Scheduling for Register File Energy Minimization in Explicit Datapath Architectures

Dongrui She, Yifan He, Bart Mesman, Henk Corporaal
Department of Electrical Engineering, Eindhoven University of Technology
{d.she, y.he, b.mesman, h.corporaal}@tue.nl

***Abstract*—In modern processor architectures, the register file (RF) consumes considerable amount of the processor power. It is well known that by allowing software to have explicit fine-grained control over the datapath, the transport-triggered architectures (TTAs) can substantially reduce the RF traffic, thereby minimizing the RF energy. However, it is important to make sure that the gain in RF is not cancelled out by the overhead due to the fine-grained datapath control, in particular, the deterioration of code density in conventional TTAs.**

In this paper, we analyze the potential of minimizing RF energy in MOVE-Pro, a TTA-based processor framework. We present a flexible compiler backend, which performs energy-aware instruction scheduling to push the limit of RF energy reduction. The experimental results show that with the proposed energy-aware compiler backend, MOVE-Pro is able to significantly reduce RF energy compared to its RISC/VLIW counterparts, by up to 80%. Meanwhile the code density of MOVE-Pro remains at the same level as its RISC/VLIW counterparts, allowing the energy saving in RF to be successfully transferred to total energy saving.

***Index Terms*—TTA, MOVE-Pro, Low Power, Code Generation, Register File**

I. Introduction

In the coming years, embedded systems, especially the ones in mobile devices like smart phones, are becoming more and more important in everyday life. The rapid development in embedded processors enables such devices to run high performance applications like wireless communication and high definition video codecs. However, power efficiency is becoming the dominant determinant of embedded system design: in most cases, embedded systems have limited power sources like batteries, which greatly constrain the use of high performance processors. Moreover, high power dissipation makes the chip's thermal design more difficult. A lot of work has been done in different ways to reduce the processor power. We observed that a substantial part of the energy is consumed by register file (RF). Therefore in this paper, we particularly focus on reducing energy consumption of the RF, and converting this energy saving into the total core energy reduction.

In typical embedded application kernels, most of the variables have very low use count. Table I shows the local variable statistics of five representative kernels. In a pipelined processor, a lot of such variables can be accessed via the bypassing network instead of the RF. However, in conventional processor architectures, these RF accesses are usually performed regardless of the necessity unless relatively complex pre-decode logic is introduced, which result in a power hungry RF and a hotspot on the chip.

This work is supported in part by the Dutch Technology Foundation STW, project NEST 10346.

Table I: Kernel local variable statistics in basic blocks

	Histogram	FIR	IDCT	YUV2RGB	MatVec
Avg. reads per var.	1.59	1.71	1.55	1.77	1.55
Local var. with < 3 uses	98.25%	94.10%	98.72%	90.31%	85.00%

Intuitively, having fine-grained control over the datapath can reduce the unnecessary RF accesses. The transport triggered architecture (TTA) is an ideal candidate for this purpose. Because in a TTA the software explicitly controls every data movement in the datapath, making it relatively easy to reduce RF accesses [1]. However, a conventional TTA has drawbacks in code density and compiler design complexity, as well as unsatisfying power and performance results. MOVE-Pro is a TTA-based processor architecture [20]. It exploits the software bypassing flexibility in TTA, while a number of features are added to improve the efficiency over conventional TTAs. In this paper we present the design of a compiler backend for MOVE-Pro. The core of the MOVE-Pro compiler is the scheduler. Apart from reducing the number of RF accesses, the scheduler also needs to ensure that the schedule is as short as possible, since its length has great impact on both performance and energy consumption. Experimental results show a significant RF energy reduction: up to 80%. The energy saving is still significant when the energy consumption of instruction memory is taken into account.

This paper makes the following contributions:

- We introduce the design of a new compiler for MOVE-Pro using a graph-based resource model for the datapath. Compared to the compiler in [8], the proposed design has the following advantages: 1) the resource model is much more flexible, which enables code generation for less regular architectures; 2) with the proposed model, the compiler can integrate more passes to improve code generation results.
- Based on the resource model, max flow calculation is used to ensure deadlock freedom, which guarantees that the scheduler can generate valid schedules.
- Comprehensive comparison of energy consumption between MOVE-Pro and its RISC/VLIW counterparts is performed, demonstrating the effectiveness of the proposed design.

The remainder of this paper proceeds as follows: Section II introduces the basic idea behind register file access reduction. MOVE-Pro, the TTA-based architecture proposed in [20] is briefly presented in Section III. The proposed energy-aware compilation flow for MOVE-Pro is depicted in Section IV. In Section V, the proposed design is verified with a detailed

978-3-9810801-8-6/DATE12/©2012 EDAA

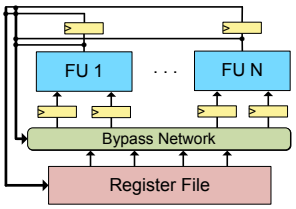


Figure 1: Operand bypassing in a typical VLIW processor datapath

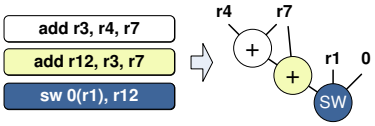


Figure 2: RF access elimination example

comparison between MOVE-Pro and its RISC/VLIW counterparts. Section VI discusses related work. Finally, Section VII concludes our findings and discusses future work.

II. Register File Access Reduction

In processors designed for multi-media and/or high performance signal processing, the RF is one of the most power hungry components in the datapath, which could account for over 40% of the datapath power consumption [3]. For multi-issue architecture, the demand for RF with many ports is especially costly [12]. In this work, we focus on saving the RF power consumption by reducing the number of RF accesses.

Fig. 1 shows the datapath of a typical VLIW processor. To reduce pipeline stalls caused by data dependencies, a bypass network is usually employed to allow an instruction to use the results which are available in the pipeline but haven't yet been written back to the RF. With the bypass network, there are three situations where RF accesses are not necessary:

- *Bypassing*: the result of an operation can be read from the pipeline register before it is written back to RF.
- *Dead result elimination*: if all uses of a variable are bypassed, writing it back to the RF is not necessary.
- *Operand sharing*: when an operand is used successively on the same port of a function unit (FU), it only needs to be read once.

The left side of Fig. 2 shows an example in which all three types of RF access eliminations are possible:

- Register r3 in the second addition (*add*) and r12 in the store instruction (*sw*) can be bypassed;
- After bypassing, values of r3 and r12 can be discarded;
- Register r7 is shared by the two additions, so only the first instruction needs to actually read r7 from the RF.

In Table II, we depict five representative streaming kernels which are studied in this paper. Table I shows the percentage of local variables with less than 3 uses in these kernels. Clearly, most variables have very small use counts, which can be interpreted as a huge potential to eliminate RF accesses.

III. Exposed Datapath Architecture

Architectures like RISC, Superscalar, and VLIW can be categorized as operation-triggered architectures. In such architectures, an instruction typically specifies what the operation is and what the required operands are. Usually, they cannot eliminate the unnecessary RF accesses identified in Section II.

Table II: Kernel description

Kernel	Description
FIR	5-tap 32-bit finite impulse response filter
Histogram	256-bin histograming for 8-bit gray-scale image
YUV2RGB	YUV to RGB color space conversion for 24-bit image
IDCT	Inverse cosine transformation in the JPEG decoder
MatVec	Matrix vector multiplication

In a transport-triggered architecture (TTA), however, instructions control the datapath by specifying the data transportation between different units, and operations are merely side-effects of the transportation [1], as shown in Fig. 3. TTA is well-known for its cost-effective trade-off between performance and flexibility. It is able to generate optimized cores for various domains, e.g., multimedia, telecommunication [1], [2], [16].

In this work, we explore TTA's low-power potential inherited from its *explicit datapath* nature, i.e., the capability of directly transferring an operand from the output of one unit to the input of another. With proper scheduling of the data transport, RF accesses can be dramatically reduced compared to its RISC/VLIW counterparts. As a result, the total processor energy consumption is expected to be reduced accordingly. However, it is challenging to achieve this goal due to some disadvantages of conventional TTAs:

- Code density is likely to be lower than RISC/VLIW [13].
- The separate scheduling of source operands increases circuit switching activity, causing more power consumption.
- Operations can only be triggered by moving data to the trigger port, introducing extra scheduling constraints.
- Inefficient support for large immediate, which deteriorates code density and power consumption [1].

To fully utilize TTA's low-power potential and to convert the energy saving in RF to the energy saving of the complete processor core, we proposed MOVE-Pro, a TTA-based processor framework [20]. A brief description of the MOVE-Pro architecture is presented in the remainder of this section.

A. MOVE-Pro Architecture

The block diagram of the MOVE-Pro architecture is shown in Fig. 4. The main differences between MOVE-Pro and conventional TTAs are:

1) Output Buffer: Each output port of the FUs is connected to a small buffer, which is a first-in-first-out (FIFO) queue and all entries are accessible. The result of an operation is kept in the buffer until it is popped out by new ones. The use of output buffer greatly increases the chance of bypass, thereby eliminating unnecessary RF accesses.

2) Shadow Input Register: To avoid the extra FU circuit activity caused by separate scheduling of source operands, we introduce shadow input buffers for FUs with multiple input

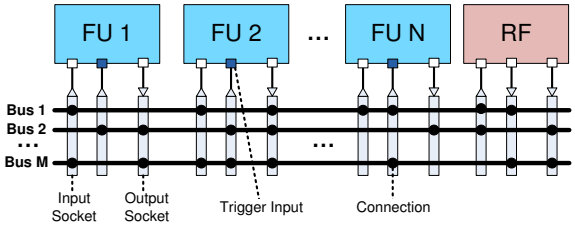


Figure 3: TTA architecture, with exposed inter FU and RF datapath

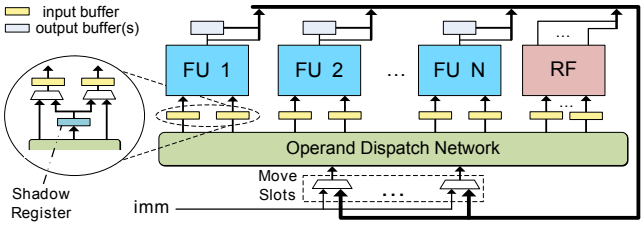


Figure 4: MOVE-Pro architecture

ports, where new input values of an operation are kept until the trigger arrives. The FU gets all the input operands in the same cycle, which greatly reduces FU circuit activities.

The use of shadow buffers has another important advantage: the result of an operation is available in the FU's output port until the next one is produced, which effectively makes the output port an extra entry of the output buffer.

3) Unified FU Input Port: In MOVE-Pro, an FU can be triggered by a *move* to any of its input ports. This provides more scheduling flexibility for the compiler compared to conventional TTAs. A MOVE-Pro instruction triggers the destination FU if its *opcode* field is set.

Similar to conventional TTAs, the connectivity of the *operand dispatch network* can be tailored based on application characteristics, which can reduce the power and area overhead introduced by the extra flexibility compared to RISC/VLIW architectures. The trade-offs in the connectivity of the network is beyond the scope of this paper. For the remainder of this paper, we assume that the network is fully connected.

B. MOVE-Pro ISA

Code density not only affects processing performance, but also has a strong influence on the total energy consumption. To improve the code density of conventional TTAs and to provide flexible immediate support, a new ISA is proposed [20].

MOVE-Pro uses a 16-bit *move* instruction format, specifying the source, which can be an FU, a register or an instruction's immediate field, and the destination, which can be the input of an FU or a register. When the destination is an FU, an opcode can be set if the move is a *trigger move*. Some helpful features are introduced to improve code density:

- A special type of move called *I-Move* is introduced, which can encode two moves to the same FU, one of them with a 16-bit immediate.
- Two kinds of branch instructions are supported: 16-bit *Short Branches*, with 10-bit offset, and 32-bit *Long Branches* with 26-bit offset.

IV. Energy-Aware Compilation

In TTA, a typical binary operation needs three moves: two for the source operands and one for the result. It typically takes 16 bits to encode a move. Therefore, a direct translation from *operation code* to *move code* would probably result in a very bad code density. Fig. 5 shows an example where move code is 66% larger in a direct translation. The problem can be solved by the compiler. In the example, after performing software bypassing and instruction scheduling, the final move code has the same number of instructions as the operation code, with half of the RF reads and all of the RF writes eliminated. Obviously the compiler plays an important role in

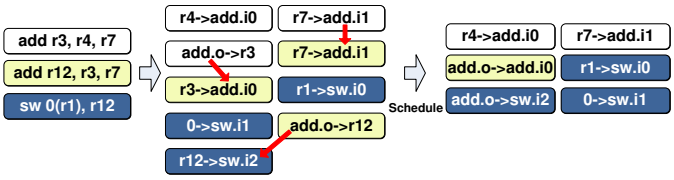


Figure 5: TTA code scheduling example

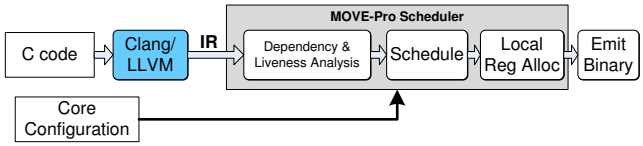


Figure 6: MOVE-Pro compiler framework

a TTA. In this section the design of the MOVE-Pro compiler backend, in particular, the instruction scheduler is discussed.

A. MOVE-Pro Compilation Flow

The compiler framework of MOVE-Pro is shown in Fig. 6. The LLVM compiler framework is used as the front-end, which produces intermediate representation (IR) for the back-end. The instruction set of the IR is similar to a RISC ISA, with support of extra meta-data. In this paper we assume that all operations in IR can be mapped to FU, so the backend does not need to perform instruction selection. Therefore the core of the backend is the instruction scheduler.

In this work, the scheduler performs basic block level scheduling. It minimizes the energy consumption by eliminating unnecessary RF accesses. The number of instructions is another optimization target. Otherwise the scheduler may choose to serialize operation execution in order to increase the bypassing opportunity. A post-pass register allocator is used. Due to the nature of TTA, spilling rarely happens in MOVE-Pro. When there is any spill code that fails to fit into the current schedule, the scheduler simply performs a partial reschedule.

The remainder of this section describes the model and algorithms used in the scheduler in detail.

B. Resource Model of MOVE-Pro

1) Resource Graph: A space-time representation of the datapath is used in this work. We first construct a spatial graph representation of the datapath, in which each allocatable resource is represented by a node, and edges are added between resources that can communicate within the same cycle. Then, the graph is duplicated for each cycle. Two types of edges between nodes in different cycles are added: 1) **move edges** are added between nodes that can communicate but with a latency, e.g., the edges from *Issue Slot* in t to *FU Input* in $t+1$; 2) a **storage edge** is added between the nodes for a resource in adjacent cycles if it can keep values, e.g., the edge from *RF* in t to *RF* in $t+1$. An example of the resource graph model is shown in Fig. 7. The edges between buffer nodes in different cycles enable the modeling of the FIFO behaviour. The output port of an FU is modeled as a buffer entry, as discussed in section III.

With the aforementioned resource graph, the problem of binding operations to FUs and scheduling data transportation becomes the problem of embedding the data-flow graph (DFG) of the program to the resource graph: operations are mapped

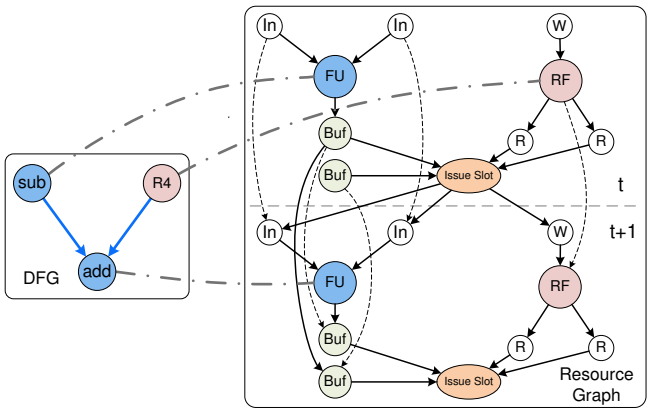


Figure 7: Resource graph and mapping example

to nodes representing the required resources, e.g., actual operations are mapped to FUs and live-in values are mapped to RF; a data dependency edge is mapped to a path from producer to consumer in the resource graph, with a few constraints:

- Paths with different producers cannot have common nodes, except for the ones that are actually representing multiple resource instances, e.g., the issue slot nodes;
- Paths with the same producer cannot split at issue slots;
- A path can only use buffers before its producer's result is popped out.

Compared to the compiler design in [8], the advantages of the proposed resource model are:

- It is possible to model almost any datapath structure with the proposed resource model, including irregular interconnect, clustered register files. This is particularly interesting for highly customizable architectures like TTA;
- With the proposed model, different passes can be integrated to improve code generation result. In this work, the FU assignment is integrated with instruction scheduling.

2) Checking for Deadlocks: In this work, we uses a forward operation scheduler, i.e., an operation is always scheduled before its consumers. Due to the FIFO behaviour of the FU output buffer in MOVE-Pro, a value is lost if it is still alive but not written back to RF when it is popped out. Therefore, it is critical for the scheduler to ensure that the output of every producer can always reach its consumers in a partial schedule. Otherwise the schedule cannot be complete, we call this a *deadlock*. In a partial schedule, a conservative approach to avoid deadlocks is to guarantee that all scheduled *live producers*, which are scheduled operations with live outputs, can be written back to RF. If that is the case, the consumers of the live producers can always get the value from the RF and it is guaranteed that there exists at least one valid schedule.

In this work, deadlocks in a partial schedule are checked by calculating the max flow of the *live-producer flow graph*, which is obtained from the resource graph. It contains:

- The FUs that live producers are mapped to, the nodes reachable by the live producers, and edges between them;
- A source node, and a unit capacity edge from the source to each live producer;
- A sink node, and an infinite capacity edge from each RF node to the sink.

In a partial schedule, all live producers can find a valid path to the RF if and only if the max flow of the flow graph equals to the number of live producers. Fig. 8 shows an example flow graph with two live producers on the same FU in two cycles.

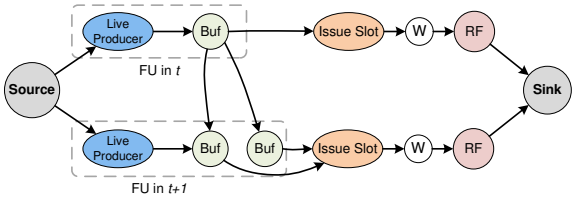


Figure 8: Flow graph for checking deadlocks

C. MOVE-Pro Instruction Scheduler

The MOVE-Pro scheduler performs forward scheduling, in which an operation can only be scheduled after its predecessors are scheduled. The scheduling is *operation-based*, i.e., the scheduler issues the selected operation in any time slot as long as it is valid. It is more flexible and effective than *instruction-based* scheduling, in which the scheduler tries to fill the issue slots on a cycle-by-cycle basis [5].

The scheduler uses modified list scheduling, as shown in Algorithm 1. The priority of the ready operations used by **select_next_operation** is calculated based on the mobility (critical path) and the estimated issue cost. When an operation is selected, the scheduler finds the earliest available FU. Then **find_min_cost_paths** tries to find shortest paths from all producers of the selected operation to the input ports of the FU. The weight of the edges are set based on the energy cost, and issue cost is added if a path increases the schedule length. So the search for shortest paths on the resource graph minimizes the energy consumption of the schedule whiling keeping the schedule short. As discussed in Section IV-B2, the max flow of the flow graph is calculated to check for deadlocks. The worst case complexity of finding the max flow is $O(V^2\sqrt{E})$, where V is the number of nodes and E is the number of edges. [15], but the number of live producers is typically very small. So the size of the flow graph is very small compared to the resource graph, and the max flow calculation is not expensive.

V. Evaluation and Analysis

A. Energy Consumption Model

In order to quickly validate the proposed energy-aware compiler backend and to illustrate its efficiency in our MOVE-Pro framework before implementing a complete processor design, we first estimate the potential energy saving by only focusing on the major components whose energy consumption are affected. The datapath of conventional VLIW processors and MOVE-Pro processors are very similar when they use the same amount/type of functional units and same bypass network (e.g., fully connected). The difference of energy consumption is mainly caused by the number of RF accesses and the instruction size [20]. For the same kernel, the instruction size of a conventional TTA is usually much larger than its VLIW counterpart [13].

Table III lists the average energy consumption of RFs with different number of ports and SRAMs with different data widths and sizes. The library we used in the experiment throughout this paper is TSMC 90nm low-power technology. The RF access energy is derived by synthesizing the RTL design, extracting the physical information, and estimating the

Algorithm 1: Basic Block Scheduling

Input : Operation DDG of the basic block B and machine model M
Output : Scheduled move instructions of the basic block M_B

```
// Set number of cycles based on conservative estimation
G ← initialize_resource_graph (B, M)
R ← ∅                                                  // Ready set
// Initialization: map live-in and immediate values
foreach o ∈ {x|x ∈ B, predecessors(x) = ∅} do
    Map o to RF in cycle 0 or immediate unit
    R ← R ∪ {s|s ∈ successors(o), s is ready}
end
S ← ∅                              // Set of scheduled operations
L ← ∅                                  // Set of live producers
while |S| ≠ |B| do                    // Schedule the basic block
    o ← select_next_operation (R)
    t ← ready_time (o)             // Earliest time to schedule o
    for t ← ready_time (o) to maximal_time (G) do
        value ← false
        F ← available_FUs (o, t)     // Set of possible FUs in t
        while F ≠ ∅ do
            // Find an FU, prefer sharing if possible
            f ← get_function_unit (o, F, G)
            F ← F \ f
            P ← find_min_cost_paths (o, f, G)    // See Sec. IV-C
            if P ≠ ∅ then
                reserve_resources (o, P, G)
                fg ← gen_flow_graph (L, G)    // See Sec. IV-B2
                // If no live producer is blocked
                if max_flow (fg) = |L| then
                    valid ← true
                    break
                end
            end
        end
        if valid = true then       // o is successfully scheduled
            break
        else            // Failed, release reserved resources
            release_resources (o, P, G)
        end
    end
    S ← S ∪ {o}
    // A producer of o is closed if o is the last user
    check_and_close_producers (o, L)
    if o has consumer then
        L ← L ∪ {o}
    end
end
M_B ← get_move_schedule (G, B) // Convert mapping to schedule
```

average toggle rate of each port by performing 1024 random access. The energy of the memory is estimated by CACTI [7].

B. Experimental Results

The five representative kernels listed in Table II are used in the evaluation. These kernels are compiled with our newly developed compiler. The compiled benchmarks are then executed on a cycle-accurate simulator to collect access statistics, from which the RF access energy and memory access energy of each kernel are calculated. In all the experiments, we conservatively set the RF to the same size for all the processors, even though the RF pressure in MOVE-Pro is much lower.

Fig. 9(a) shows the RF (32b×32 2R1W) energy consumption comparison among a RISC (*R1*, 32-bit instr.), a 2-issue MOVE-Pro (*M2*, 32-bit instr.), and a 3-issue MOVE-Pro (*M3*, 48-bit instr.). The results are normalized to the RF energy consumption of the RISC processor for each kernel. We can see that the MOVE-Pro processors significantly reduce the RF access energy consumption. Compared to its 32-bit RISC counterpart, *M2* saves an average of 72.6% write access energy and 65.6% read access energy. Increasing the issue width of MOVE-Pro can further decrease the traffic to RF as FU results may stay in the bypass network for a longer time, e.g., *M3* saves 26.0% of RF energy compared to *M2* for YUV2RGB. Similar results are observed when comparing VLIW processor and MOVE-Pro processors. A maximum of 82.3% is observed on YUV2RGB. It is worth mentioning that since the RF traffic is greatly reduced in MOVE-Pro, the requirement on the number of RF's read/write ports alleviates accordingly. With reduced RF read/write ports, *M4*' (4-issue MOVE-Pro, 64-bit instr., 2R1W RF) shows an additional RF energy saving of 15.3% compared to *M4*, which has a 4R2W RF.

Table III: Energy consumption of different data accesses

	32b×32 2R1W RF		32b×32 4R2W RF		8kB Memory		9kB Memory
	Read	Write	Read	Write	32-bit	64-bit	48-bit
Energy per access (pJ)	1.81	5.46	1.95	6.67	16.38	19.67	17.03

Table IV: Instruction counts of kernel loops on different processors

	RISC 2R1W (*R1*)	MOVE-Pro-2 2R1W (*M2*)	VLIW-2 4R2W (*V2*)	MOVE-Pro-4 4R2W (*M4*)	MOVE-Pro-4 2R1W (*M4'*)	MOVE-Pro-3 2R1W (*M3*)
Instr Size	32 bits		64 bits			48 bits
Histogram	10	10	8	8	8	8
FIR	20	20	12	12	12	14
IDCT	87	83	48	49	51	60
YUV2RGB	42	42	27	29	30	32
MatVec	22	22	13	14	14	17

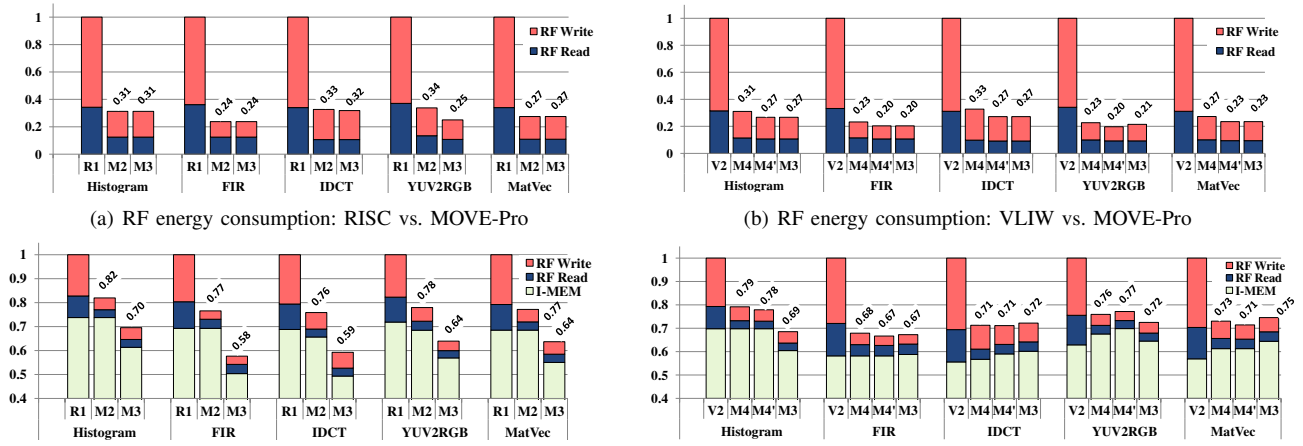
As discussed in Section V-A, the energy saving on RF access does not guarantee the energy saving in the whole processor. Conventional TTAs have poorer code density compared to their RISC/VLIW counterparts, which can easily eat up the energy saving on RF. The proposed compilation flow on our MOVE-Pro processors solves this issue. Table IV shows the code sizes of our MOVE-Pro processors and their RISC/VLIW counterparts. We can see that for the 64-bit cores, *M4* is only slightly worse in some kernels, while for the 32-bit cores, *M2* even has better code size than RISC. The comparison between *M4* and *M4*' shows that the number of RF ports has less impact on MOVE-Pro. It is interesting to mention that due to the finer scheduling grain (16-bit *move*), we can design MOVE-Pro processors like 48-bit *M3*. These kinds of intermediate solutions introduce more flexibility to application-specific designs.

We present the combined impact of the RF access energy and memory access energy in Fig. 9(c) and Fig. 9(d). The result shows that even when taking the code size into consideration, the energy saving is still significant. Comparing to a RISC, we achieve an average of 22.0% energy saving in *M2*, with a maximum of 24.2% on IDCT, while comparing to the VLIW, we see an average of 26.8% energy saving in *M4*', with a maximum of 33.3% on FIR.

VI. Related Work

In microprocessors, the register file is one of the central components, which accounts for considerable amount of energy. A detailed analysis of RF power consumption is given by Zyuban and Kogge in [22]. In [14], Rixner et al. analyze the trade-offs in different register file designs for media processors. Balfour et al. introduce the ELM architecture that reduces RF accesses by using a combination of software bypassing and hierarchical RFs [10], [11]. The compiler for ELM uses a similar way to check for deadlocks in partial schedules [6]. Compared to ELM, MOVE-Pro has finer grained scheduling and more explicit data path.

The TTA is proposed by Corporaal [1]. The MOVE [9] and the TTA-based codesign environment (TCE) [17] are the two most well known implementations of TTA. Hoogerbrugge [4]



(a) RF energy consumption: RISC vs. MOVE-Pro

(b) RF energy consumption: VLIW vs. MOVE-Pro

(c) RF + I-Mem energy consumption: RISC vs. MOVE-Pro

(d) RF + I-Mem energy consumption: VLIW vs. MOVE-Pro

Figure 9: Energy consumption results. *R1*: RISC with 2R1W-RF; *V2*: 2-issue VLIW with 4R2W-RF; *M2*: 2-issue MOVE-Pro with 2R1W-RF; *M3*: 3-issue MOVE-Pro with 2R1W-RF; *M4*: 4-issue MOVE-Pro with 4R2W-RF; *M4'*: 4-issue MOVE-Pro with 2R1W-RF

and Janssen [5] analyze different aspects of compiler backend design for TTA. Guzma et al. discuss the performance impact of software bypassing in TTA [18], and the power consumption implication is in [19], but a comparison between TTA and other architectures is missing. In this paper, we present a comprehensive comparison between TTA and RISC/VLIW. Reshadi and Gajski present a compiler design for No-Instruction-Set-Computing (NISC) architecture [21]. NISC resembles TTA, but its control overhead is less of a concern as it mainly targets platforms that are able to reconfigure the control logic. In this paper we exploit the potential of building a TTA-based low power processor. The combination of MOVE-Pro and the proposed compiler overcomes most of the problems causing inefficiency in conventional TTAs.

VII. Conclusions and Future Work

In this paper, we presented the design of a compiler backend for MOVE-Pro, a TTA-based processor. A resource graph model is introduced, allowing the compiler to map and schedule efficiently without deadlocks. The proposed compiler is capable of generating energy efficient codes by reducing RF accesses while keeping the schedule as short as possible. In the experiments, we showed that compared to its RISC/VLIW counterparts, MOVE-Pro achieves significant RF energy reduction: up to 80%. When the energy consumption of instruction memory is taken into account, the effective energy saving is up to over 30%. It shows that the proposed design is able to achieve high energy efficiency in a complete core. In systems with large processing element array, the proposed method could result in substantial overall energy saving.

Further optimization is possible with the proposed compiler design. For example, with modification to the resource model, register allocation can be integrated into the proposed scheduler, which would be useful if the register file size is very small. Extending this work to support software pipelining would also improve the efficiency. Future work also includes extending this work to SIMD and multi-core architectures, which is expected to be even more energy efficient than current MOVE-Pro architecture.

References

[1] H. Corporaal. *Microprocessor Architectures: From VLIW to TTA*. Wiley, 1998.

[2] H. Corporaal and H. J. Mulder. Move: a framework for high-performance processor design. In *Proc. of the 1991 ACM/IEEE conference on Supercomputing*, pages 692–701, 1991.

[3] D. R. Gonzales. Micro-RISC architecture for the wireless market. *IEEE Micro*, 19:30–37, 1999.

[4] J. Hoogerbrugge. *Code Generation for Transport Triggered Architectures*. PhD thesis, Delft University of Technology, 1996.

[5] J. Janssen. *Compiler Strategies for Transport Triggered Architectures*. PhD thesis, Delft University of Technology, 2001.

[6] J. Park and W. J. Dally. Guaranteeing forward progress of unified register allocation and instruction scheduling. Technical Report 127, Stanford University, 2011.

[7] CACTI. cacti 5.3, rev 174. http://quid.hpl.hp.com:9081/cacti/.

[8] D. She et al. Energy efficient code generation for processors with exposed datapath. In *Proc. 9th Workshop on Optimizations for DSP and Embedded Systems (ODES-9)*, 2011.

[9] Delft University of Technology. MOVE. http://ce.et.tudelft.nl/MOVE/.

[10] J. Balfour et al. An energy-efficient processor architecture for embedded systems. *Computer Architecture Letters*, 7(1):29–32, 2007.

[11] J.Balfour et al. Operand registers and explicit operand forwarding. *Computer Architecture Letters*, 8(2):60–63, 2009.

[12] J.W. van de Waerdt et al. The TM3270 media-processor. In *Proc. of the 38th Int'l Symposium on Microarchitecture*, pages 331–342, 2005.

[13] O. Esko et al. Customized exposed datapath soft-core design flow with compiler support. In *Proc. of 20th Int'l Conference on Field Programmable Logic and Applications*, pages 217–222, 2010.

[14] S. Rixner et al. Register organization for media processing. In *Proc. of the 6th Int'l Symposium on High-Performance Computer Architecture*, pages 375–386, 2000.

[15] T. Cormen et al. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[16] T. Pitkanen et al. Low-power, high-performance TTA processor for 1024-point fast fourier transform. *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 227–236, 2006.

[17] Tampere University of Technology. TTA-based Codesign Environment (TCE). http://tce.cs.tut.fi/.

[18] V. Guzma et al. Impact of software bypassing on instruction level parallelism and register file traffic. In *Proc. of the 8th Int'l Workshop on Embedded Computer Systems*, pages 23–32, 2008.

[19] V. Guzma et al. Reducing processor energy consumption by compiler optimization. In *IEEE Workshop on Signal Processing Systems (SiPS)*, pages 63–68, 2009.

[20] Y. He et al. MOVE-Pro: a low power and high code density tta architecture. In *Proc. 11th Int'l Conference on Embedded Computer Systems (SAMOS-XI)*, pages 294–301, 2011.

[21] M. Reshadi and D. Gajski. A cycle-accurate compilation algorithm for custom pipelined datapaths. In *Proc. of the 3rd Int'l Conference on Hardware/Software Codesign and System Synthesis*, pages 21–26, 2005.

[22] V. Zyuban and P. Kogge. The energy complexity of register files. In *Proc. of the 1998 Int'l Symposium on Low Power Electronics and Design*, pages 305–310, 1998.