Energy Efficient Code Generation for Processors with Exposed Datapath

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

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

In a modern processor architecture the register file (RF) consumes considerable amount of power. Therefore it is important to reduce the RF accesses when designing an energy efficient architecture. It is well-known that with datapath exposed to software, the transport-triggered architectures (TTAs) can substantially reduce the RF traffic. In this paper, we analyze the potential of using MOVE-Pro, a TTA-based processor architecture. And we propose the compiler back-end for MOVE-Pro which can generate code that saves energy consumption by performing energy aware instruction scheduling to reduce RF accesses. The proposed architecture and compiler design is flexible. In the experiments we compare the proposed architecture with a RISC processor with the same resource, and achieve a reduction of RF accesses by up to 80%, which results in up to 11% saving in total core power. Meanwhile the dynamic cycle count remains almost the same as the reference processor, which means energy is saved without compromising performance.

Keywords

Low Power, Code Generation, Register File Power, TTA

1. INTRODUCTION

In the coming years, 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 multiple applications with high performance demand, e.g., wireless communication and high definition video codecs. However, high performance often comes with high power consumption, while 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 power efficiency is becoming the dominant determinant of the processor design for mobile devices. A lot of works have been done on different parts of the processor to reduce the power. We observed that a substantial part of the energy is consumed by the register file (RF). Therefore in this paper, we research methods to reduce the RF power. In particular, we focus on reducing unnecessary accesses to the RF.

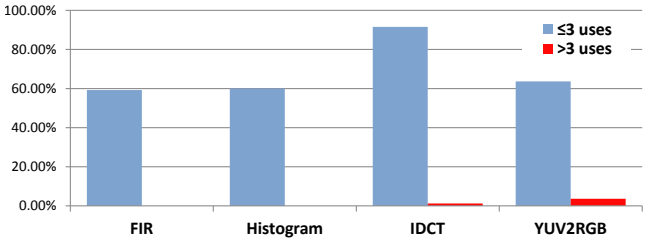

Figure 1: Variable use count in basic blocks

In typical embedded application kernels most of the variables have very low use count. Fig. 1 shows the use counts within basic blocks of four typical 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, resulting in a power hungry RF.

Intuitively, having fine-grained control over the datapath can reduce the unnecessary RF accesses. We propose MOVE-Pro, a transport-triggered architecture (TTA) based processor, in which the software controls every data movement in the datapath and thus removing unnecessary RF accesses is relatively easy. In such architecture, it is up to the compiler to produce efficient code and fully exploit the low energy potential. Since there is noticeable difference in architecture between MOVE-Pro and the conventional TTA, a new compiler back-end is required. The goal of this paper is to design the compiler back-end for the MOVE-Pro which can eliminate unnecessary RF accesses and thereby reduce the power consumption of the processor datapath. The result

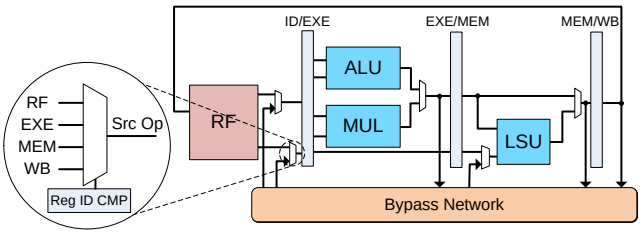

Figure 2: Operand bypassing in processor datapath

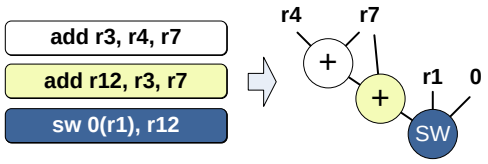

Figure 3: RF access elimination example

of RF access reduction in MOVE-Pro is significant: about 70% of the RF accesses are eliminated on average. The RF power reduces dramatically as a result. The saving in RF power leads to a noticeable reduction in total core power up to 11%.

This paper makes the following contributions:

- We propose the compiler back-end design for the TTA-based MOVE-Pro architecture. The compiler design is different from the ones for the conventional TTA due to the change of architecture and optimization goal. We show in the experiments that the proposed compiler design is efficient.
- Accurate power analysis is performed with a RISC reference processor optimized for power efficiency. We show that despite the more complex instruction decoding stage, MOVE-Pro can achieve higher power efficiency with the help of the compiler.

The remainder of this paper is organized as follows: the basic idea behind register file access reduction is introduced in Section 2. Section 3 presents the concept of TTA and the design of the TTA-based MOVE-Pro architecture. The design of the compiler back-end for MOVE-Pro is described in Section 4, with emphasis on the instruction scheduling. Section 6 discusses the related work. Finally, Section 7 concludes our findings and discusses the future work.

2. REGISTER FILE ACCESS REDUCTION

The register file consumes about 15% of the core power in a typical single issue RISC processor. Fig. 10 shows the power breakdown of the core when running the YUV2RGB kernel. The number becomes even higher in processors with more instruction level or data level parallelism [6, 16, 17]. 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 in the same port of the same function unit.

The left side of Fig. 3 shows an example in which all three RF access eliminations are possible.

- Register r3 in the second addition and r12 in the store instruction 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 therefore only the first one needs to actually read it from the r7.

Table 1: 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 kernels are studied in this paper, as shown in Table 1. 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.

3. ARCHITECTURE WITH EXPOSED DATAPATH

In conventional processor architectures such as RISC, VLIW and superscalar, an instruction controls the datapath by specifying the operation and its operands. The hardware is responsible for recognizing bypassing variables and selecting the proper source operands for each instruction. As stated in the Section 2, a lot of RF accesses can potentially be eliminated by using the bypass network. However, in conventional processor architectures, it is difficult to fully exploit such opportunities due to the following reasons:

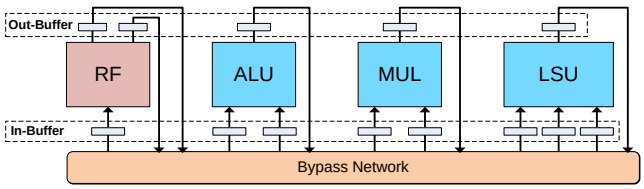

Figure 4: The MOVE-Pro architecture

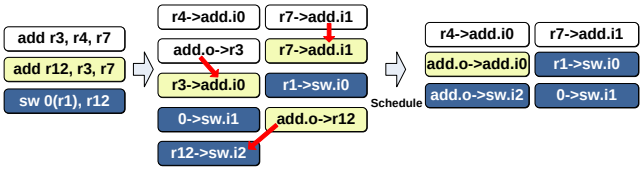

Figure 6: TTA code scheduling example

- Register indexes of the operands are unknown until the decoding stage, at which the register file is read. To eliminate unnecessary register reads, the processor has to perform extra checking before the decoding stage, which would either worsen the timing or increase the number of pipeline stages.
- As the liveness information of result variables is not encoded in the instructions, dead result elimination is almost impossible in many cases.
- Existing compilers usually do not optimize for bypassing in the instruction scheduling and register allocation passes.

These problems can be solved by giving the software more control over the datapath and making the compiler optimize for reducing register file accesses. The transport-triggered architecture (TTA) is an ideal architecture for this purpose. In TTA, instructions only specify the transportation of data between RF and FU or between different FUs. All the operations are side-effect of the data transportation [2]. In such a way, the communications between function units (FUs), including RF, is fully exposed to the software. Therefore, it is possible to eliminate unnecessary RF accesses by properly scheduling the data transportation.

3.1 MOVE-Pro Processor

There are several implementation of the conventional TTA, such as MOVE [12] and TCE [14]. The conventional TTA has limitation in a few things that can affect power efficiency:

- Code density is likely to be lower compared to its RISC or VLIW counterparts [3], which affects both performance and energy efficiency.
- The separate schedule of source operands increases circuiting switching activity, causing more power consumption.

In order to fully exploit its low-power potential and to solve the aforementioned issues of the conventional TTAs, we propose MOVE-Pro, a new TTA-based architecture, as a counterpart of conventional processors.

Compared to the original TTA [2], the Move-Pro has the following distinctive features:

- A move with 16-bit immediate can be packed in a 32-bit instruction with another move of the same operation.
- 26-bit immediate is supported in an 32-bit branch instruction.
- Operand and trigger signal are decoupled, i.e., the constraint that operations can only be triggered by moving data to a specific destination register is removed.
- Operands dispatch network is isolated from the computational logic, which reduces the circuit switching activity and increases the live interval of the results of each FU cluster.
- The output buffer is able to preserve the result as long as it is not popped out by new results. The isolated operand dispatch network enables the output port of the FU to act like an extra entry of the output buffer.

Fig. 5 shows the instruction format of the MOVE-Pro architecture. In this work, a processor instance of Move-Pro is built, which contains three FU clusters and a 32×32 RF, and executes 32-bit instructions, as illustrated in Fig. 4. Each FU has a one entry output buffer, which creates a two-entry logical buffer for the software. A new compiler back-end is required due to the differences in the architecture between MOVE-Pro and conventional TTAs.

4. 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 result in a much worse code density than the RISC code. Moreover, since there are only two move issue slots in the MOVE-Pro instance in this work, bad code density also results in bad performance. So the compiler plays an important role in a TTA-based architecture.

A small example is given in Fig. 6. 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 move, and both write-back moves 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. Section 4.1 introduces the overall compilation flow. The resource model used in the back-end is presented in Section 4.2. Finally the details of the basic block instruction scheduler are described in Section 4.3.

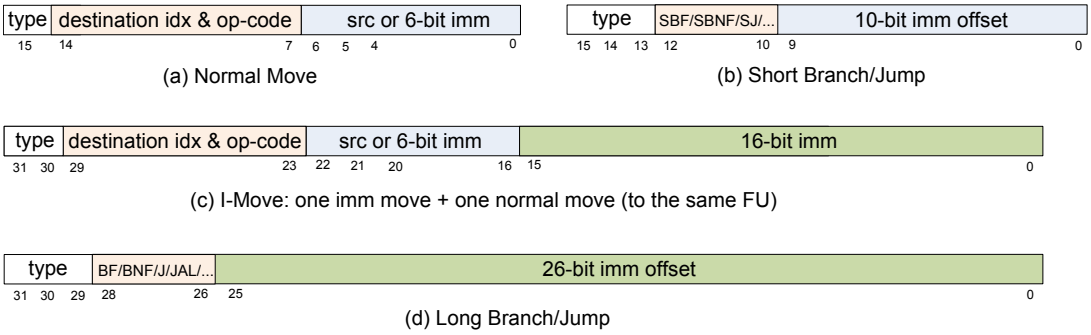

Figure 5: The MOVE-Pro instruction formats

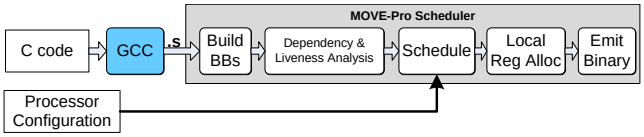

Figure 7: MOVE-Pro compiler framework

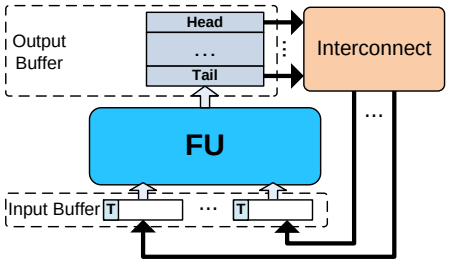

Figure 8: Generic function unit model

4.1 MOVE-Pro Compilation Flow

The compiler framework of MOVE-Pro is shown in Fig. 7. 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 the goal by performing the following optimization:

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

The MOVE-Pro scheduler processes the input in each basic block. It first analyze the data dependency graph (DDG) of the basic block. After that, the scheduler schedules the basic block. Then before emitting the final code, a local register allocator is used to allocate registers.

4.2 Resource Model for Instruction Scheduler

In MOVE-Pro, resources can be categorized into two groups, namely, register files and function units. Register files is the data storage for the core while the FUs perform different operations. Fig. 8 illustrates the generate model for FUs in MOVE-Pro. The scheduler has to keep track of the state of each FU during the scheduling:

(i) Source operands of an operation can be scheduled separately. The source does not produce any result until the operation is triggered. An FU is occupied by an operation after the first source move. And it is available to other operations after the trigger move if it is a pipelined or single-cycle FU. Otherwise the FU is available after the current operation is finished.

(ii) An operation can be triggered by moving an operand with trigger bit to any of the input registers. This is one of the most important differences between MOVE-Pro and the conventional TTA.

(iii) A FIFO-like buffer is used as the output buffer. When an operation is triggered in cycle t, its result pushed to the tail of the output buffer in cycle $t + l$, where l is the latency of this operation. In the same cycle, the value in the head of the buffer is popped out and no longer accessible.

(iv) All entries of the output buffer are accessible via the interconnection network, as opposed to a FIFO which only the head of the buffer can be accessed.

4.3 Basic Block Scheduling

In this work, the instruction scheduler performs scheduling at the basic block level. As discussed in [8], there are two ways to schedule move code:

- *Instruction-based scheduling* tries to pack as many moves as possible to an instruction and then proceeds to the next instruction.
- *Operation-based scheduling* selects an ready operation and try to place the moves of this operation.

The operation-based scheduling is likely to be better for TTA because in general it has a larger ready set than the instruction-based approach, i.e., more scheduling flexibility.

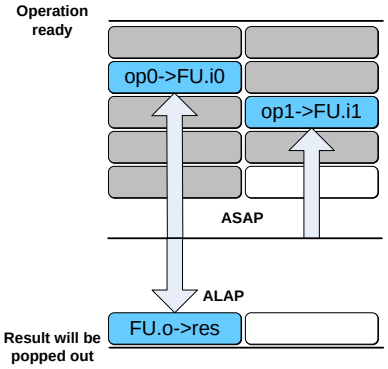

Figure 9: Schedule the moves of a binary operation

In this paper we use the operation-based approach. The complete basic block scheduling process is shown in Algorithm 1. The algorithm is different from the one in [8] in a few aspects due to the differences in architecture and optimization goal:

- The difference between operand and trigger move does not exist in MOVE-Pro. Any move can be a trigger move by setting a bit in the instruction. This results in more scheduling flexibility.
- The scheduling of source moves and result moves are separated, as there is a high chance that the result is killed by bypassing.
- In the operation selection, more preference is given to the operation that can utilize bypassing, while in [8] the scheduler gives priority only to the critical path.

An operation is ready and can be selected when all the dependencies are satisfied. When an operation is selected, the following two rules are used to decide which time slot the moves should be assigned to.

(i) The source operand moves are scheduled as early as possible.

(ii) The write-back move is scheduled only when the result is about to be invalidated, or it is impossible to be eliminated due to the liveness

The first rule increases the possibility of reading the source operand from FU output buffer, while the second one reduces the chance to schedule an unnecessary write-back. Fig. 9 illustrates the scheduling using these rules. When an operation is selected, the scheduler first save the result in the buffer head if it is alive (line 17 to 21), which ensures no result is lost. Algorithm 2 shows how the scheduler schedule the source moves of an operation. When a source move uses a bypassed value, the scheduler checks if all uses of that variable are bypassed. If that is the case, the variable is killed and does not need to be written back to register file.

Table 2 shows the dynamic RF access reduction statistics obtained by cycle accurate simulation. In all tested kernels

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 B_M
 1 R ← ∅                                              // Ready operation set
 2 foreach o ∈ B do   // Initialize ready set, o is operation in B
 3 |   if predecessors (o) = ∅ then
 4 |   |   R ← R ∪ {o}
 5 |   end
 6 end
 7
 8 // Size of worst-case schedule based on FU latency
 9 N ← maximal_schedule_size (B, M)
10
11 B_M ← {I_0, I_1 ··· , I_{N-1}}                  // I_i is empty instruction
12 S ← ∅                                            // Scheduled operation set
13 O ← ∅                                              // Operation result set
14 while |S| ≠ |B| do                          // Schedule the basic block
15 |   o ← select_next_operation (R)
16 |   f ← get_function_unit (o, M)
17 |   if outbuffer_head_alive (f) then  // Have to save the result
18 |   |   r ← outbuffer_head (f)
19 |   |   schedule result move of r to earlist possible empty slot
20 |   |   O ← O \ {r}
21 |   end
22 |   schedule_source_move (o, f, B_M)          // See Algorithm 2
23 |   if o has output then
24 |   |   O ← O ∪ {o}
25 |   end
26 |   S ← S ∪ {o}
27 |   foreach succ ∈ successors (o) do          // Update ready set
28 |   |   if predecessors (succ) ⊆ S then
29 |   |   |   R ← R ∪ {succ}
30 |   |   end
31 |   end
32 end
33 foreach r ∈ O do                          // Remaining alive results
34 |   f ← get_function_unit (r, M)
35 |   schedule result move of r to earlist possible empty slot
36 end
37 imax ← biggest_index_of_used_instructions (B_M)
38 B_M ← {I_0, I_1 ··· , I_imax}                // Trim the instruction list
```

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.

5. EXPERIMENT AND EVALUATION

To set up a practical reference, a 5-stage single-issue in-order RISC processor with the OpenRISC ISA [11] is implemented. From the resource perspective, it is comparable to the MOVE-Pro instance presented in Section 3. The reference processor is aggressively optimized for low power using the following methods:

(i) The pre-decoding logic in the IF stage eliminates the unnecessary RF read accesses. It refers to accesses which are caused by toggling of the register ID bits in the instructions where no RF access is needed.

(ii) Write data and write index are only propagated to the RF when RF write is enabled, which reduces dynamic power consumption in RF.

(iii) FUs are clustered and each cluster has separate operand registers, which reduces the unnecessary circuit switching activity inside the FUs.

(iv) Clock gating is applied to the processor, including the register file.

Algorithm 2: Schedule Operation Source Move

Input : DDG node of the operation to be scheduled o, the function unit f and the move instruction list $B_M = \{I_0, \cdots, I_{N-1}\}$
Output :

```
1  e ← earlist cycle in which f is available and o is ready
2  foreach s ∈ get_source_moves (o) do
3      si ← e                    // Index of the instruction to pack the move
4      while s cannot be packed in I_si do        // Find earlist slot
5          si ← si + 1
6      end
7
8      Add s to I_si
9
10     if source of s is already in destination register then
           // s does not occupy issue slot if it is not a trigger
11         Mark s as skipped
12     end
13     if s has producer then
14         p ← get_producer (s)
15         if s can use bypass read then
16             increase_bypass_count (p)
17             if all uses of p are bypassing read then
18                 Kill the result move of p
19             end
20         end
21     end
22 end
23
24 if o needs trigger then
25     Set trigger flag in last source move
26 end
27
28 if o has result then
29     f.push_result (o)
30 end
```

Table 2: 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%

Both processors are implemented in Verilog RTL and synthesized with the TSMC 90nm low-power technology. All four kernels listed in Table 1 are tested by running gate-level simulation. Power consumptions are accurately estimated with the back-end information and real toggle rate generated by running each kernel with a test image of 2000 pixels.

The detailed power breakdown of the reference RISC for the YUV2RGB kernel is given in Fig. 10, which shows that the RF consumes about 16% of the total core power. Noted that the power consumption of the memory part is not counted as it highly depends on the memory size. Fig. 11 shows the power breakdown of both the reference RISC and MOVE-Pro when running the different kernel. 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 3. For all the results, we consider the logic part of the core only. 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 also substantial in all kernels except the histogram. The main reason for this is that in histogram, the computation is relatively simple and the register file consumes only a smaller portion of the total power compared to other kernels.

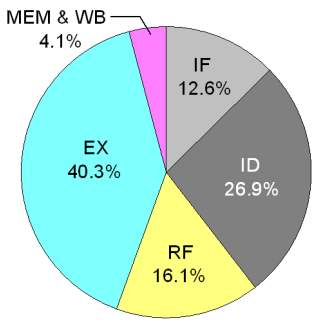

Figure 10: Power breakdown of the reference RISC processor (YUV2RGB, 4.64mW)

Table 3: Power comparison bteween 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%)

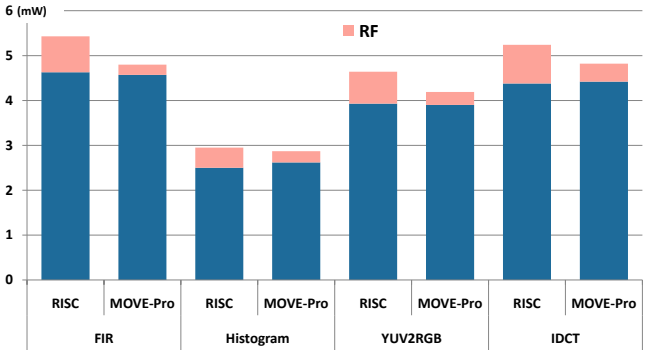

Figure 11: Power consumption result

According to Table 3, the dynamic instruction count of MOVE-Pro processor is almost the same as its RISC counterpart, which means MOVE-Pro is able to save energy without compromising the performance. With further compiler improvement, such as software pipeline and if-conversion, we expect that code density issue of the conventional TTAs can be nicely solved.

6. RELATED WORK

In microprocessors, the register file is one of the central components, which can account for considerable amount of the total energy consumption. A detailed analysis of RF power consumption is given by Zyuban and Kogge in [18].

In [15], Rixner et al. analyze the performance, area and power trade-offs in different register file designs for media processors. The idea of using variable liveness information to reduce register accesses in conventional processor architecture has bee exploited in a few previous studies [1, 9, 13, 17].

The TTA is proposed by Corporaal [2]. One of TTA's main advantages is the reduced register pressure. The MOVE project is the first implementation of TTA [12]. The TTA-based codesign environment (TCE) is the latest implementation [14]. The MaxQ micro-controller designed by the Maxim is a commercial processor which exploits the concept of TTA [10]. As an important part of the TTA, the compiler design is also discussed in several previous researches. Hoogerbrugge [7] and Janssen [8] analyze different aspects of compiler back-end design for TTA. Guzma et al. discuss the performance impact of software bypassing in TTA [4], and the power consumption implication is in [5]. In this paper we exploit the potential of building a TTA-based low power processor. And we present an energy-aware compiler back-end design which can cope with the new features in the proposed architecture.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we show that in typical kernels in streaming applications, it is possible to eliminate most of the register file accesses. And in MOVE-Pro, a TTA-based processor, such possibilities can be fully exploited. In this paper we present the design of an energy efficient compiler back-end for MOVE-Pro. A list scheduler with simple heuristic is used to perform basic block level scheduling. In the experiment, the RF access saving is significant: up to over 80%. The reduction of RF accesses also result in up to over 11% reduction of the total core power. As the dynamic cycle count remains the same, energy is saved without compromising the performance.

In this work we compare a MOVE-Pro instance with a single-issue RISC processor. It is natural to extend this to a more parallel architecture such as a VLIW processor in future work. And although in the experiment the basic block scheduler is very efficient, in control-oriented codes, the limited scheduling scope makes it difficult to achieve good result. 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.

8. REFERENCES

[1] J. Balfour, R. Halting, and W. Dally. 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] 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 *Proceedings of 20th International Conference on Field Programmable Logic and Applications*, 2010.

[4] V. Guzma, P. Jääskeläinen, P. Kellomäki, and J. Takala. Impact of software bypassing on instruction level parallelism and register file traffic. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, volume 5114 of *Lecture Notes in Computer Science*, pages 23–32. 2008.

[5] V. Guzma, T. Pitkänen, P. Kellomäki, and J. Takala. Reducing processor energy consumption by compiler optimization. In *IEEE Workshop on Signal Processing Systems, 2009. SiPS 2009*, pages 063 –068, 2009.

[6] Y. He, Y. Pu, Z. Ye, S. Londono, R. Kleihorst, A. Abbo, and H. Corporaal. Xetal-Pro: An Ultra-Low Energy and High Throughput SIMD Processor. In *Proceedings of the 47th Annual Design Automation Conference*. ACM, 2010.

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

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

[9] 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. IEEE Computer Society Press, 1995.

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

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

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

[13] N. Higaki 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.

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

[15] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, and J. Owens. Register organization for media processing. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, pages 375–386, 2000.

[16] J.-W. van de Waerdt, S. Vassiliadis, S. Das, S. Mirolo, C. Yen, B. Zhong, C. Basto, J.-P. van Itegem, D. Amirtharaj, K. Kalra, P. Rodriguez, and H. van Antwerpen. The TM3270 media-processor. In *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 12 pp.–342, 2005.

[17] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. AnySP: anytime anywhere anyway signal processing. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 128–139, 2009.

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