VLIW-like configuration and merging transport buses.
- Based on said design space, an automated TTA design flow is proposed that exploits execution trace information and a greedy algorithm to give a very fast runtime.
- We compare the instruction word and power consumption effects of an optimized IC to a fully-connected TTA. To our best knowledge this is attempt to experimentally quantify the effects of IC design on TTA power.

This article is structured as follows. Section 2 is a brief discussion of related work. Section 3 describes the proposed design flow. In Section 4, the algorithm is evaluated by generating and synthesizing processors. Section 5 concludes the article and proposes future works.

2. Related work

Transport Triggered Architecture A TTA design space exploration flow described in [13] concentrates on selecting FUs and RFs, but includes an IC optimization pass which starts from a fully-connected IC and removes connections in a round-robin fashion from each bus, choosing the connection with least effect on program runtime. The algorithm appears to produce sparce ICs, at the

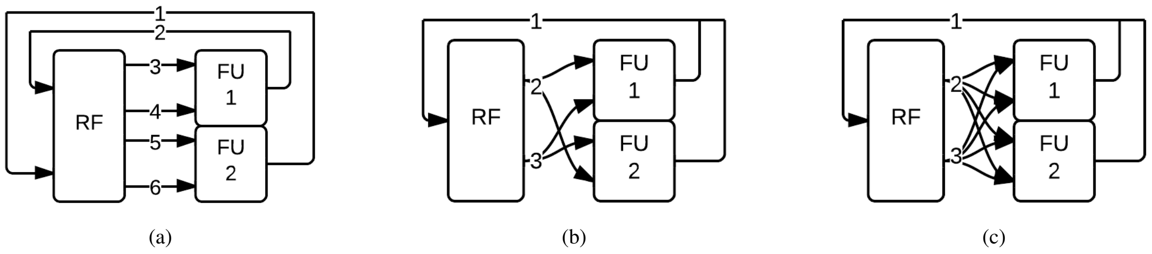


Figure 1: RF port sharing topology classification from [6]. (a) A baseline VLIW has idle ports, especially with software bypassing. (b) An example *direct connection* topology shares 3 RF ports between 6 FU ports so that each FU port is only connected to one RF port. (c) A *complete connection* gives better instructions per clock (IPC) than (b) since there are fewer resource conflicts, but is less energy-efficient due to IC overhead.

cost of a very long runtime. In [17], a C-to-FPGA ASIP design flow is proposed including a TTA design space explorer. Focus is again on exploring the number of function units and RFs. The IC is only optimized by removing entire buses or completely unused connections. The resulting ICs are almost fully connected.

Several studies propose to optimize the TTA instruction word through code compression, most recently *dictionary based compression* [10]. The MOVE-Pro TTA processor [9] reaches a code density similar to RISC despite its small fully-connected IC by means of a hand-optimized instruction encoding with separate instruction modes for, e.g., long immediates and jump addresses.

VLIW and exposed-datapath exploration Our simplified design space is similar to the problem of allocating function units to issue slots in a VLIW architecture, except finer-grained, since individual function unit ports are allocated separately. This is regarded in the literature as an difficult optimization problem. For example, Hekstra et al. explore FU allocation with a combination of design space partitioning, exhaustive search of significant variables, and greedy random variation of less-significant variables, over a runtime of two weeks [11]. This suggests that, though our simple greedy algorithm gives results close to prior art, more expensive global optimization might yield significant improvements. Due to the cost of FU allocation, Lapinskii et al. separate it into a later design stage, while earlier stages exploring e.g. clustering topology and the number of FUs use a coarse FU allocation model [15]. Lapinskii et al. overprovision each issue slot with every possible FU, followed by a machine shrinking optimization that removes FUs [14]. This approach is inapplicable here due to the possibility of placing ports of the same FU in separate slots.

Another conceptually similar technique in high-level synthesis is *resource sharing*, which merges resources that are never in simultaneous use. There is a large literature on the NP-complete graph-theoretic problem of finding maximal *cliques* of exclusively used resources which, e.g., e.g., Witte et al have applied to RF sharing [20]. In our problem space it is often a good tradeoff to merge resources that are used, e.g., 1% of the time and reschedule the program, so this technique was not used.

A recent study by Goel et al. characterizes the design space of RF port sharing, bypass-aware VLIWs [6]. They classify ICs into *complete*, *partial* and *direct* connections as shown in Figure 1. A full connection gives the best instructions per clock (IPC), but has worse energy efficiency than direct connection due to the overhead of the IC logic. Their results show significant performance differences between direct connection topologies, but no systematic method is suggested for finding an efficient topology.

Hoang et al. use design space exploration to prune the IC network in the FlexCore exposed-datapath architecture [12]. The crossbar IC of FlexCore is similar to our VLIW-TTA starting point. The algorithm first removes all unused *links*, i.e., crossbar multiplexer inputs, then proceeds to greedily remove least-used links, for significant core energy savings. The approach is complimentary to our algorithm, which would reduce the number of multiplexers in the crossbar-like IC and, therefore, control signals in the instruction word.

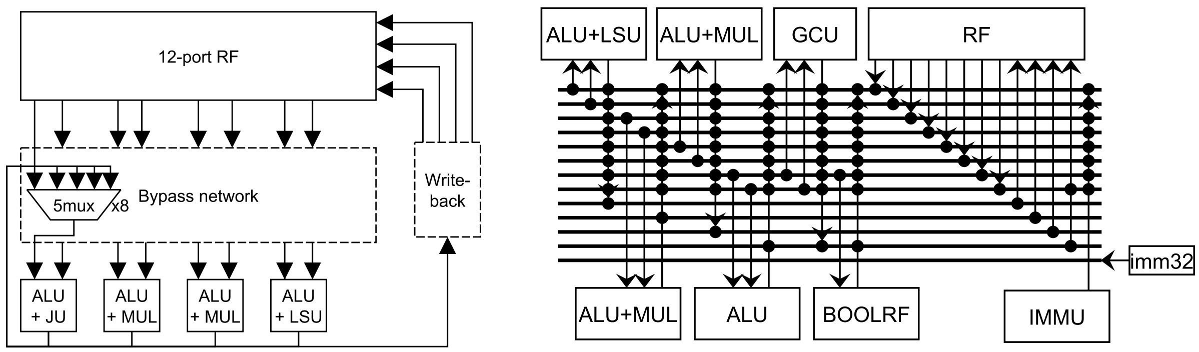
3. Exploration algorithm

In the proposed algorithm, we rely on the fact that conventional VLIW datapaths can be closely represented as TTA ICs; Fig. 2 shows an example. This kind of a *TTA-VLIW* datapath has favorable properties which make it interesting as a starting point for exploration. Since each function unit port is only connected to one bus, the encoding has no redundancy: there is only one way to transport data, e.g., from RF to LSU-ALU or back. This also simplifies the IC network logic: transports in a general TTA IC would be routed through two multiplexers, one for the bus and one for the input port, while TTA-VLIW requires only one.

However, a TTA-VLIW is inefficient in a different way: the register file has unnecessarily many ports and, since most operands are bypassed, the writeback buses are underutilized. Pruning bypass connections as in [12] is an unsatisfactory reduction step from this starting point since it has little effect on either issue. We instead shrink the baseline in by *merging* buses: we select two buses that are infrequently active on the same clock cycle, and combine them into a single bus with the connections of both. Redundant RF ports are removed after the merge. For example, if the first two buses in Fig.1(b) were merged, the LSU address and data ports would share a single RF port, and only one of these could be written in the same clock cycle. The instruction word is consequently shorter by one RF index. This yields a design space similar to the *direct connection* topologies in [6], except that input and output ports may be merged in the same bus. This may not have much effect on hardware complexity, but simplifies the instruction word.

This combination of a VLIW-TTA starting point and bus merges forms a small design space compared to the space of all possible TTAs. A natural way to traverse this space is using a greedy algorithm which, at each step, merges the buses with the least impact on performance. Due to the $O(n2)$ complexity of evaluating the performance costs of each bus pair, we approximate them using execution traces. The proposed explorer computes a covariance matrix of bus activities based on the previous round of simulation, and picks the bus pair with the lowest covariance, according to the following *IC pass* algorithm:

```
1: while busCount $> 1$ do
2:     busCov $\leftarrow [\,]$
3:     for $p = 1$ to benchmarkCount-1 do
4:         partialBusCov $\leftarrow [\,]$
```



(a) (b)

Figure 2: (a) 4-issue VLIW datapath with bypassing, similar to the default configuration of the HP VEX architecture [4]. Computational resources include four *arithmetic-logic units* (ALU), two *multipliers* (MUL), a *load-store unit* (LSU), a *jump unit* (JU) and a multi-port *register file* (RF), grouped into four issue slots. (b) Corresponding TTA architecture used as baseline for exploration. The first eight buses represent the "bypass network" while the next four buses represent "writeback". The "ALU+JU" issue slot is represented as three separate units in the TCE toolset: the general control unit (GCU) handles *call* and *jump* instructions and the boolean RF (BOOLRF) is used to implement all conditional behavior.

```
 5:     $\mathrm{cycleCount}, \mathrm{executions} \leftarrow \mathrm{simulate(p)}$
 6:     for $k = 1$ to instrCount do
 7:       for $i = 1$ to busCount-1 do
 8:         if hasMove(k,i) then
 9:           for $j = i + 1$ to busCount do
10:             if hasMove(k,j) then
11:               $\mathrm{partialBusCov}[\imath][\jmath]{+}{=}\mathrm{cycleCount}$
12:             end if
13:           end for
14:         end if
15:       end for
16:     end for
17:     $\mathrm{busCov}{+}{=}\mathrm{partialBusCov}/\mathrm{cycleCount}$
18:   end for
19:   $\mathrm{busA}, \mathrm{busB} \leftarrow \mathrm{minarg(busCov)}$
20:   merge(busA, busB)
21: end while
```

In the algorithm, the *simulate(p)* procedure performs cycle-accurate simulation on the pth benchmark program and produces an execution trace *executions[k]* as output: this tells the number of times the kth instruction in the program image was executed. The procedure *hasmove(k,j)* queries whether that instruction has a data transport on the jth bus. This information is used to compute the bus activity covariance matrix *partialBusCov*. If there are multiple benchmark programs, their covariance matrices are averaged together into *busCov*, whose lowest element selects the bus pair to be merged.

The algorithm always merges exclusively used bus pairs if available; otherwise the merge may cause performance degradation. In principle, the degradation is between *0..busCov*. At worst every instruction with simultaneous activity has to be split into two instructions, but usually the schedule has some slack, and some of the displaced operations can be moved to earlier clock cycles or different buses without degrading performance. Practical compilers are not guaranteed to produce optimal schedules, so the actual effect on performance may be larger than *busCov* or even negative.

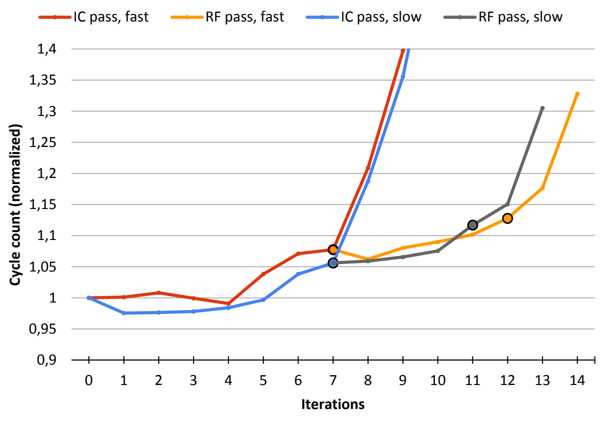
One challenge in this approach is immediate coding. The research platform used here [3] understands bus-specific *short immediates*, which are placed in the source field of a bus instruction slot, and *long immediates*, which replace one or more bus instruction slots, and are read from the *immediate unit*. Good performance requires access to abundant immediates of varying sizes, however, providing these in short immediates inflates the instruction word. An efficient design seems to require a long immediate, the placement of which is a nontrivial optimization problem. We co-optimize long immediate positioning with the IC network by constructing the initial machine with an unconnected dummy bus whose instruction slot hosts the immediate. A long immediate counts as a move on the bus hosting it, and transfers to the result when that bus is merged.

Greedy methods are simple and fast but may end up stuck in local optima. This effect is evident in the proposed method: if the greedy process has produced a well-balanced four-bus machine with the same amount activity on each bus, the next step must produce a three-bus machine with 50% activity on one bus and 25% on the others, which is clearly inefficient. This effect appears to be the most pronounced in very small machines, which might be feasible to handle using exhaustive search. The number of possible n-bus configurations reduced from k original buses is equivalent to the number of n-bin partitions of a k-element set, which may be computed using Stirling numbers of the second kind [7]. For example, a 3-bus machine reduced from a 2-issue TTA-VLIW with 7 buses has $S_7^{(3)} = 301$ possible configurations. Exhaustive search quickly becomes intractable for larger machines: a 6-bus machine reduced from 13 buses has $S_{13}^{(6)} = 9321312$ configurations. Consequently, some form of global optimization such as simulated annealing may be interesting as future work.

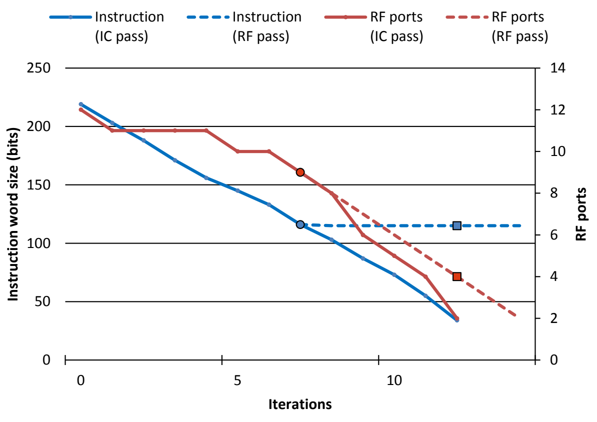
After the IC pass, though some RF ports are reduced, each bus still has at least one dedicated port. These are poorly utilized since most moves are bypasses. Therefore, we run a *RF pass* which computes covariance matrices of RF port accesses, and merges ports with low covariance. The algorithm is similar to the IC pass algorithm, except the *merge* procedure joins RF ports instead of buses. Moreover, we prohibit merging write ports with read ports.

3.1 Design flow

We implement an automated design flow based on the IC pass and RF pass algorithms as a design space explorer plugin in the TTA-based Codesign Environment [3]. The toolset has been used to implement most recent research TTAs such as [5]. Some of the toolset's components of interest are a GUI-based TTA design tool *ProDe*, a retargetable LLVM-based compiler *tcecc*, a cycle-accurate simulator *ttasim*, a modular design space exploration tool *explore*, and a processor RTL generator *ProGe*.



(a)



(b)

Figure 4: Exploration process in terms of (a) cycle count and (b) hardware complexity. Output configurations of each pass are emphasized.

Figure 3 shows the implemented design flow. The designer first specifies an initial machine configuration using ProGe and chooses a benchmark. If the benchmark includes multiple programs, these are weighed equally when measuring performance. After this, the explorer plugin processes the configuration first by repeated IC passes and then by repeated RF passes. The plugin repeatedly modifies the architecture through a C++ plugin API, compiles the benchmark for the changed architecture using *tcecc*, and performs traced simulation using *ttasim*. The produced architectures and simulation results are stored in a emphdesign database, and can be processed into synthesizable RTL, including a testbench for power estimation, using *ProGe*.

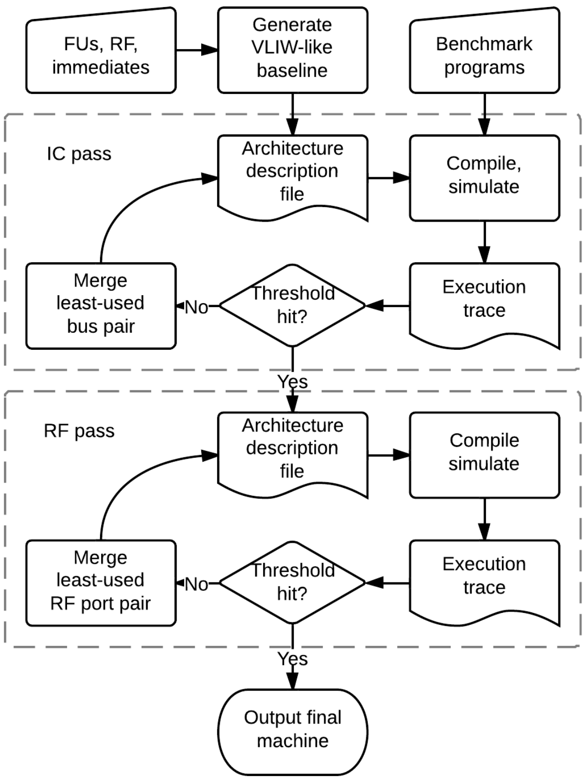


Figure 3: Proposed design flow.

The only decision points in the implemented design flow are switching to the RF pass and writing out the final configuration. These may be done manually by, e.g., running the IC pass until only one bus is left, plotting the resulting configurations as in Figure 4, and choosing a promising configuration. Another approach is to use a treshold to target a specific bus or port count, or to limit performance degradation with a cycle count threshold. These terminating conditions are somewhat unsatisfactory given that the designer is interested in finding a good tradeoff between IC simplification and performance penalty, which is likely to require manual trial and error. In the future it would be interesting to integrate the TTA power model of Pitkänen et al. [18] into the flow and automatically search for an energy-optimal configuration.

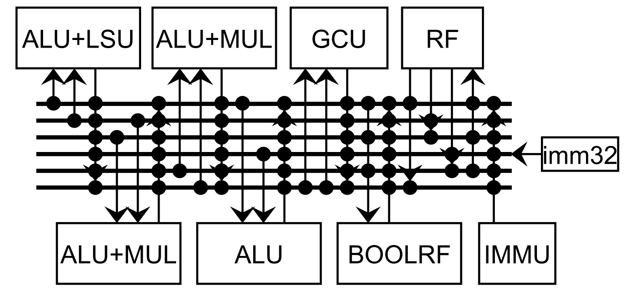
4. Evaluation

In this section the proposed design flow is evaluated by generating a sample processor starting from a 4-issue TTA-VLIW baseline, based on a subset of the CHStone high-level synthesis benchmark [8]. The following tests were used:

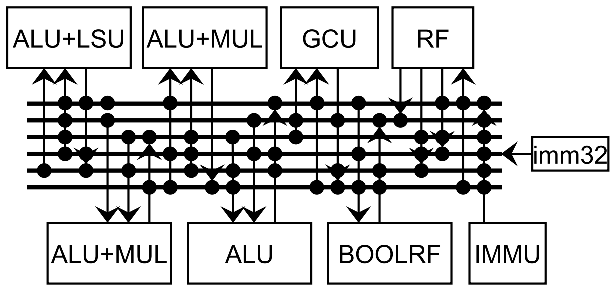
aes	AES encryption and decryption.
adpcm	ADPCM decoder and encoder.
blowfish	Blowfish encryption and decryption.
gsm	GSM linear predictive coding
jpeg	JPEG image decompression
motion	MPEG-2 motion vector decoding
sha	Secure hash algorithm

The design flow gives equal weight to each of the benchmark programs. We omitted the *softfloat* and *mips* benchmarks which are more interesting in high-level synthesis than as actual embedded processor workloads. When evaluating custom hardware, one source of uncertainty is the compiler: if the explorer manages to prune much of the VLIW datapath without degrading performance, this might be simply because the compiler is optimized for smaller IC, and generates poor code for the full VLIW. In order to increase confidence in the results, we used a baseline VLIW architecture close to the 4-issue default configuration of HP's VLIW Example (VEX) architecture [4], and compared cycle counts to the high quality VEX compiler. The baseline TTA is shown in 2.

The main decision in the proposed design flow is when to switch from IC pass to RF pass and when to halt. In this evaluation, we use a 15% cycle count threshold for both: exploration halts when a produced architecture gives an average execution time over all



(a)



(b)

Figure 5: Optimized TTAs: (a) our method and (b) round robin explorer [13].

benchmarks that is over 15% worse than the original configuration, and the previous architecture is returned.

Another question is whether the trace-based heuristics in the IC sweep and RF sweep algorithms are good approximations of performance. This was investigated by implementing a 'slow' option in the explorer plugin which measures the performance of each possible bus or RF port merge through simulation instead of using heuristics, according to the following algorithm:

```
1: while $\mathrm{busCount} > 1$ do
2:   $\mathrm{penalty} \leftarrow [\,]$
3:   for $p = 1$ to benchmarkCount-1 do
4:     $\mathrm{partialPenalty} \leftarrow [\,]$
5:     $\mathrm{originalCycleCount, executions} \leftarrow \mathrm{simulate(p)}$
6:     for $i = 1$ to busCount-1 do
7:       for $j = i + 1$ to busCount do
8:         $\mathrm{merge(i, j)}$
9:         $\mathrm{cycleCount, executions} \leftarrow \mathrm{simulate(p)}$
10:        $\mathrm{partialPenalty[\imath][\jmath]} \leftarrow$
11:          $\mathrm{cycleCount/originalCycleCount}$
12:        $\mathrm{undoMerge()}$
13:      end for
14:    end for
15:    $\mathrm{penalty{+}{=}partialPenalty/benchmarkCount}$
16:  end for
17:  $\mathrm{busA, busB} \leftarrow \arg\min_{\imath,\jmath} \mathrm{penalty[\imath][\jmath]}$
18:  $\mathrm{merge(busA, busB)}$
19: end while
```

The algorithm accumulates the cycle count penalties of each possible merge into the *penalty* matrix. The *undoMerge* function rolls back speculative merges. The actual implementation uses a *originalCycleCount* saved from the previous iteration. Many more simulation runs are performed than in the heuristic algorithm.

4.1 Exploration

Exploration with the trace-based plugin finished in 58 minutes (35 min for the IC pass and 23 min for the RF pass) and required 24 simulation runs. Optimization progress is shown in Fig. 4 and the generated processor in Fig. 5a. In this case, the IC is reduced from 13 to 6 buses before hitting the threshold. The four writeback buses are eliminated first, with negligible effect on performance. At very small bus counts, the cycle count begins to grow steeply. The IC pass results in a 6-bus configuration with 9 RF ports remaining (6 read, 3 write), which the RF port pass reduces to 4 (3 read, 1 write) before crossing the cycle count threshold. The optimized processor has a 115-bit instruction word as opposed to 128 bits for the VEX.

The 'slow' plugin finished in 780 minutes and 324 simulation runs. As shown in Figure 4, it generates higher-performance processors than the trace-based heuristic, but the difference is modest: on average 2,1% and at most 4%. For very small machines of 1..3 buses the slow method is actually worse, presumably due to the load balancing issue of the greedy algorithm.

Fig. 6a shows a per-program cycle count breakdown for the optimized processors, a fully-connected TTA the size of the final processor, and VEX. The results are mixed: our overall performance is better than VEX by a large factor, but much of the difference is due to our conditional execution support, which speeds up the branch-heavy inner loops in AES and Motion. We perform worse in the Blowfish test where the *tcecc* compiler fails to scalarize an often-referenced array into registers. Nevertheless, it appears that sufficiently high-quality code is generated.

4.2 Comparison

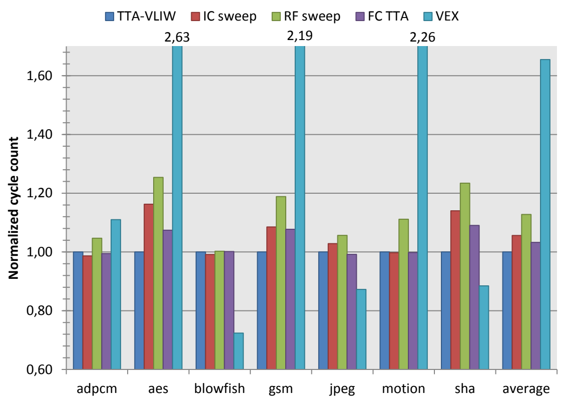
The processors generated by [17] are very close to fully connected. In order to obtain a better point of comparison, we reimplemented the round-robin IC optimizer in [13], and ran it on a fully-connected TTA the size of our optimized processor: 6 buses and 4 RF ports. A run in this configuration took 50 hours and 1410 simulation runs, and removed 119 connections. Figure 6b shows a comparison between the proposed method and round-robin. Several of the generated configurations have less connections and approximately 3.5% faster cycle count than the proposed method, however, exploration takes two orders of magnitude more time to run. The circled configuration was selected for comparison and is shown in Figure 5b.

The most visible difference is that the proposed method always connects each FU input port to exactly one transport bus, and each bus to at most one RF read port. This feature prevents replication of opcode information between instruction slots, which appears to have a small beneficial effect on code density: the RF sweep result has 3 less instruction bits than round-robin despite the sparser IC of the latter.

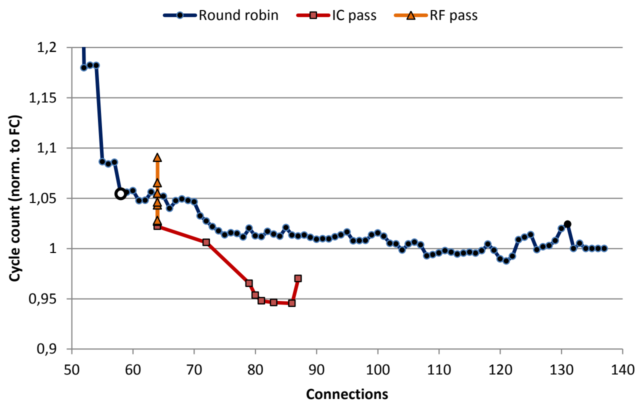
4.3 Synthesis

Intuitively, the improved fetch, RF, and IC power of the optimized TTA should compensate for the small decrease in IPC. In order to evaluate energy savings, we synthesized the TTA-VLIW and fully connected baselines and the optimized processor cores on a 1.5V 130nm ASIC technology and performed RTL power estimation. *gsm* was used as the power estimation workload since it appears representative of the averaged benchmark suite in Fig. 6a, and has a moderately high IPC. A low-IPC workload such as *jpeg* would give overly optimistic results since the 4-issue VLIW is overprovisioned for it in the first place, reducing the performance penalty incurred by shrinking the machine.

Instruction memory power was estimated using the CACTI tool. We assume that all the accesses fall in a 128-word L1 cache, which is divided into 4-7 32-bit banks and synthesized as 90nm low power SRAM, which is the coarsest technology supported by CACTI. These assumptions are optimistic so as to obtain a lower bound.



(a)



(b)

Figure 6: Cycle count comparisons: (a) TTA against VEX and (b) proposed explorer against the round-robin explorer in [13].

Data memory is not modeled since its utilization does not vary significantly between configurations, as the 64-word register file is sufficiently large to eliminate most spills into memory. Power estimation results are shown in Fig. 7a. The estimated power was multiplied by program runtime on each processor to obtain an energy breakdown, shown in Fig. 7b.

Approximately half the power in each configuration is dissipated on fetching and decoding instructions. The TTA-VLIW baseline suffers from a large RF and very poor code density, partly caused by the separate 32-bit immediate field but mostly by inefficient encoding. The fully connected machine has a more reasonable instruction word and a small RF, but its decoder and IC consume enough power to almost catch up with the TTA-VLIW.

The execution trace shows approx. 1 RF read and 0.5 RF writes per operation for TTA-VLIW, so the shown RF power is already reduced to roughly one-half by software bypassing. It seems likely that a straightforward VLIW would dissipate 15mW in the register file, so even the fully-connected machine may be an improvement. The optimized processors (RF pass and round robin) have small RFs, relatively short instruction words and lightweight ICs. Taking cycle count into account, both optimized processors dissipate 10% less energy per run than the fully-connected TTA.

5. Conclusion

We proposed a trace-based TTA design flow that generates architectures with efficient, sparse ICs two orders of magnitude faster than prior work, at the cost of a slightly lower IPC for the resulting processor. The design flow operates in a VLIW-like design space and might, therefore, be applicable to topology exploration for conventional VLIW RF port sharing.

Synthesis results support our hypothesis that the reported IC complexity and disadvantages of TTAs can be ameliorated by good IC design. A fully-connected TTA has a long instruction word relative to a comparable VLIW, and dissipates more power power in the IC than in the register file. The optimized TTAs have instruction word slightly shorter than a typical 4-issue VLIW. IC power is reduced by 44% (prior work) or 64% (proposed method).

The overall energy savings of 13% are modest since a large fraction of power is dissipated on fetching and decoding instructions. We are now working on wide-SIMD TTA processors where larger savings are expected, since the fetch and decoding logic is amortized across many data items and becomes insignificant. Wide-SIMD LIW processors are recently popular in radio baseband processing, and one recent wide-SIMD processor [21] dissipates 30% of power in the vector register file.

We also plan to optimize the instruction encoding for this new type of network. We estimate that better encoding would reduce the instruction bits of a TTA-VLIW by 30% and the optimized processor by 20% with little impact on performance. After these improvements, TTAs might become preferable to VLIWs in terms of code density, at least in dense inner loops. Moreover, we are experimenting with pruning bypass connections using an optimization pass similar to [12]. This might yield noticeable energy savings since the IC is on most critical paths.

Acknowledgments

This work was funded by the Academy of Finland (funding decision 253087), the Finnish Funding Agency for Technology and Innovation (project "Parallel Acceleration", funding decision 40115/13) and the ARTEMIS joint undertaking under grant agreement no 641439 (ALMARVI).

References

[1] H. Corporaal. *Microprocessor Architectures: From VLIW to TTA*. John Wiley & Sons, Chichester, UK, 1997.

[2] J.-L. Cruz, A. González, M. Valero, and N. P. Topham. Multiple-banked register file architectures. In *Proc. Int. Symp. Comp. Arch.*, pages 316–325, Vancouver, BC, Canada, 2000.

[3] O. Esko, P. Jääskeläinen, P. Huerta, C. S. de La Lama, J. Takala, and J. I. Martinez. Customized exposed datapath soft-core design flow with compiler support. In *Proc. Int. Conf. Field Programmable Logic and Applications*, pages 217–222, Milano, Italy, 2010. .

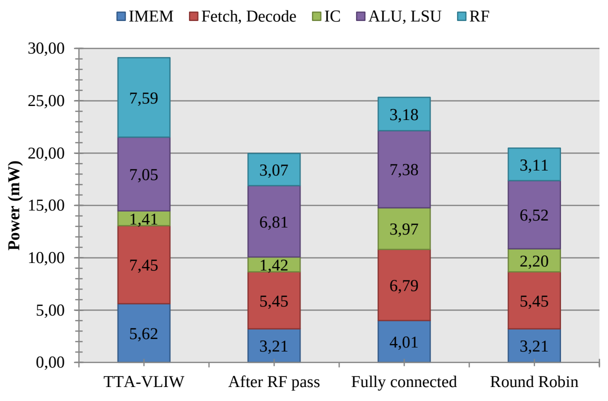
[4] J. A. Fisher, P. Faraboschi, and C. Young. *Embedded computing: a VLIW approach to architecture, compilers and tools*. Elsevier, 2005.

[5] A. Ghazi, J. Boutellier, J. Hannuksela, S. Shahabuddin, and O. Silven. Programmable implementation of zero-crossing demodulator on an application specific processor. In *IEEE Workshop on Signal Processing Systems*, pages 231–236. IEEE, 2013.

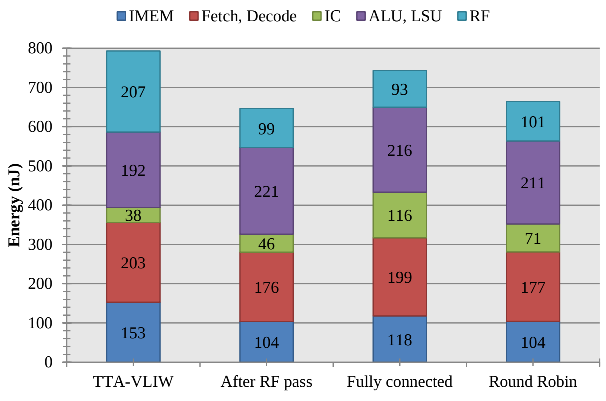
[6] N. Goel, A. Kumar, and P. R. Panda. Shared-port register file architecture for low-energy VLIW processors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(1):1, 2014.

[7] H. Gould et al. The q-stirling numbers of first and second kinds. *Duke mathematical journal*, 28(2):281–289, 1961.

[8] Y. Hara, H. Tomiyama, S. Honda, and H. Takada. Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis. *Inf. Media Tech.*, 4(4):740–752, 2009.



(a)



(b)

Figure 7: Estimated (a) power and (b) energy breakdowns in the *gsm* benchmark.

[9] Y. He, D. She, B. Mesman, and H. Corporaal. MOVE-Pro: a low power and high code density TTA architecture. In *Proc. Int. Conf. Embedded Comp. Syst.: Arch. Modeling Simulation*, pages 294–301, Samos, Greece, 2011.

[10] J. Heikkinen, J. Takala, and H. Corporaal. Dictionary-based program compression on customizable processor architectures. *Microprocessors and Microsystems*, 33(2):139 – 153, 2009.

[11] G. J. Hekstra, G. La Hei, P. Bingley, and F. Sijstermans. TriMedia CPU64 design space exploration. In *Computer Design, 1999.(ICCD'99) International Conference on*, pages 599–606. IEEE, 1999.

[12] T. T. Hoang, U. Jälmbrant, E. der Hagopian, K. P. Subramaniyan, M. Sjalander, and P. Larsson-Edefors. Design space exploration for an embedded processor with flexible datapath interconnect. In *IEEE Int. Conf. Application-Specific Syst. Arch. Proc.*, pages 55–62, Rennes, France, 2010.

[13] J. Hoogerbrugge and H. Corporaal. Automatic synthesis of transport triggered processors. In *Proc. First Ann. Conf. Advanced School for Computing and Imaging*, Heijen, The Netherlands, 1995.

[14] R. Jordans, R. Corvino, L. Jozwiak, and H. Corporaal. Instruction-set architecture exploration strategies for deeply clustered VLIW ASIPs. In *Mediterranean Conf. Embedded Computing*, pages 38–41. IEEE, 2013.

[15] V. S. Lapinskii, M. F. Jacome, and G. A. De Veciana. Application-specific clustered vliw datapaths: Early exploration on a parameterized design space. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(8):889–903, 2002.

[16] R. Leupers. Instruction scheduling for clustered VLIW DSPs. In *Proc. Int. Conf. Parallel Architectures and Compilation Techniques*, pages 291–300. IEEE, 2000.

[17] T. Patyk, P. Salmela, T. Pitkänen, P. Jääskeläinen, and J. Takala. Design methodology for offloading software executions to FPGA. *J. Signal Process. Syst.*, 65(2):245–259, 2011.

[18] T. Pitkänen, T. Rantanen, A. Cilio, and J. Takala. Hardware cost estimation for application-specific processor design. In *Embedded Comp. Sys.: Architectures, Modeling, and Simulation*, pages 212–221. Springer, 2005.

[19] M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalom. Exploiting data forwarding to reduce the power budget of VLIW embedded processors. In *Proc. Conf. and Exhib. Design, Automation and Test in Europe*, pages 252–257. IEEE, 2001.

[20] E. M. Witte, A. Chattopadhyay, O. Schliebusch, D. Kammler, R. Leupers, G. Ascheid, and H. Meyr. Applying resource sharing algorithms to adl-driven automatic asip implementation. In *Proc. IEEE Int. Conf. Computer Design*, pages 193–199. IEEE, 2005.

[21] M. Woh, Y. Lin, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, R. Bruce, D. Kershaw, A. Reid, M. Wilder, et al. From SODA to scotch: The evolution of a wireless baseband processor. In *IEEE/ACM Int. Symp. Microarchitecture*, pages 152–163. IEEE, 2008.