MOVE-Pro: a Low Power and High Code Density TTA Architecture

Yifan He
Eindhoven University of Technology, the Netherlands
y.he@tue.nl

Dongrui She
Eindhoven University of Technology, the Netherlands
d.she@tue.nl

Bart Mesman
Eindhoven University of Technology, the Netherlands
b.mesman@tue.nl

Henk Corporaal
Eindhoven University of Technology, the Netherlands
h.corporaal@tue.nl

***Abstract*—Transport Triggered Architectures (TTAs) possess many advantageous, such as modularity, flexibility, and scalability. As an exposed datapath architecture, TTAs can effectively reduce the register file (RF) pressure in both number of accesses and number of RF ports. However, the conventional TTAs also have some evident disadvantages, such as relative low code density, dynamic-power wasting due to separate scheduling of source operands, and inefficient support for variant immediate values. In order to preserve the merit of conventional TTAs, while solving these aforementioned issues, we propose, MOVE-Pro, a novel low power and high code density TTA architecture. With optimizations at instruction set architecture (ISA), architecture, circuit, and compiler levels, the low-power potential of TTAs is fully exploited. Moreover, with a much denser code size, TTAs performance is also improved accordingly. In a head-to-head comparison between a two-issue MOVE-Pro processor and its RISC counterpart, we shown that up to 80% of RF accesses can be reduced, and the reduction in RF power is successfully transferred to the total core power saving. Up to 11% reduction of the total core power is achieved by our MOVE-Pro processor, while the code density is almost the same as its RISC counterpart.**

***Index Terms*—TTA, Low Power, Code Density, Register File**

I. Introduction

In the coming years, mobile devices like smart phones are becoming more and more important in our daily lives. The rapid development in embedded processors enables such devices to run multiple applications with high performance demand, e.g., wireless communication and high definition video codecs. High performance often implies high power consumption. However, in most cases mobile devices have very limited power sources, such as batteries. Moreover, even when power supply is not an issue, high power dissipation makes the chip's thermal design much more difficult. So it is clear that power efficiency is the most important determinant of the processor design for mobile devices.

In typical embedded application kernels, most of the program variables have very low use counts. Fig. 1 shows the use counts within basic blocks of four typical image processing kernels. In a pipelined processor, many of the reads can be accessed via the bypassing network instead of accessing a register file (RF). And many of the RF writes can also be avoided because they could already be dead results after bypassing. However, in conventional processor architectures, these RF accesses are performed regardless of the necessity, resulting in unnecessary RF dynamic power wasting.

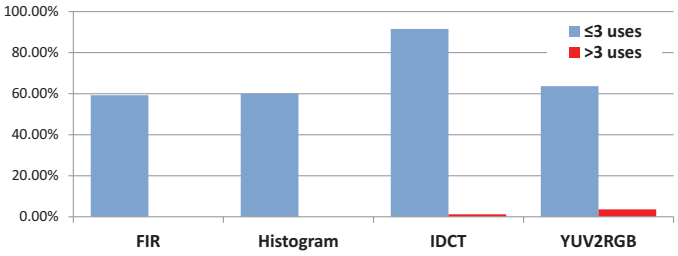

Fig. 1. Within-basic-block variable use counts. Here 100% accounts for all uses, including within-basic-block variable uses and across-basic-block variable uses.

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. In a TTA software controls every data movement in the datapath, thus removing unnecessary RF accesses through software bypassing is relatively easy [2]. However, the conventional TTA has drawbacks in code density and compiler design complexity, as well as unsatisfying power and performance results. In this paper, we propose MOVE-Pro, a TTA-based processor architecture, which exploits the software bypassing flexibility in TTA and is able to generate more efficient code for streaming applications. The RF access reduction in MOVE-Pro is significant: on average about 70% of the RF accesses are eliminated, which results in a dramatic reduction of the RF power. With the proposed optimization in MOVE-Pro, the RF power saving is fully transferred to a total core power saving of up to 11.6%.

This paper makes the following contributions:

- We propose MOVE-Pro, a low-power TTA-base processor architecture, which exploits the datapath flexibility of TTA and has better code density compared to conventional TTAs.
- A compilation flow for MOVE-Pro is also proposed. The compiler is able to generate power efficient codes for MOVE-Pro without compromising the performance.
- Detailed power consumption analysis of MOVE-Pro is performed, in comparison with a RISC-based processor with similar resources. To the best of our knowledge, this

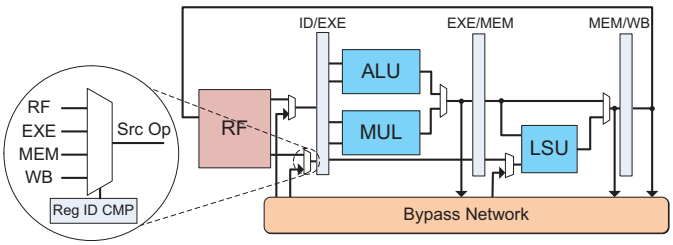

Fig. 2. Operand bypassing in processor datapath

is the first detailed analysis and comparison of the power consumption of a TTA-based architecture with its non-TTA counterpart.

The remainder of this paper proceeds as follows: Section II introduces the basic idea behind register file access reduction. MOVE-Pro, the proposed low-power and high code density TTA architecture is presented in Section III. The energy-aware compilation flow for MOVE-Pro is depicted in Section IV. In Section V, the proposed architecture is verified with a detailed comparison between one MOVE-Pro instantiation and its RISC counterpart. Section VI discusses the related work. Finally, Section VII concludes our findings and discusses future work.

II. Register File Access Reduction

The register file consumes about 15% of the core power within a typical single issue RISC processor, and even more in processors with more instruction level or data level parallelism [8], [18], [19]. This power consumption can be reduced by reducing RF accesses.

Fig. 2 shows the datapath of a typical RISC processor with a 5-stage pipeline. To reduce pipeline stalls caused by data dependency, a bypass network is usually employed to allow an instruction to use the results which are available but haven't been written back to the RF. With such a bypass network, there are three situations where RF accesses can be eliminated:

- *Bypassing*: if a source operand of an operation is the result of a previous operation and it is still in the pipeline, it can be read from the pipeline register instead of the RF.
- *Dead result elimination*: if all uses of a variable are bypassed, there is no need to write it back to the RF.
- *Operand sharing*: an operand only needs to be read from the RF once if it is used by successive operations on the same port of the same function unit.

The left side of Fig. 3 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.
- The write-back of r3 and r12 in the first two instructions can be eliminated if they are both bypassed.
- The two additions share the second operand (r7), therefore only the first addition needs to actually read the value from r7.

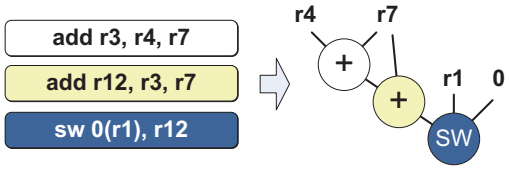

Fig. 3. RF access elimination example

TABLE I
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

Four representative image processing kernels are studied in this paper, as shown in Table I. Fig. 1 shows the variable use count of the four kernels after renaming all registers. Only the def-use pairs in the same basic blocks are counted and those variables live across basic blocks are conservatively considered non-eliminable. Despite this conservative assumption, we can see that there is a huge potential to eliminate many RF accesses.

III. Exposed Datapath Architecture

Architectures, such as RISC, Superscalar, and VLIW, can be categorized as operation-triggered architectures. Instructions specify operations and require this operation to trigger the transport of source and destination operands. Conventionally, they cannot eliminate the unnecessary RF accesses which are identified in Section II. In a transport triggered architecture (TTA), however, the instructions directly control the datapath by specifying the data transportation between different units, and operations are side-effect of the transportation [2]. Fig. 4 presents the conventional TTA architecture. By programming the communications, data are explicitly transported from one unit to another. Since TTA's interconnection network is exposed to the software level, RF accesses can be dramatically reduced by proper scheduling of the transportation.

One of our main goals in this work is to fully exploit the low-power potential of TTAs and to succeed in converting the energy-saving in RF to the energy-saving of the complete

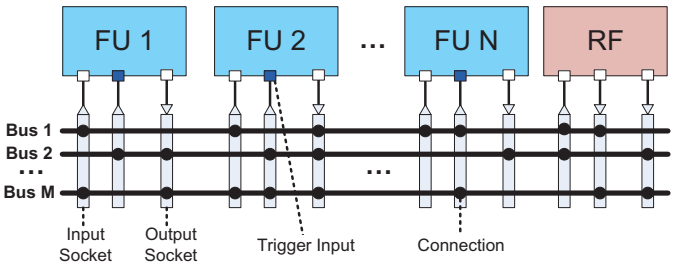

Fig. 4. TTA architecture, with exposed inter FU and RF datapath

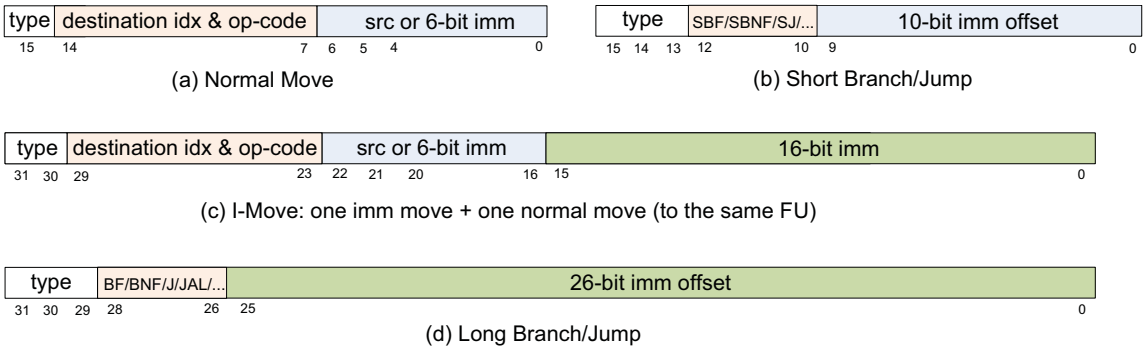

Fig. 5. The MOVE-Pro instruction formats

processor core. Another main focus of this work is to improve TTA's performance by optimizing its code density at various levels: instruction set architecture (ISA) level, architecture level, and compiler level. With these efforts, a high-performance and low-power TTA architecture is achieved.

A. Pros and Cons of Conventional TTAs

TTA offers a cost-effective trade-off between the size and performance of ASICs and the programmability of general-purpose processors. They can be efficiently used to generate optimized application cores in variant domains, e.g., multimedia, telecommunication. The main advantages of TTAs are listed below [2], [4], [14].

- Modularity, flexibility, and scalability. Quick application specific processor design.
- Reduced RF pressure in both number of accesses and number of RF ports.
- Extra freedom due to separate source operands scheduling.

However, the conventional TTAs also have some evident disadvantageous, which are mainly unsatisfying power and performance results, as well as the complexity of building an efficient compiler. We list a few insufficient aspects of conventional TTAs:

- Code density is likely to be lower compared to its RISC or VLIW counterparts [5].
- The separate scheduling of source operands increases circuit switching activity, which causes more power consumption. This will be explained in more detail in Section III-C.
- The difference between a normal data port and a trigger data port (shaded port in Fig. 4) introduces extra constraints for scheduling, which reduces flexibility and efficiency.
- Inefficient support for immediate values of variant lengths, which deteriorates code density and power consumption as well [2], [12].
- Lack/inefficient interrupt support.

In order to fully exploit the low-power potential of TTA, as well as solving these existing issues of the conventional TTAs, we propose MOVE-Pro, a new low-power and high code density TTA architecture as a counterpart of conventional processors. In this work, we focus on addressing the first four issues listed above. The solution for interrupt support will not be presented in this paper. Section III-B introduces the instruction set architecture of MOVE-Pro, and Section III-C describes the details of the MOVE-Pro architecture.

B. Proposed MOVE-Pro Instruction Set Architecture

Code density not only affects processing performance, but also has a strong influence on the total energy consumption. In order to improve the code density of conventional TTAs and to provide flexible immediate support, we propose a new ISA for our MOVE-Pro architecture. Some helpful features are introduced from the RISC instruction set. The MOVE-Pro instruction formats are shown in Fig. 5.

The 16-bit *Normal Move* (Fig. 5(a)) is similar to the conventional TTA instruction format. The main difference is that in MOVE-Pro we removed the constraint that operations can only be triggered by moving data to a specific destination register/buffer. Thus, the binding between trigger move and a specific destination port is decoupled. This improvement increases the scheduling freedom and reduces the compiler complexity, which improves code efficiency. To implement this idea with minimum overhead, we did not introduce an extra instruction bit to distinguish a trigger move operation and a non-trigger move operation. Instead, we consider all the non-trigger moves as a same class of operation (*Not-a-Trigger-Operation*), and code it into the *op-code* field of the instruction similar as *ADD*, *MUL*, et al.

A 32-bit *I-Move* (Fig. 5(c)) provides efficient support for 16-bit immediate. Two moves are encoded in this format: one is a normal move and the other is a 16-bit immediate move. The constraint here is that the destination of these two moves must be different input buffers of the same *Functional Unit (FU)*. This instruction format efficiently encodes the 16-bit immediate move, which again helps to improve TTA's code density. Since an immediate is always ready for issue (as it is encoded in the instruction, no extra dependency), the compiler can easily pack it with a 16-bit *Normal Move* to the same FU, and then form a 32-bit *I-Move* instruction.

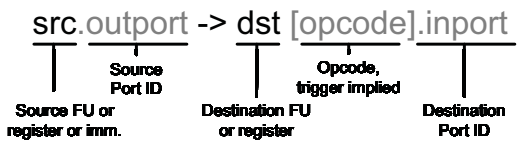

Fig. 6. MOVE-Pro assembly format

Two kinds of jump/branch instructions are supported, which only differs in the jump scope (range of offset). The purpose is for code density as well. The 16-bit *Short Branch/Jump* (Fig. 5(b)), which can be packed with a *Normal Move*, has an offset range of 10 bits. The 32-bit *Long Branch/Jump* (Fig. 5(d)) has a much larger branch scope, which is 26 bits.

Since 32-bit instructions are introduced in the proposed MOVE-Pro ISA, the minimum number of data-move issue slots in our MOVE-Pro processors is two. Fig. 6 depicts the MOVE-Pro assembly format.

C. Proposed MOVE-Pro Architecture

Section III-B described the main features of MOVE-Pro ISA. In this section the proposed low-power MOVE-Pro architecture will be presented in detail. Its simplified block diagram is shown in Fig. 7.

One limitation in the conventional TTA implementation is that the separate scheduling of source operands increases circuit switching activity. Take a multiplication, $a \times b = c$, as an example. In a RISC architecture, the multiplicand a and multiplicator b are scheduled together. They are latched into the two input operand registers of a multiplier at the same time. The multiplier circuit only toggles once before producing the product c. However, for a TTA-based processor, a and b can be scheduled separately. Before producing the required product c, the multiplier circuit may toggle one more time to produce an unused intermediate result. The extra switching activity causes dynamic power wasting. In the proposed MOVE-Pro architecture, we solve this issue by providing the option of introducing a shadow buffer for FUs with multiple input operands, i.e., multiple-operand operation (Fig. 7). Suppose FU-1 in Fig. 7 is a multiplier and the multiplicand a is scheduled earlier than the multiplicator b. In the proposed MOVE-Pro architecture, a will be first buffered in the shadow register until b is issued. They will then be loaded into the two input operand buffers at the same time. The multiplier is only activated once.

Introducing the shadow buffer has another important advantage. Let's take the same multiplication example again. Since the early arrived operand a will be buffered in the shadow register, the previous multiplication result(s) can be held in the output of the multiplier for much longer time without being flushed by a garbage data. This results in three beneficial outcomes:

- Reduced RF read accesses because the available time of previous results in the bypass network is increased.
- Reduced RF write accesses due to the increased possibility for the previous results to be a dead result.

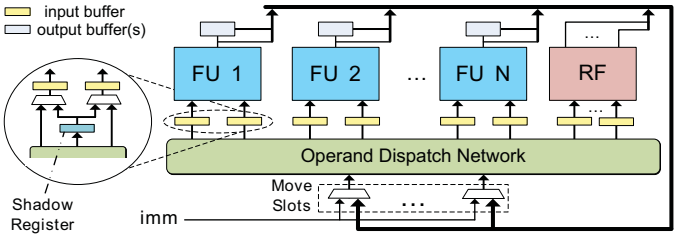

Fig. 7. MOVE-Pro architecture

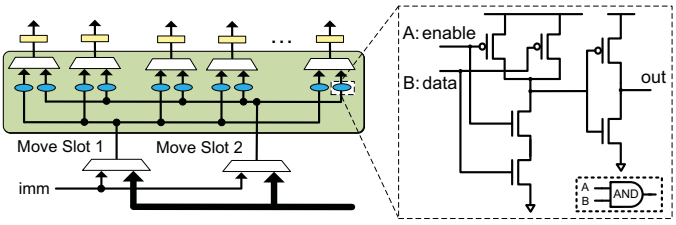

Fig. 8. Reduce the load capacitance by *AND*-gate isolation. *enable* is the destination write enable signal (early arrival signal) and *data* is the move data

- Reduced output buffers requirement.

The finest scheduling granularity of TTA is operand moves, which provides the potential extra freedom compared to its RISC/VLIW counterpart. However, this extra flexibility also introduces considerable amount of power penalty due to the fact that the load capacitance of the *operand dispatch network* could increase significantly. Each move slot can potentially send data to each destination location, while in the RISC/VLIW counterpart, the freedom is much less. They are usually bound to only one or a few input buffers of FUs. Reducing the *operand dispatch network* connectivity can mitigate this issue but at the cost of reduced scheduling flexibility.

Viewing the fact that only part of the destination registers (input buffers of FUs) are enabled per cycle, especially when the move issue width (i.e., the number of scheduled moves per cycle) is much smaller than the total amount of destination registers, we applied a circuit-level optimization as a more effective alternative. Fig. 8 describes the basic idea. An 2-1 *AND* gate is inserted before the *MUX*. One of the *AND* gate inputs, A, is the destination write enable signal, which is decoded in such a way that it arrives always earlier than the other input, B, which is the move data. When the destination register writing is not enabled, i.e., $A = 0$, the frequently changed move data, B, is isolated from propagating to the output, regardless of its value. With this method, the effective load capacitance is reduced, which results into a reduction in the dynamic power.

IV. Energy-Aware Compilation for MOVE-Pro

In TTA, a typical binary operation is translated into three moves: two for the source operands and one for the result. Usually, it takes 16 bits to encode a move. Therefore, a direct translation from operations to moves would almost definitely

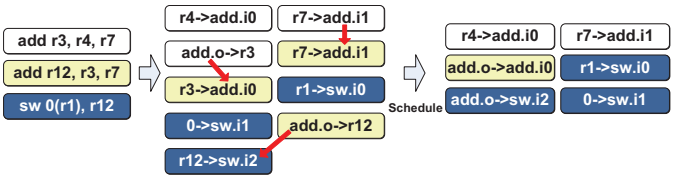

Fig. 9. TTA code scheduling example

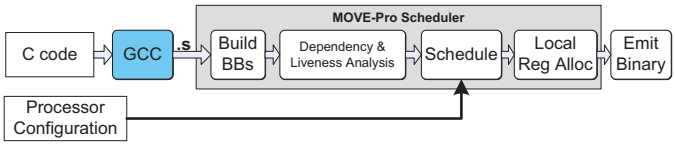

Fig. 10. MOVE-Pro compiler framework

result in a much worse code density than the corresponding RISC code. So the compiler plays an important role in a TTA-based architecture.

A small example is given in Fig. 9. In this example, a direct translation results in 60% increase in code size compared to the RISC code. However, after performing software bypassing and instruction scheduling, three out of six register reads are replaced with bypassing moves, and both write-backs are eliminated. The final code density is the same as the one of the RISC processor. In this section the design of the compiler back-end is discussed.

A. MOVE-Pro Compilation Flow

The compiler framework of MOVE-Pro is shown in Fig. 10. The GCC compiler is used as the front-end. The RISC assembly output of the GCC, which specifies the operations and dependencies between them, is used by the MOVE-Pro scheduler as the input. The goal of the scheduler is to minimize the energy consumption and maximize the performance. In this work, the scheduler achieves these goal by performing the following optimizations:

- Minimize the number of instructions.
- Minimize the number of register file accesses.

The MOVE-Pro scheduler processes the code in each basic block. It first analyzes the data dependency graph (DDG) of the basic block. After that, the scheduler schedules the instructions. Then before emitting the final code, a local register allocator is used to allocate registers. A partial reschedule of LSU related code is needed for such allocator if there is any spill.

B. Move-Pro Instruction Scheduler

As illustrated in Fig. 10, the central part of the MOVE-Pro compiler is the instruction scheduler. As described in Section III, the instructions in MOVE-Pro specify the data transportation between different resources, which can be categorized into two groups, namely, RFs and FUs. RFs are the data storage for the core while the FUs perform different operations. Fig. 11 shows the generic FU model for the

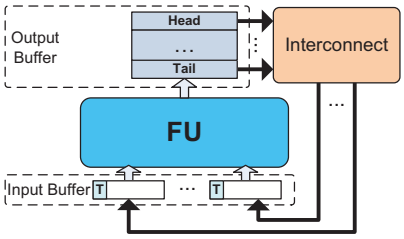

Fig. 11. Generic function unit model

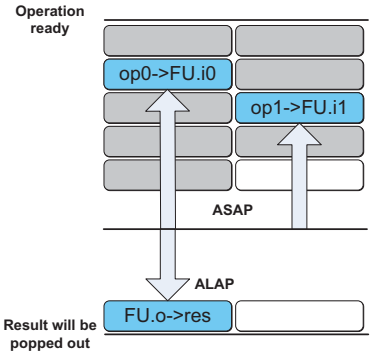

Fig. 12. Schedule the moves of a binary operation

MOVE-Pro compiler. The scheduler has to keep track of the state of each FU during the scheduling:

1) Source operands of an operation can be scheduled separately. An FU is occupied by an operation after the first source move, and is not available to other operations until the trigger move is scheduled.
2) The result of an operation is pushed to the tail of the output buffer when it is ready, which pops out the old value in the head of the buffer and shifts other values in the buffer by one step.
3) All entries of the output buffer are accessible via the interconnection network.

The MOVE-Pro instruction scheduler perform basic block level scheduling using the model described above. Similar to [9], the MOVE-Pro instruction scheduler uses an *operation-based* list scheduler which selects operation and schedule it before trying another one. The scheduler algorithm in this work, however, is different from the one in [9] in several aspects due to the differences in target architecture and optimization goal:

- There is no special trigger input port in a FU. Any source operand move can be a non-trigger move by setting the *op-code* field as *Not-a-Trigger-Operation*. And any source operand move can be a trigger move by setting the *op-code* field as a specific operation, e.g., *ADD*, *MUL*, et al.
- The scheduling of source moves and result moves of an operation are separated, as there is a high probability that the result is killed by bypassing.
- In the operation selection, more preference is given to the operation that can utilize bypassing, while in [9] the

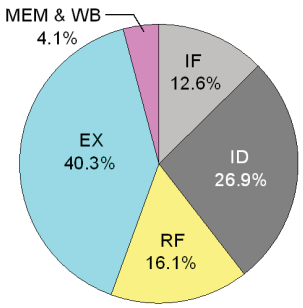

Fig. 13. Power breakdown of the reference RISC processor (YUV2RGB, 4.64mW)

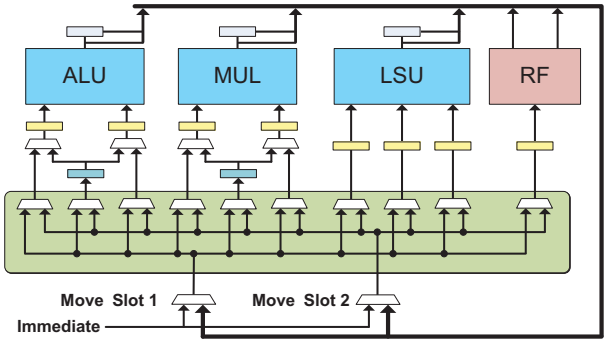

Fig. 14. The block diagram of the two-issue MOVE-Pro processor

scheduler gives priority only to the critical path.

Fig. 12 illustrates how the scheduler schedules an operation which has both source operand moves and result move. The compiler schedules the source moves and the result move separately. The following rules are used to decide which time slot the moves should be assigned to:

1) The source operand moves are scheduled as early as possible. This increases the possibility of reading the operand from the output buffer before it is popped out.
2) The write-back to RF is scheduled only when the result is about to be invalidated, or it is impossible to be eliminated due to its liveness. This reduces the chance of scheduling an unnecessary write-back.

When an operation is selected, the scheduler first save the result in the buffer head of the target FU if necessary. Then it schedules the source operand moves of the selected operation, and updates the liveness information of the results in the output buffer accordingly.

Table II shows the dynamic RF access reduction statistics obtained by cycle accurate simulation. In all tested kernels the RF access reduction is substantial, while the dynamic instruction counts in all tested kernels remain almost the same as they are in the RISC counterpart, i.e., there is no performance loss in the scheduling.

V. Experimental Results

A. Experimental Setup

To set up a practical reference, we first build a classical 5-stage RISC processor (Fig. 2) compatible with the OpenRISC ISA [13]. It is worth mentioning that to have a more fair and valid comparison, we heavily optimized the reference processor at both the micro-architecture and circuit level:

- Write data and write index are only propagated to the RF when RF write is enabled, which reduces redundant switching activity, thus saves dynamic power consumption in RF
- FUs are clustered, with dedicated input operand registers. This method reduced the unnecessary circuit switching activity inside the FUs
- Clock gating is applied onto the processor, including the register file

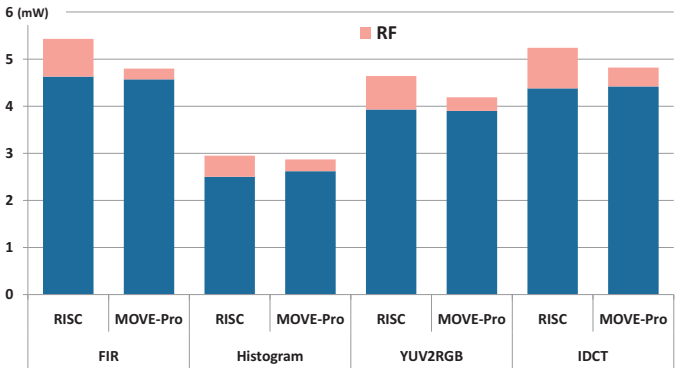

Fig. 15. Power consumption result

Fig. 13 shows the power breakdown of this reference RISC processor when running a YUV to RGB color space conversion kernel. Power consumption of the memory part is not counted as it highly depends on the memory size. The register file consumes about 16% of the total core power in this application.

We applied the same techniques on the two-issue MOVE-Pro processor implementation. Furthermore, we use the same FUs as in the reference processor. Fig. 14 shows the block diagram of this two-issue MOVE-Pro processor. The reason that we did not introduce shadow registers for LSU is that the LSU is pipelined into two stages, the first stage is a simple adder, which is not a power-hungry component, and we can reuse the pipeline registers inside the LSU to preserve the result.

Both processors are completely implemented in Verilog RTL and synthesized with the TSMC 90nm low-power technology. The maximum clock frequency is 200MHz with the critical path inside the multiplier. All four kernels listed in Table I are tested. Power consumptions are accurately estimated with the back-end information and real toggle rate generated by running each kernel with test image of 2000 pixels.

TABLE II
RF ACCESS REDUCTION IN MOVE-PRO

	FIR	Histogram	YUV2RGB	IDCT
Read	60.0%	57.8%	61.4%	60.15%
Write	82.3%	69.4%	73.5%	61.45%

TABLE III
POWER COMPARISON BETWEEN MOVE-PRO AND THE RISC REFERENCE

		Reference RISC	MOVE-Pro
FIR	Relative Cycle count	1.00	1.00
	RF Power	$0.80mW$	$0.23mW$ (-71.0%)
	Total Power	$5.43mW$	$4.80mW$ (-11.6%)
Histogram	Relative Cycle count	1.00	1.00
	RF Power	$0.45mW$	$0.25mW$ (-45.2%)
	Total Power	$2.95mW$	$2.87mW$ (-2.50%)
YUV2RGB	Relative Cycle count	1.00	1.02
	RF Power	$0.71mW$	$0.29mW$ (-58.4%)
	Total Power	$4.64mW$	$4.19mW$ (-9.70%)
IDCT	Relative Cycle count	1.00	1.01
	RF Power	$0.86mW$	$0.40mW$ (-53.4%)
	Total Power	$5.24mW$	$4.82mW$ (-8.85%)

B. Power Consumption and Dynamic Instruction Count Comparison

Fig. 15 shows the power breakdown of both the reference RISC and the two-issue MOVE-Pro processor when running the four benchmark kernels. We can see that the reduction in RF power is directly transferred to the total core power saving. The detailed comparisons are shown in Table III. In all kernels, the reduction of RF power is more or less proportional to the reduction of RF accesses. The saving of total core power is up to 11%.

According to Table III, the dynamic instruction count of MOVE-Pro processor is almost the same as its RISC counterpart. Under the same clock frequency, our MOVE-Pro has hardly any performance loss compared to the reference RISC processor. On the other hand, the code density and the dynamic instruction count of the conventional TTAs are likely to be much worse compared to their RISC or VLIW counterparts [5]. With further compiler improvements, such as software pipelining and if-conversion, we expect that the code density of MOVE-Pro can even outperform the density of a RISC, and it will give even better energy results.

C. Implication for Wide-Issued Architectures

In the experiment above, we compared a two-issue MOVE-Pro processor with its RISC counterpart. The register file used here is small, 32×32, and contains very few ports, two read and one write. It takes up only about 15% of the total RISC core power consumption. However, even with such a less power-hungry RF, we still demonstrated a substantial power saving.

TTAs are even more interesting when used as VLIW architecture alternatives. In VLIW processors, register files are usually much larger and contains much more read/write ports [18]. The detailed analysis in [17] shows that the total power dissipation of the central register file organization grows as N^3, while the total power dissipation of the distributed register file organization with two-port register files grows as N^2. Thus, we expect that our MOVE-Pro processors will exhibit even better power-saving characteristics when scaled to wider issue rates.

VI. RELATED WORK

In microprocessors, the register file is one of the central components, which can account for a considerable amount of the total energy consumption. A detailed analysis of RF power consumption is given by Zyuban and Kogge in [20]. In [17], Rixner et al. analyze the performance, area and power tradeoffs in different register file designs for media processors.

The idea of using variable liveness information to reduce register accesses in conventional processor architectures has been exploited in a few previous studies [1], [10], [11]. The work of [1] exploits the use of software bypassing and a FU buffer in a conventional architecture. In [19], Woh et al. address the similar problem in the SIMD context, and RF partitioning is used as the solution. As an alternative, TTAs provide a more natural and efficient solution to reduce RF accesses due to its transport-triggered feature. The TTA is proposed by Corporaal [2], [3], [4]. MOVE32INT is the first impementation of a TTA [15]. And the TTA-based codesign environment (TCE) is the latest implementation [16]. The MaxQ microcontroller designed by the Maxim is a commercial processor which exploits the concept of a TTA [12]. One of TTA's main advantages is the reduced register pressure. However, the conventional TTA has drawbacks in code density and compiler design complexity, as well as unsatisfying power and performance results. Guzma et al. discuss the performance impact of software bypassing in TTA [6], and the power consumption implication in [7]. However, the power aspects of TTAs are never studied in a detailed and systematic way. In this paper we proposed a novel TTA architecture, evaluated the potential of building a TTA-based low power and high code density processor, and performed a detailed comparison between TTA and its RISC alternative.

VII. CONCLUSIONS AND FUTURE WORK

Transport Triggered Architectures possess many advantageous, such as modularity, flexibility, and scalability. As an exposed datapath architecture, TTAs can effectively reduce the RF pressure in both number of accesses and number of RF ports. However, conventional TTAs also have some evident disadvantages, such as relative low code density, dynamic-power wasting due to separate scheduling of source operands, unequivalent normal data port and trigger data port, inefficient support for variant immediate values, etc. In order to preserve the merit of conventional TTAs, while solving these aforementioned issues, we proposed, MOVE-Pro, a novel low power and high code density TTA architecture. With optimizations at ISA, architecture, circuit, and compiler levels, the low-power potential of TTAs is fully exploited. Moreover, with a much denser code size, TTAs performance is also improved accordingly.

In the head-to-head comparison, we showed that up to 80% of RF accesses can be reduced with the proposed architecture. We successfully transferred the reduction in RF power to the total core power saving. The experiments showed that up to 11% reduction of the total core power is achieved, while the code density is almost the same as its RISC counterpart.

In this work we only compared a two-issue MOVE-Pro processor with its RISC counterpart. It is natural to extend this to a more parallel architecture such as a VLIW processor in future work. And although in the experiments the basic block scheduler is very efficient, in control-oriented codes, the limited scheduling scope makes it difficult to achieve good results. So in the future, it is essential to extend the scheduler to support more advance features such as cross basic block scheduling and software pipelining.

REFERENCES

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

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

[3] H. Corporaal. TTAs: Missing the ILP complexity wall. *Journal of Systems Architecture*, 45(12-13):949–973, 1999.

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

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

[6] V. Guzma et al. Impact of software bypassing on instruction level parallelism and register file traffic. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 23–32. 2008.

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

[8] Y. He et al. Xetal-Pro: An Ultra-Low Energy and High Throughput SIMD Processor. In *Proceedings of the 47th Annual Design Automation Conference*, pages 543–548, 2010.

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

[10] H. Kubosawa et al. A 2.5-GFLOPS, 6.5 million polygons per second, four-way VLIW geometry processor with SIMD instructions and a software bypass mechanism. *IEEE Journal of Solid-State Circuits*, 34(11):1619–1626, 1999.

[11] L. A. Lozano and G. R. Gao. Exploiting short-lived variables in superscalar processors. In *Proceedings of the 28th annual international symposium on Microarchitecture*, pages 292–302, 1995.

[12] MAXIM-IC. MaxQ Microcontroller. http://www.maxim-ic.com/products/microcontrollers/maxq.cfm.

[13] OpenCores. OpenRISC 1200. http://opencores.org/openrisc.

[14] 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.

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

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

[17] S. Rixner et al. Register organization for media processing. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, pages 375–386, 2000.

[18] J. van de Waerdt et al. The TM3270 media-processor. In *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 331–342, 2005.

[19] M. Woh et al. AnySP: anytime anywhere anyway signal processing. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 128–139, 2009.

[20] V. Zyuban and P. Kogge. The energy complexity of register files. In *Proceedings of the international symposium on low power electronics and design*, pages 305–310, 1998.