Compiler Supports and Optimizations for PAC VLIW DSP Processors *

Yung-Chia Lin Chung-Lin Tang Chung-Ju Wu Ming-Yu Hung
Yi-Ping You Ya-Chiao Moo Sheng-Yuan Chen Jenq-Kuen Lee

Department of Computer Science
National Tsing-Hua University
Hsinchu 300, Taiwan

Abstract. The Parallel Architecture Core (PAC) is a new VLIW DSP architecture, featuring a two cluster design, and partitioned, distributed register files with restricted access ports. Such an irregular processor poses many challenges in the construction of its compiler. This paper presents our work in providing the compilation support for PAC, based on the Open Research Compiler (ORC). We describe the design of the PAC processor and code generation methods used in coping with its features. Evaluation results of the compiler/architecture are then given, demonstrating the effectiveness of our presented methods.

1 Introduction

Current embedded applications and mobile systems are moving towards the conflicting requirements of high-performance and low-power consumption. This has influenced embedded processor design to evolve into a style of large computation resources combined with restricted and/or specialized datapaths and register storage. These architectures rely on the compiler to do much work, statically orchestrating the generated code to work around the restrictions of the hardware.

In this paper we describe our work in supporting such an embedded DSP processor. The Parallel Architecture Core (PAC) [3–5] is a new VLIW DSP processor designed by the SoC Technology Center, Industrial Technology Research Institute, Taiwan. It is architecturally designed to meet high-performance and low power requirements of embedded multimedia in portable devices.

The remainder of this paper is organized as follows. Section 2 describes the PAC architecture and its related compilation issues. The development of the PAC compiler based on ORC [1] and its code generation are presented in Section 3. Experimental benchmark results of the compiler/architecture in its current state are given in Section 4. Our current experiments show that our scheduling/allocation methods for PAC's datapath and register file design are quite effective. Other

* The work was supported in part by NSC under grant no. 93-2220-E-007-019 and 93-2220-E-007-020, by Ministry of Economic Affairs under grant no. 93-EC-17-A-03-S1-0002 and 94-EC-17-A-01-S1-034, and by MOE research excellent project under grant no. 94-2752-E-007-004-PAE in Taiwan

transformations such as peephole optimizations, and the higher-level loop-nest optimizer (LNO), are also put into comparison. Section 5 concludes this paper.

2 The Parallel Architecture Core

The Parallel Architecture Core (PAC) is a fixed-point, 5-way issue VLIW DSP architecture. PAC contains two Arithmetic Logic Units (ALU), two Load/Store Units (LSU), and a single Scalar Unit. The ALU and LSU units are organized into two clusters, each containing a pair of both functional unit (FU) types. The Scalar Unit executes branch operations, and is also capable of most common load/store and arithmetic operations. The architecture is illustrated in Fig. 1.

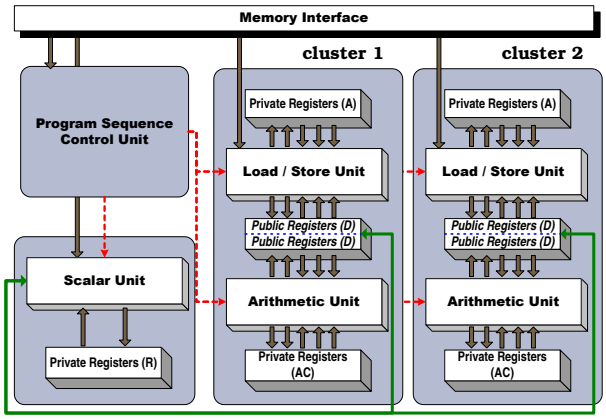

Fig. 1. The PAC DSP Architecture

The registers of PAC are highly partitioned and distributed. The rationale of this design is to avoid complete connections to all functional units, thus lowering register file port counts. This is to avoid the large area, slow access speed, and high power consumption of a fully connected unified register file, although at the expense of an irregular architecture.

Examining PAC's registers, the A and AC register files are private registers, directly attached to and only accessible by each LSU and ALU respectively. The R-register file is private to the Scalar Unit. There are four D-register files, partitioned among the two clusters. Each of the D-register files have only 3 read ports and 2 write ports (3R/2W). Among them, 1R/1W are dedicated to the Scalar Unit, leaving only 2R/1W for the cluster FUs to use. The remaining set of 2R/1W ports are not enough to connect to both cluster FUs simultaneously. Instead, they are *switched* between the LSU/ALU: during each cycle, the access ports of each of the two D-register files (in a single cluster) may be connected to the LSU **or** ALU, but **not both**. This means that access of the two D-register files are mutually-exclusive for each FU, and each LSU/ALU can access only one of them during each cycle. The designers of PAC called this a '*ping-pong register file*' design, presumably due to the back-and-forth style of register file access.

These port access restrictions makes operand access in the PAC architecture *resource dependent*. This makes instruction scheduling and register assignment inherently inseparable. For example, the short code sequence:

```
mov TN1, 1
mov TN2, 2
```

Two move operations that assign a constant to a virtual register: they can only be scheduled in parallel if TN1 and TN2 are assigned registers from different D-register files; if both are assigned to the same D-register file, they can only be scheduled and issued sequentially. We see parallelism restricted by the ping-pong register in this example.

Adding another operation, the example becomes more complicated. The third add instruction accesses TN1 and TN2, thus they must be simultaneously accessible by the add operation to be valid.

```
mov TN1, 1
mov TN2, 2
add TN3, TN1, TN2
```

As mentioned above, an operation cannot access both D-register files in the same cycle. As shown in Fig. 2, a copy operation must be inserted to transfer TN1 or TN2 to another register file, to generate valid code. This example used an additional copy with no gain in performance

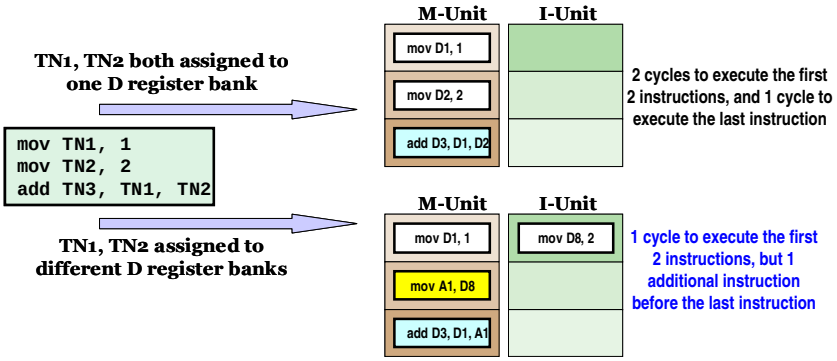

Fig. 2. An illustration of interference caused by ping-pong register files

Figure 3 shows example of how cross cluster communication works: the current version of PAC requires explicit cross-cluster copy operations to transfer data between clusters. Cross cluster data transfer is a high latency operation requiring many cycles to complete, and can hinder performance if not used with respect to the overall schedule.

3 Compiling for the PAC Architecture

The design of PAC as an instruction set has made it a hard fit into contemporary compiler target machine models, affecting our development of its compiler. The

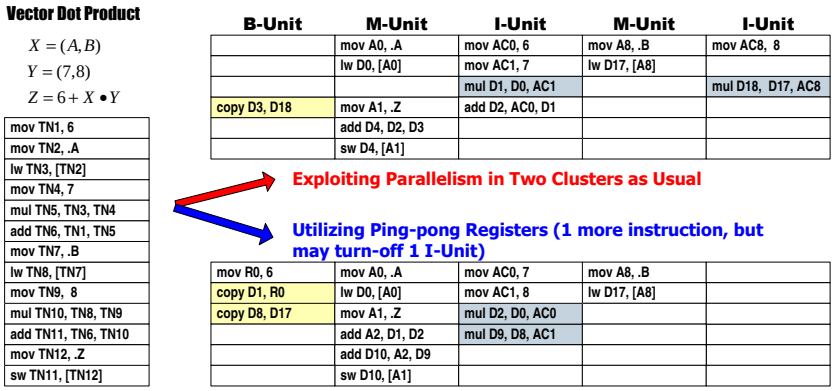

Fig. 3. An example of scheduling operations across clusters

porting of the main ORC infrastructure involved filling in many target machine information fields, and writing code expansion procedures for WHIRL-to-CG translation. Much of the machine level transformations, however, required rolling out our own methods. This section will describe some of our work in generating code for PAC.

3.1 Allocation of Operand Storage

The clustered design and ping-pong register files in PAC, implies the concept of scheduling has to be extended from functional units, to resources guarding access of operand storage. The access port restriction of the ping-pong registers is an example.

Our current method, is a scheduling pass that simultaneously optimizes the instruction sequencing and resource assignment by *simulated annealing* (SA). The design originates from Leupers' work [9] and our initial implementation [6], using a combined instruction scheduling/register file assignment algorithm to iteratively obtain better results.

Leupers' algorithm operates by first generating a random bi-partitioning of the operations; a list scheduler (LS) then schedules the partitioned instructions, inserting/managing cross cluster communications along the way. The LS then returns the obtained schedule length as the 'energy' value used in a simulated annealing optimization process, representing an evaluation of the current partitioning state. This process is repeatedly invoked, and in each iteration, depending on whether improvement is gained or not, the random change may be retained or discarded. Iteration continues until the energy/evaluation falls under some threshold, where we are confident that the obtained optimization state is of sufficient quality.

To adapt this SA method for PAC we need a change in the formulation of optimized state: instead of a partitioning of operations, we optimize a *register file assignment* of the virtual registers. Figure 4 shows the high-level simulated annealing control. It iteratively runs the scheduler, which does fine-grain sequencing of operations, and returns an evaluation of the current optimization state. Currently, the evaluation consists of just the length of the schedule, as in

Combined Instruction Scheduling and Register File Assignment by Simulated Annealing

Input: n operations to be scheduled
Output: Schedule of the n instructions and a register file assignment (RFA) map:
VR: set of all virtual registers, RF: set of register files
$RFA\_map = \{(v_1, f_1), (v_2, f_2), ...\}$ $v_i \in VR, f_i \in RF$

1. Choose a schematic register file placement(e.g. 1 cluster, 2 clusters, ...).
2. Make initial register file assignments: randomly assign each virtual register to any of the wanted register files, and record in RFA_map.
3. Given RFA_map, run **PAC_Scheduler**, and set $sched\_len$ to the computed total schedule length in cycles.
4. Set initial values for:
   $threshold$: threshold value for the simulated annealing process.
   $energy$: initial energy, larger than $threshold$.
   p_test: a probability test value p_test $(0 < p\_test < 1)$.
5. Repeat the following steps while $energy > threshold$:
   5a. Make change in RFA_map:
   randomly choose a virtual register, and assign it to a different register file.
   optionally change the schematic register file placement.
   5b. With the new RFA assignment change, run **PAC_Scheduler** again, and set new_sched_len to the new count of total schedule length.
   5c. Adjust $energy$, $sched\_len$, and RFA_map by the following rules:
   If $new\_sched\_len < sched\_len$ then
   decrease $energy$, set $sched\_len$ to new_sched_len,
   and keep the new RFA changes made in step 5a.
   If $new\_sched\_len \geq sched\_len$, get random number $0 \leq R \leq 1$:
   If $R > p\_test$ then
   decrease $energy$, set $sched\_len$ to new_sched_len,
   and keep the changes made in step 5a.
   If $R \leq p\_test$ then
   increase $energy$ and revert changes made in step 5a.
6. Optionally choose another schematic register file placement, and repeat steps 2–5 to select the better results.
7. Retain the final schedule and RFA_map as the output results.

Fig. 4. The high-level simulated annealing algorithm

The *PAC_Scheduler* Algorithm

Input: $ReadyList$ of operations to be scheduled
RFA_map, in the form of a function $RegisterFile : VR \rightarrow RF$
Output: Schedule of the n instructions, and the schedule length

```
While ReadyList is not empty:
  Select operation Op from ReadyList
  Find earliest cycle Cycle we can schedule Op
  While Op is not scheduled:
    Examine available resources in Cycle, and:
    For each register operand oi of Op:
      If no resources available to access RegisterFile(oi):
        Enumerate possible copy sequences to transfer oi to an accessible register file
        For each copy sequence cpseq
          If cpseq is schedulable in prior cycles:
            Feasible(cpseq) = true
    If for each register operand oi of Op we have resources to access RegisterFile(oi),
    or we have some c such that Feasible(c):
      Schedule Op into Cycle, advance ReadyList
      Break from inner loop
    else
      Increment Cycle
Return length of schedule
```

Fig. 5. The scheduler/evaluation algorithm

Leupers' algorithm. We are currently looking at incorporating other statistics (e.g. register pressure, number of inserted copies) as part of the evaluation. The assignment of register files will improve progressively throughout the SA iterations, with respect to the evaluation function. A final register allocator is then

run to allocate and assign hardware registers, which is guided by the result register file assignments (*RFA_map*). Fig. 5 illustrates the details of the scheduler algorithm.

3.2 Peephole Optimizations for PAC

The Extended Block Optimizer (EBO) is a collection of peephole optimization passes performed at the scope of extended basic blocks on the low-level machine instruction IR in ORC. Our work on EBO not only refines existing optimizations, but also intend to utilize PAC's DSP-specific features. Table 1 shows a comparison of EBO in the original ORC for Itanium, and our PAC port of ORC.

Table 1. Comparison of EBO collections

EBO optimization	ORC for IA64	PAC DSP compiler
Forward Propagation	×	×
Common Subexpression Elimination (CSE)	×	×
Constant Folding	×	×
Dead Code Elimination (DCE)	×	×
Resolve Conditional Branch	×	-
Condition Redundant	×	-
Merge Memory Offset	-	×
Compound Operation Conversion	×	×
Subword Calculation	-	×
Dual Operation	-	×

Some of the optimizations, specifically constant folding and DCE, are less affected by the restrictions of PAC. But others, when applied after register allocation, such as forward propagation and CSE which modify the operation referenced operands (TNs) may produce invalid code due to register access restrictions.

```
Original code sequence
    1: TN186(d1) <- copy_m TN185(a1)
    2: TN187(d9) <- copy_m TN186(d1)
    3: TN188(d8) <- addi_i TN187(d9) 0x4

After propagation
    TN188(d8) <- addi_i TN185(a1) 0x4 ;
```

An example above illustrates a typical copy propagation: The value from *TN185(a1)* is transferred to *TN187(d9)* in lines 1 and 2, so *TN185(a1)* should be able to propagate to the reference of *TN187(d9)* in line 3, leaving lines 1 and 2 as redundant code that can be eliminated. However, it is invalid under PAC's architectural restrictions; the operation code *addi_i* is an I-unit (ALU) instruction whose operand must be in the AC- or D-register files. *a1* is not accessible by *addi_i* under any conditions, and thus not valid code. In order to prevent such errors, we should utilize the register file assignment map, mentioned in the last section, to guard against invalid code transformations. We are also working on optimizations that makes use of PAC's more sophisticated operations: multiply-accumulate, double-word load/store, vector operations, post-increment/decrement addressing, etc.

4 Experimental Results

Our experiments were done with the DSPstone benchmark suite [10]. Since the PAC DSP compiler is still in progress, we have only evaluated some stable optimization combinations to obtain an early assessment of the effectiveness of our work. All benchmark programs are compiled with three types of option sets: the traditional register allocation (TRA), TRA with LNO and EBO (LNO+EBO), and register allocation with simulated annealing scheduling (SARA). TRA refers to a modification of the original ORC register allocator that assumes full general purpose registers, with a post-process pass that forces copy operations to generate valid code. It is treated as the base reference in our comparison.

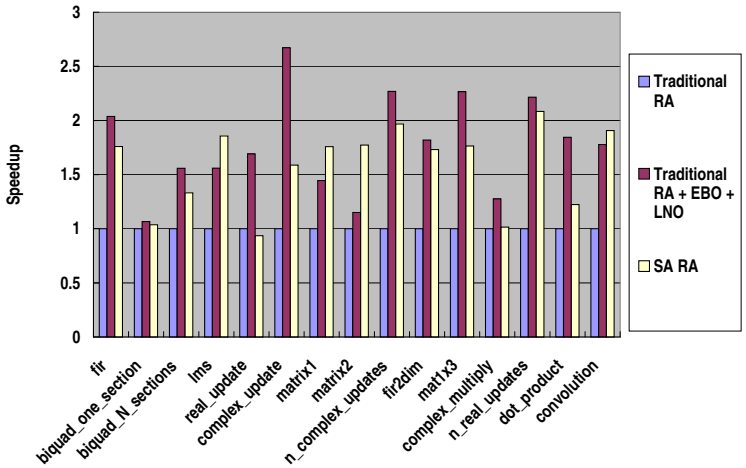

Fig. 6. Comparison of speedups under various optimization options

Fig. 6 shows and compares the speedup of the benchmarks with option sets LNO+EBO and SARA, using TRA as the baseline. As shown in Fig. 6, the performance gains vary widely across different benchmarks with average speedup of 1.78 for LNO+EBO and 1.58 for SARA.

Although the LNO+EBO+SARA combination is not yet stable enough for full testing, the current results show that our methods of scheduling/allocation, code optimizations, and integration of the LNO phase could yield substantial improvement in the generated code. Also, the simulated annealing process gives a potentially exhaustive exploration on how the register usage affects PAC, investigating architectural parameters. Currently, there is a restriction that data dependencies across different functional units takes 3 cycles, due to lack of bypass paths between FUs. This causes significant restrictions in utilizing all functional units, causing some of the benchmarks, such as *biquad_one_section*, less affected by our optimizations. We should note that these test results have all been used as feedback for the DSP design team, and we expect an enhanced next generation PAC to improve on these deficiencies.

5 Conclusion

We have presented our compiler development for PAC, a new VLIW DSP processor which features a clustered design and complex, partitioned register files. We have described methods of code generation for PAC and demonstrated their viability through several preliminary experiments which were done with the PAC DSP prototype. We also note that many deficiencies of the first generation PAC architecture were revealed by our evaluation and we are currently referring to the experiences and reforming the development of compilers for the next generation of PAC DSP architecture, which will further extend our current work.

References

1. Roy Ju, Sun Chan, and Chengyong Wu: Open Research Compiler for the Itanium Family. Tutorial at the 34th Annual Int'l Symposium on Microarchitecture, Dec. 2001
2. Tay-Jyi Lin, Chen-Chia Lee, Chih-Wei Liu, and Chein-Wei Jen: A Novel Register Organization for VLIW Digital Signal Processors. Proceedings of 2005 IEEE International Symposium on VLSI Design, Automation, and Test, pages 335–338, 2005
3. David Chang and Max Baron: Taiwan's Roadmap to Leadership in Design. Microprocessor Report, In-Stat/MDR, Dec. 2004. http://www.mdronline.com/mpr/archive/mpr_2004.html
4. Tay-Jyi Lin, Chin-Chi Chang. Chen-Chia Lee, and Chein-Wei Jen: An Efficient VLIW DSP Architecture for Baseband Processing. Proceedings of the 21th International Conference on Computer Design, 2003
5. Tay-Jyi Lin, Chie-Min Chao, Chia-Hsien Liu, Pi-Chen Hsiao, Shin-Kai Chen, Li-Chun Lin, Chih-Wei Liu, Chein-Wei Jen: Computer architecture: A unified processor architecture for RISC & VLIW DSP. Proceedings of the 15th ACM Great Lakes symposium on VLSI, April 2005
6. Cheng-Wei Chen, Chung-Lin Tang, Yung-Chia Lin, and Jenq-Kuen Lee: ORC2DSP: Compiler Infrastructure Supports for VLIW DSP Processors. Proceedings of 2005 IEEE International Symposium on VLSI Design, Automation, and Test, pages 224-227, 2005
7. OMAP5910 Dual Core Processor - Technical Reference Manual, Texas Instruments, Jan, 2003.
8. S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U. J. Kapasi, and J. D. Owens: Register organization for media processing. International Symposium on High Performance Computer Architecture (HPCA), pp.375-386, 2000
9. R. Leupers: Instruction scheduling for clustered VLIW DSPs. In Proc. Int'l Conference on Parallel Architecture and Compilation Techniques, pages 291–300, Oct. 2000
10. V. Zivojnovic, J. Martinez, C. Schläger and H. Meyr: DSPstone: A DSP-Oriented Benchmarking Methodology. Proc. of ICSPAT, Dallas, 1994