A Retargetable Compiler of VLIW ASIP for Media Signal Processing

Zhou Zhixiong, Yang Xu, He Hu and Sun Yihe
{zhouzx02, mujueyun99}@mails.tsinghua.edu.cn, {hehu, sunyh}@mail.tsinghua.edu.cn

Abstract*

In the last decade extensive researches have been carried out in ASIP (Application Specific Instruction Processor) design field. One of the key steps in ASIP design is code generation by a retargetable compiler. In this paper we describe our experience in implementing a retargetable compiler for VLIW ASIP based on ORC (Open Research Compiler) framework. Orienting towards a new register file access architecture model, we narrate the process making modifications on ORC framework to get the compiler. The experimental results indicate that our method is effective to get compilers retargeting at VLIW ASIPs.

Key Words: compiler, ASIP, VLIW

1. Introduction

In embedded system designs, processors are expected to achieve high performance, small chip area, low power consumption and short time-to-market. ASIC (Application Specific Integrated Circuit) is specifically designed for one behavior, where the performance and area can be easily optimized, but it is difficult to make any changes at a later stage, and the process of designing is tedious and lengthy. While CPU can offer high programmability and short design time, but may not satisfy the power and area constraints. ASIP (Application Specific Instruction Processor), which is designed for a set of applications and can overcome both the shortcomings of ASIC and CPU in its application specific field, has been extensively researched in the last decade.

ASIPs are designed for a set of applications, and in general custom-designed specific function units are implemented to improve the performance, which are controlled by specific instructions. And the hardware parameters (such as register file size, cache size, etc.) are often adjusted to meet the demand of applications. According to Manoj et al [1], a retargetable compiler or compiler generator is one of the key steps in ASIP design. Research into retargetable compilers has been carried out for some ASIP architectures, such as IMPACT [2] with code optimization components especially developed for multiple-instruction-issue processors, DRESC [3] compiler for coarse-grained reconfigurable architectures focusing on exploiting loop-level parallelism on a wide range of loops and Trimaran compiler [4] for HPL-PD architecture especially geared for instruction-level parallelism, etc.

In this paper we describe our experience in deploying ORC [5] (Open Research Compiler) framework for a new ASIP processor, which equips with new architecture features, such as global register files and double destination mode of instructions. The remainder of this paper is organized as follows. Section 2 gives an overview of ORC and presents the advantages using it to form a retargetable complier. Section 3 describes the target VLIW architecture and Section 4 proposes the method of making modification over ORC and Section 5 gives the experimental result of compiling some benchmarks. Finally Section 6 summarizes the work and mentions the future directions.

2. Overview of ORC

ORC is a retargetable C/C++ and Fortran compiler aiming at IPF architecture on Linux platform based on Pro64 open source compiler from SGI, comprised of compiler front-end, optimization, code generation, code emission part and machine description files. The intermediate representation of ORC, which is called WHIRL, serves as the common interface among all these components. In the following we will present the characters of ORC that facilitate the modification over ORC framework to get a retargetable compiler for ASIPs.

First, ORC has a perfect retargetable architecture. The processor architecture information is included in machine description files, which the compiler makes use of directly. Second, the machine description is flexible enough to form a machine model, which include instruction set, function units, register files, cache information etc. Third, ORC instruction set, which is comprised of over seven hundreds instructions and includes media ones, cover most instructions in ASIP, which facilitates code expansion into the target ASIP instruction set in compiling processes.

* This paper is supported by the National Natural Science Foundation of China (NSFC) under grant No.60236020.

Lastly, ORC incorporates many state-of-the-art technologies, such as global instruction scheduler integrated with a finite-state-automaton-based resource management, control and data speculation with recovery code generation, if-conversation, and predicate analysis, etc. Compilers base on ORC framework may utilize the technologies to improve performance and size of object code.

3. The target VLIW architecture

VLIW architecture has been extensively adopted in ASIPs for media signal processing. One feature of VLIW architecture is that many function units (FUs) and register files exist in the processors to reduce port number of register files so as to improve efficiency. But it is difficult to communicate between the function units in different clusters. Many kinds of methods have been developed to solve the problem, such as [6][7], etc. We have proposed a new architecture for media signal processing with organization of local register files and global register file [8] to facilitate communications among different clusters, as shown in Fig.1.

In this architecture there are four kinds of elementary function units. They are Arithmetic and Logic unit (AL), Arithmetic and Branch unit (AB), Multiplication unit (MP) and Load Store unit (LS). FU in clusters can also be defined by users to perform some specific function. AL unit can be used to perform most of the arithmetic and logic operation; AB unit is mainly used to execute branch operations and can also perform come arithmetic operation; MP unit is only be used to perform multiplication; and the main function of LS is to load/store data from/to memory. The FUs in Fig.1 can be either these elementary units or custom FUs for specific applications.

A global register file is used to exchange data among FUs in different clusters in this new ASIP architecture. Moreover, data produced by one cluster can be directly accessed by others clusters without costing extra cycles and operations. All the operations have two destination modes: single destination mode and double destination mode. When single destination mode is used, results will be written to the destination register specified by destination address. Whereas when double destination mode is used, results will be written not only the destination register but also the associate global register of the destination register. So, function units in a cluster share their local register file and the ones in different clusters share the global register file without extra cycles by double destination operation mode in this architecture.

The architecture model is consisted of a basic architecture and variable parameters. One AL unit, one AB unit, one MP unit and one LS unit, two local register files, a global register file and a set of basic instructions compose the basic architecture. The variable parameters include: 1) issue width of this VLIW architecture, namely the number of function units in the processor. 2) number of local register files, which is also number of clusters for every cluster is provided with a local register file. 3) number of registers in every register file. 4) function units in every cluster, which can be either the elementary FUs or

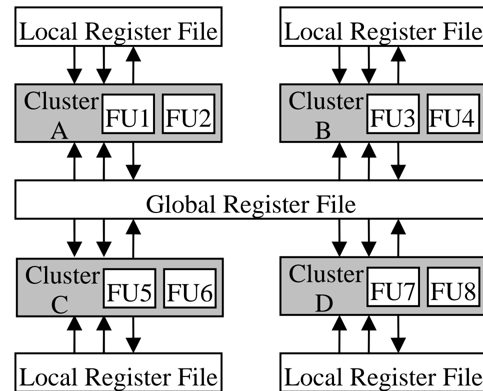

Fig.1 ASIP architecture with global register file

custom FUs. 5) expanded instruction set. The instruction set in this architecture is consist of basic instruction set which cannot be changed and expanded ones which can be adjusted. The expanded instructions are those executed in custom FUs. Moreover, the delay and latency of expanded instructions are also variable.

Targeting at the model of this VLIW architecture, we will give the process how to generate a retargetable compiler based on ORC in the following section.

4. The retargetable compiler

In ORC complier, the architecture model is defined in machine description files, so the all-important modifications are rewriting the machine description files. Firstly, all the new architecture instructions are kept back, the others are deleted and instructions in custom FUs are added. The instruction information is consisted of opcode, operand and latency information, all of which should be replaced. Secondly, type, number and characters of new register file architecture are written into the machine description files to replace the original, such as local or global register character. Thirdly, function units information is modified to adapt to the new architecture and custom function units are added. Lastly, the relation of function units and instructions, register files should be included in the machine description.

The front end of ORC compiler transforms source files into WHIRL intermediate representation, and then the back end turns WHIRL into assembly code, at last the assembler converts assembly code into object code. The front end is independent of machine, so we focus on the back end, which includes code expansion, instruction scheduling, register allocation and code emission phases.

In code expansion phase, WHIRL intermediate representation is converted into regions and basic blocks. Every basic block is a link list of operations containing the dependence relation. In our ASIP architecture, all the instructions executed in the elementary FUs are covered by Itanium instructions, so only small changes should be

made to expand WHIRL into the instructions of the elementary FUs. The small changes include instruction formats in machine description files, forbidding illegal operations generation, etc. For the instructions bundled with custom FUs, extra instruction match method must be developed to introduce instructions into the link lists of operations. And the function unit information are added in machine description files, such as what operations they can execute, which cluster they belong to, etc.

In instruction scheduling phase, we introduce preprocessing algorithm to improve ILP (Instruction Level Parallelism) and achieve high performance for the ASIP architecture. Global registers is mainly used to exchange data among different clusters, so local registers must be utilized with priority. In general register assignment won't be considered until register allocation phase, yet it isn't the case in this ASIP architecture. Because clusters and local register files are bundled, which cluster an instruction is executed in will affect the latter register allocation of its operands. If an instruction can execute in function units in different clusters, there will be preferred clusters which can utilize local register adequately. Before scheduling instructions, preprocessing to identify preferred clusters is introduced as follows.

A scheduling graph usually has four components:

$$G = (N, E, FUs, delay) \quad (1)$$

where N is the set of instructions and E is the edge between two instructions representing true dependency. Each $n \in N$ has a delay given by *delay(n)* and the function units given by *FUs(n)* indicating function units that n can execute in. Any FU belongs to some cluster, the set of clusters that all the *FUs(n)* belong to is denoted by *clusters(n)*.

To record the preferred clusters and facilitate the latter register allocation, two new attributes are affiliated with instruction n, *prefercluster(n)* indicating the preferred clusters and *double(n)* indicating whether n runs by double destination mode.

The basic idea is Bottom-Up Greedy (BUG) algorithm. All the instructions are traveled in post-order. Any instruction and its predecessors form a set, and the preferred clusters of all the instructions in the set are their common clusters. If they have no common clusters, one of the predecessors can be run by double destination mode to write its result into global register files, so the preferred clusters are the common clusters of the rest instructions. Continue this process until the preferred cluster isn't null. When one predecessor is marked with double destination mode, its other successors which have been traced must be adjusted to fit the change. When the traveling is finished, all the instructions have been given values of *prefercluster(n)* and *double(n)*. Then the scheduling graph has been turned into:

$$G = (N, E, FUs, delay, prefercluster, double) \quad (2)$$

Preferred clusters should be considered at first when allocating every instruction a function unit in scheduling process. Once an instruction cannot execute in its preferred clusters, it should be run in double destination mode.

Register allocation problem in multiple register file architecture has been solved by many methods. We adopt the solution extending graph coloring model by incorporating register file port restrictions into a hyper-graph. But two points, instruction mode and spilling code must be taken care of specially. When an instruction is in single destination mode, local registers should be allocated firstly; when in double destination mode, associate local register and global register should be assigned. If registers are not enough, new save and load operations will be inserted to spill a symbolic register to memory and then restore it to a register, the destination operand of load instructions must be assigned global registers.

At last in code emission phase, we deploy new assembly format different from Itanium's. In ORC framework assembly format is saved as a kind of retargetable information in machine description. Change the assembly format in machine description files, and then the right assembly code can be printed out.

In conclusion, ORC framework has characters of high flexibility and good retargeting ability when orienting toward VLIW processors. Only less effort is needed to be applied on the framework to generate a new retargetable compiler, and optimization for ASIP architecture can easily be incorporated into it to achieve higher performance.

5. Experimental result

Programs from the UTDSP benchmark suite [9] are used to verify the performance of the new compiler. These programs are divided into two sets of six "kernels" and twelve "applications". The "kernels" programs, which include *fft*, *fir*, *iir*, *latnrm, lmsfir* and *mult*, are compiled by the new compiler and then run in a simulator to get the runtimes in the target architecture.

Table 1 Several architectures in the target model

	N1	N2	N3	Clusters			
				1	2	3	4
A1	2	16	16	AL,MP	LS,AB	-	-
A2	2	32	32	AL,MP	LS,AB	-	-
A3	4	16	16	AL,MP	LS, AB	AL, LS	MP,AB
A4	4	32	32	AL,MP	LS, AB	AL, LS	MP,AB

N1: number of clusters
N2: number of registers in every local register file
N3: number of registers in global register file

Several parameters, number of local registers, issue width and function units, are selected in the target architecture model to form several architectures, as shown in Table 1. The column named "clusters" indicates what function units exist in every cluster, for example, in the first row there are two clusters: one cluster includes AL and AB function unit, and the other includes LS and MP unit. The architecture representing by the first row is the baseline architecture, and the runtime of benchmarks on this architecture are regarded as baseline runtime. For convenience, these architectures are named A1, A2, A3 and A4 respectively.

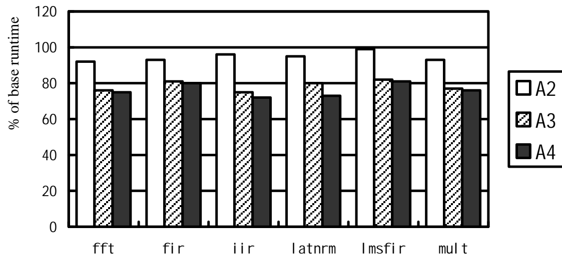

Fig.3 Performance of architectures in the model with different parameters normalized to the baseline

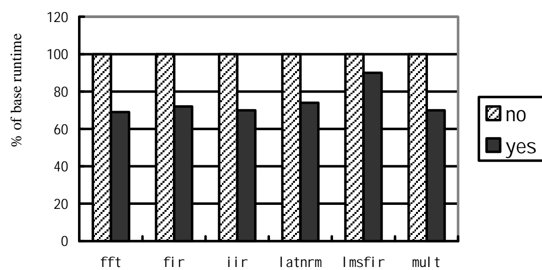

Fig.4 Performance improvement when introducing preprocessing algorithm before scheduling phase

All the benchmarks are compiled into object code by the retargetable compiler on the four architectures listed in Table 1, then are run in the target architecture simulator to get the their runtime. Fig.2 shows the runtime of the other three architectures normalized to the baseline, where the labels in the right correspond to different architectures in Table 1, for example, the white rectangles with label "A2" correspond to the second row which represents architecture with two clusters and 32 registers in every local and global register file. Compared with the baseline, architecture A2 has double registers, architecture A3 has double function units, and A4 has double registers in every register file and double function units. From Fig.2, it is obvious that shorter runtime is achieved when more registers and FUs are available. At the same time we see only little benefit of using double registers when we compare A2 with A1 and A4 with A3, and much benefit is attained by using double FUs when we compare A3 with A1 and A4 with A2. In a word, this experimental result shows retargeting ability of the new compiler.

In Section 4 we present preprocessing algorithm before scheduling phase and relevant changes in register allocation phase. Fig.3 gives the contrast of benchmark runtime between compilation process with this optimization and that without. The target architecture of compilation in this experiment is the baseline architecture, namely architecture A1. The labels "yes" or "no" indicate whether this optimization is introduced. It's obvious that higher performance is achieved when introducing this optimization, which indicates the validity of the preprocessing algorithm before scheduling.

6. Conclusion

In this paper a new method of getting a retargetable compiler by making modifications over ORC framework is presented. In [12], we have proposed a new register file access architecture of VLIW ASIP. Targeting at this architecture model and according to the new method, we generate a retargetable compiler based on ORC framework. The experimental results prove that this is a feasible method that only needs less effort and can incorporate optimizations for ASIP architecture easily.

References

[1] Manoj K.; M. Balakrishnan; and Anshul Kumar, ASIP Design Methodologies: Survey and Issues. *VLSI Design, 2001. Fourteenth International Conference on*, pp.76 – 81, 3-7 Jan. 2001.

[2] P. P. Chang; S. A. Mahlke; W. Y. Chen; N. J. Warter; and W. W. Hwu, IMPACT: An Architecture Framework for Multiple-Instruction-Issue Processors, *Proceedings of the 18th International Symposium on Computer Architecture (ISCA)*, pp. 266-275, 1991

[3] Bingfeng Mei; Vernalde S.and Verkest D., DRESC: A Retargetable Compiler for Coarse- grained Reconfigurable Architectures. *Field- Programmable Technology, Proceedings. 2002 IEEE International Conference on,* pp.166–173, 16-18 Dec. 2002.

[4] Overview of Trimaran Compiler, http://www.trimaran.org

[5] Open Research Compiler for Itanium Processor Family, http://sourceforge.net/projects/open64.

[6] Zalamea, J.; Llosa, J.; Ayguade, E.; Valero, M., Hierarchical clustered register file organization for VLIW processors", *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pp.10, 22-26, April 2003.

[7] Agarwala, S.; Anderson, T.; Hill, A.; Ales, M.D. and Damodaran, R. et al.; A 600-MHz VLIW DSP, *Solid-State Circuits, IEEE Journal* of, 37(11): pp. 1532-1544, Nov. 2002.

[8] Yanjun Zhang; Hu He; and YiheSun, A New Register File Access Architecture for Software Pipelining in VLIW processors. *Asia and South Pacific Design Automation Conference (ASPDAC)***,** Jan**.** 2005

[9] C. Lee and M. Stoodley, UTDSP BenchMark Suite. http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/-UTDSP.tar.gz