Enabling Compiler Flow for Embedded VLIW DSP Processors with Distributed Register Files *

Chung-Kai Chen, Ling-Hua Tseng, Shih-Chang Chen, Young-Jia Lin,
Yi-Ping You, Chia-Han Lu, Jenq-Kuen Lee

Department of Computer Science
National Tsing Hua University
Hsinchu 30013, Taiwan

Abstract

High-performance and low-power VLIW DSP processors are increasingly deployed on embedded devices to process video and multimedia applications. For reducing power and cost in designs of VLIW DSP processors, distributed register files and multi-bank register architectures are being adopted to eliminate the amount of read/write ports in register files. This presents new challenges for devising compiler optimization schemes for such architectures. In this paper, we address the compiler optimization issues for PAC architecture, which is a 5-way issue DSP processor with distributed register files. We present an integrated flow to address several phases of compiler optimizations in interacting with distributed register files and multi-bank register files in the layer of instruction scheduling, software pipelining, and data flow optimizations. Our experiments on a novel 32-bit embedded VLIW DSP (known as the PAC DSP core) exhibit the state of the art performance for embedded VLIW DSP processors with distributed register files by incorporating our proposed schemes in compilers.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors—Compilers; D.3.4 [*Programming Languages*]: Processors—Optimization

General Terms Languages

Keywords Embedded VLIW DSP Compilers, Distributed Register Files, Software Pipelining

1. Introduction

The PAC DSP processor [7], employs a five-way-issue VLIW architecture with distributed clustered register file. In addition, there are multi-bank register files in each cluster. It also incorporates a banking technique called "ping-pong" register file structure, which is divided into two banks and in which banks can only be restrictedly accessible in a mutual-exclusive way. PAC is targeted to meet the requirements for multimedia and communication services on the next-generation mobile devices with reasonable hardware cost and flexibility, while the applications on these devices, such as H.264 decoding and encoding for video streaming, demand high computational power and low power consumption simultaneously.

The appearances of multi-banks of register files, distributed register clusters, and ping-pong architectures on embedded VLIW DSP processors present a great challenge for compilers to generate efficient codes for multimedia applications. In the literature, current research results in compiler optimizations for such problems have been limited to address issues for cluster-based architectures. It includes the work on partitioning register files to work with instruction scheduling [1], loop partitions for clustered register files [2], and global register allocations for cluster register files [3]. The work in [4] begins to address this complex optimization issue for embedded DSP processors, but only in the layer of copy propagation optimizations.

In our work, we address the complex optimization issues in meeting the challenges of features with multi-bank register files, distributed register clusters, and ping-pong architectures. We present a compiler flow to address the issues based on the well-known open-source compiler infrastructure, Open Research Compiler (ORC), to utilize the state-of-the-art compiler technologies in adressing the limited connectivity issues in register files. We address issues both for instruction scheduling and software pipelining. A hybrid optimization scheme is proposed in our support for instruction scheduling in interacting with register allocations. We first employ a graph-partitioning-like method with several assignment policies to better utilize the distributed and ping-pong register file architectures. The result is then used as a pre-conditioner for the second phase simulated annealing (SA) based approach for tuning the performance. Since the SA requires to be processed within a limited iterations (controlled by $threshold$), an appropriate initial point usually ensures a good result. In addition, we also present a modulo scheduling scheme to simultaneously address the issues with distributed clustered register file, multi-bank register files, and ping-pong register files for VLIW DSP processors. The scheme is based on the concept of multiple-phase register allocations, ping-pong constraint-aware scheduling, and register bank assignments based on wild-card schemes. The scheme looks promising in our initial experiments to address this complex optimization issue for software pipelining. Our experiments are done with PAC ISS and with DSP stone benchmarks. The experimental results shows our scheme can deliver the state of the art performances for embedded VLIW DSP processors with irregular register files and limited connectivities.

* This work was supported by NSC under grant No. NSC 95-2220-E-007-001 and NSC 95-2220-E-007-002, and MOEA research project under grant No. 95-EC-17-A-01-S1-034 and 96-EC-17-A-01-S1-034.

LCTES'07 June 13–16, 2007, San Diego, California, USA.
ACM 978-1-59593-632-5/07/0006.

2. PAC Architecture

The PAC DSP employs a five-way-issue VLIW architecture with the heterogeneous design that equips one singular scalar unit (B-unit) for light-weight arithmetic, address calculation, and program flow control, plus two data stream processing clusters in which each one contains a pair of load/store unit (M-unit) and ALU/MAC unit (I-unit) with powerful SIMD capabilities; two types of register files are disposed for each unit in the clusters, providing different accessing manners and constraints; the B-unit has its own accessible register file deployed. The A, AC, and R register files are local registers that are directly attached to and only accessible by the M-, I-, and B-units, respectively; the D register files are shared within a cluster. The distributed register files and the clustered organizations reduce the wire connections between functional units and registers in the hardware design, and thereby decrease the chip area and power consumption. Another major feature adopted by the register file architectures in the PAC DSP to further reduce the read/write ports needed is that it incorporates a banking technique called as the "ping-pong" register file structure, which is divided into two banks and in which banks can only be restrictedly accessible in a mutual-exclusive way, as the M and I-units in a cluster can only access the different banks in the same time. Additionally, a unique design used in the PAC DSP, to allow the intercluster communication through the internal data-routing paths in the memory interface unit which connects with all B- and M-units, simplifies the implementation of intercluster communication compared to other existing schemes [5], providing more reduction of area size and access time [6]. With the featured register file organizations and heterogeneous architectures, not only does the clustered design make register access across clusters an additional issue, but the switched access nature of the register file demands the new exploration into optimizing code generations. The main focus will be addressing the instruction scheduling in interacting with register allocations and software pipelining issues to address the issues with distributed clustered register file, multi-bank register files, and ping-pong register files for VLIW DSP processors.

3. Register Allocation & Instruction Scheduling

In our work, we try to enable ORC compiler flow for PAC embedded VLIW DSP processors. After going through the lowering and WHIRL-level optimization phases, the back-end driver calls the code generator to translate the WHIRL IR into CGIR (Code Generation Intermediate Representation), with our efforts to support the PAC target processor. We propose a heuristic algorithm, called *ping-pong aware local favorable* (PALF) register allocation, to improve the register allocation by efficiently utilizing the irregular register file architectures in the PAC DSP. The algorithm appropriately considers various characteristics in accessing different register files, and attempts to minimize the penalty associated with the interference between register allocation and instruction scheduling, while retaining desirable parallelism despite ping-pong register constraints and intercluster overheads. Gven a dependency DAG (directed acyclic graph) that describes the compilation regions, PALF heuristically determines the appropriate register file/bank assignment and employs state-of-the-art graph-coloring register allocation for each assigned register file/bank in PAC architectures. An overall flowchart of the proposed register allocation algorithm is shown in Fig. 1.

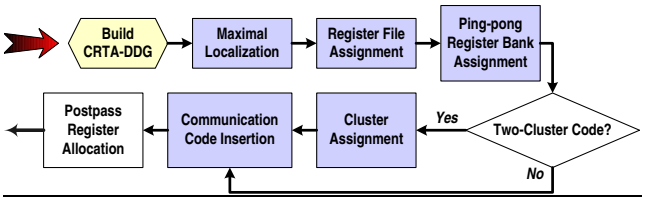

Figure 1. The flowchart of the PALF register allocation scheme

SA-based Refinements

Once the initial point is obtained, we then proceed with a simulated annealing scheme for further tunings. The design extends that of Leupers [1] and Lee [8] using a combined instruction scheduling/cluster assignment algorithm to iteratively approach the near-optimal result. In brief, the algorithm operates by first generating a random cluster partitioning of instructions, and a modified list scheduler (LS) then schedules the partitioned instructions whilst inserting/managing cross-cluster communications. The subsequent iterations involve random changes to the partitioning state and re-running of the LS. The LS returns the obtained schedule length of the instructions as the "energy" value used in a usual SA optimization process, representing an evaluation of the current partitioning state. Depending on whether a random change results in improvement or deterioration, it will be retained or discarded. This process is iterated until the energy/evaluation falls to below a threshold at which we are confident that the obtained optimization state is of sufficient quality.

4. Software Pipelining

4.1 Ping-pong Constraint aware Modulo Scheduling

In addition to the resource constraint and the recurrence constraint, the ping-pong constraint is also checked when trying to schedule instructions into execution slots. In order to examine if the ping-pong constraint is followed, the compiler maintains a mark for each cycle to keep track of ping-pong agreements. The extended resource reservation table is now with ping-pong agreement (`PPA`) fields. The `PPA` fields determine the accessibility of ping-pong registers at each cycle. The `PPA` field for each cycle are empty initially until instructions that use ping-pong registers are scheduled into that cycle. Once the `PPA` field of one cycle is set, the following instructions willing to be scheduled into that cycle must comply with its ping-pong agreement.

4.2 Multi-phase Register Allocation

In the traditional software pipelining technology dealing with uniform register files, register allocation can be done independently after the instruction scheduling. But for the case with distributed register files, there are usually scheduling constraints depending on the registers used. The ping-pong constraint in PAC is one example. It is obvious that we can not hold back the register allocation phase till the scheduling is done.

Fortunately, these kinds of constraints are usually related to which register bank is used rather than which register is used. The solution we proposed here is to split the register allocation into multiple phases. We assign virtual registers to appropriate register banks before the scheduling. Once the scheduling is completed, they will be further allocated precisely for each bank.

The early assignment of register banks might lead to unbalanced bank usage due to the lack of lifetime information deriving from the scheduling. We have developed a heuristic strategy to alleviate this side effect.

Figure 2 shows the proposed software pipelining flow with splitted register allocation phases, which includes the following items.

1. **Restricted M/I Selection**

   We use an M/I selection algorithm similar with the one used earlier in PALF (Section 3) to lower the necessity of using D

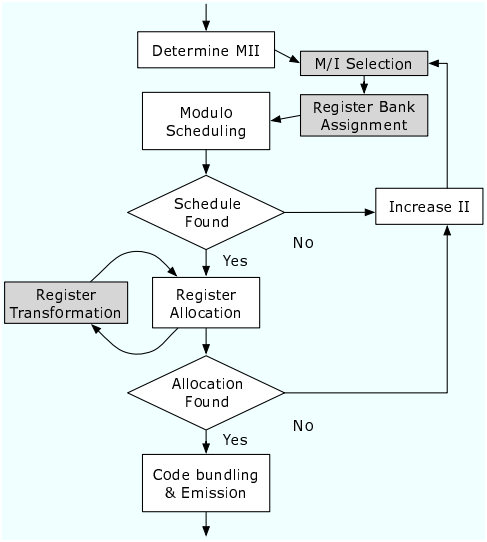

Figure 2. The modulo scheduling flow with multi-phase register allocation.

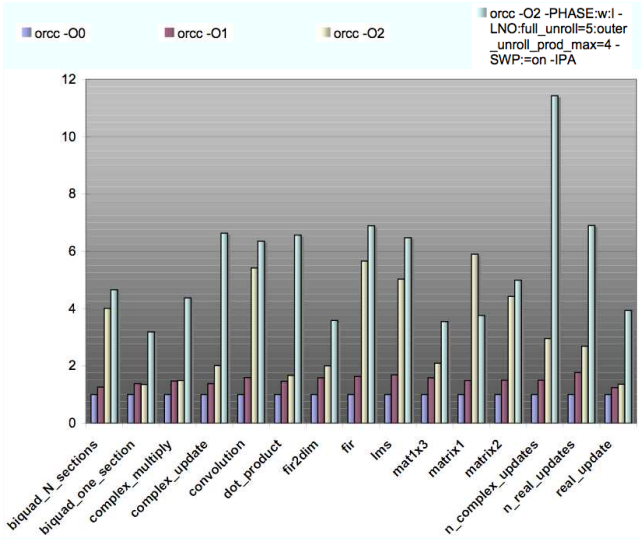

Figure 3. DSPstone benchmark performance with four compiler options.

registers. PALF scheme prefers the local register bank and then does graph partitioning scheme in case the local register first rule can't be applied. The only difference is that here we need to put an additional constraint on the number of resulting M or I instructions. It is important not to overuse certain type of instructions so that the ResMII(Resource Minimum Initiation Interval) exceeds the current II, or it will be impossible to find a feasible scheduling.

2. **Register Bank Assignment**

   After the M/I selection, we apply a simple strategy to arrange register banks to use. The D register banks are assigned only when it is necessary. Otherwise A/AC registers are preferred. Since the codes are not in single assignment form, it is possible that copy instructions are inserted to move data between D and local registers.

3. **Register Allocation with Transparent Register Transformation**

   As mentioned before, the early bank assignment may cause unbalanced allocation. This problem can be eased by the following step. We can try to find a set of "transparent registers" which can be promoted from local register to D register (ping-pong register) bank. These transformations turn the use of local registers to global registers. The "transparent" means they won't cause ping-pong constraint violation for the scheduling currently found.

5. Experiments & Summary

Preliminary experiments were performed using DSPstone benchmarks [9]. Figure 3 shows the performance of our compiler implementation on DSPstone benchmarks. The X-axis lists the name of each tested benchmark program. The Y-axis is given with the speedup of three compiler settings compared with -O0. The -O0 version is the base version for comparison, and it's the version with our retargetting effort from ORC to PAC without considering the effect of distributed register file. The instruction scheduling technique proposed in Section 3 is enabled in -O1. The revised software pipelining technique in Section 4 is enable in -O2. We can see that for -O1 we get the speedup from 13% to 45% over the base -O0 version. For -O2 we get the speedup from 16% to 500% over the base -O0 version. We can see significant speedups from `-O1` to `-O2`. In addition, the fourth set of compiler options consists of explicit parameters controlling the unroll of loop bodies, IPA (inter-procedural analysis) and LNO (loop optimizations). It shows that our optimization scheme can work with IPA and LNO to further optimize performance from 200% to 1000% over the base -O0 version. In summary, our experiments on a novel 32-bit embedded VLIW DSP (known as the PAC DSP core) exhibit the state of the art performance for embedded VLIW DSP processors with distributed register files by incorporating our proposed schemes in compilers.

References

[1] R. Leupers, Instruction scheduling for clustered VLIW DSPs, In *PACT*, pages 291-300, Oct. 2000.

[2] Y. Qian and S. Carr and P. Sweany, Optimizing loop performance for clustered VLIW architectures, In *The 2002 International Conference on Parallel Architectures and Compilation Techniques*, pages 271- 280, Sept. 2002.

[3] J. Hiser, S. Carr, and P. Sweany, Global Register Partitioning, In *Proc. Ninth Intl Conf. Parallel Architectures and Compilation Techniques*, pp. 13-23, Oct. 2000.

[4] Chung-Ju Wu, Sheng-Yuan Chen, and Jenq-Kuen Lee, Copy Propagation Optimizations for VLIW DSP Processors with Distributed Register Files, In *LCPC*, 2006.

[5] A. Terechko, E. L. Thenaff, M. Garg, Eijndhoven, and H. Corporaal. Inter-cluster communication models for clustered VLIW processors. Procs. HPCA, 2003; 354–364.

[6] T.-J. Lin, P.-C. Hsiao, C.-W. Liu, and C.-W. Jen. Area-efficient register organization for fully-synthesizable VLIW DSP cores. International Journal of Electrical Engineering, vol. 13, May 2006.

[7] 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

[8] Yung-Chia Lin, Chung-Lin Tang, Chung-Ju Wu, Ming-Yu Hung, Yi-Ping You, Ya-Chiao Moo, Sheng-Yuan Chen and Jenq Kuen Lee, Compiler Supports and Optimizations for PAC VLIW DSP Processors, In *LCPC*, 2005.

[9] V. Zivojnovic, J. Martinez, C. Schläger and H. Meyr. DSPstone: A DSP-Oriented Benchmarking Methodology. *Proc. of ICSPAT*, Dallas, 1994.