Design Space Exploration for Sobel Application using OpenIMPACT(Opensource Retargetable Compilation for VLIW Architecture)

[1]Debyo Saptono
[2]Vincent Brost
[3]Fan Yang

[1]Gunadarma University(debyo@staff.gunadarma.ac.id, debyo.saptono@u-bourgogne.fr)
[2,3]LE2I-CNRS 5158 Laboratory University of Burgundy

Abstract

Retargetable compilation infrastructure bring to growth of application-specific programmable systems which directly supporting the different target architectures and design space exploration (DSE) for the instruction set architecture and microarchitecture of the processor under development. There are three categories in this technology costumized,, semiretargetable and retargetable compiler. In DSE retargetable compilation methodology , permit to determine the optimal combination of hardwired components for example IALU, FALU ,Memory,Branch and programmable elements to get better performance that be measured by cycle count/total execution. DSP TI Processor Model as target architecture implemented, we have simulated for Sobel Application on VLIW architecture for observing optimal hardwired component needed in embedded system. With Optimization facility in compiler , result of simulation at variant model defined on system, giving information of Superblock and Hyperblock types can generate code that be executed processor better than Classical type. Model unroll looping in Optimization improved performance simulation until 50% unless in Classical type.

keywords : Retargetable compilation, DSE, VLIW architecture, DSP TI Processor Model, Sobel Application

1 Introduction

Developing of efficient retargetable compilation Infrastructures will make the good growth of application specific programmable systems because directly supporting the different target architectures and the design space exploration for the architecture and micro-architecture of the processor being developed.

The evaluation of any candidate architecture needs a compiler to map the applications to the architecture and a simulator to measure the performance. Because it is desirable to evaluate multiple candidates, a retargetable compiler (and simulator) is highly valuable. The difference between non retargetable/customization and retargetable illustrated in Figure .1.

General-purpose register files and usages of the registers also need to be specified. This type of machine description system serves as the interface between the machine-independent and the machine-dependent part of the compiler implementation.

A fully retargetable compiler is aimed at minimizing the coding effort for a range of targets by providing a friendlier machine description interface. Retargetable compilers are important for application-specific instruction-set processor (ASIP) (including digital signal processor [DSP]) designs.

Table 1: Comparison of Retargetability compiler

Type	Re-usability	Define Archi	Example Compiler	Status
Parameterizable	Little	Closed	VEX C	Openfree
SemiRetargetable	Midle	Undefined	LCC, GCC	Opensource
FullyRetargetable	High	Using ADL	OpenIMPACT	Opensource

Different degrees of retargetability exists to achieve this goal. According to the classification in [9], compilers can be assigned to one of the following classes:

Parameterizable: Such compilers can only be retargeted to a specific class of processors sharing the same basic structure. The compiler source code is largely fixed. The machine descriptiononly consists of numerical parameters such as register file sizes, word lengths, the number of functional units, or different instruction latencies.

User retargetable: An external machine descrip-

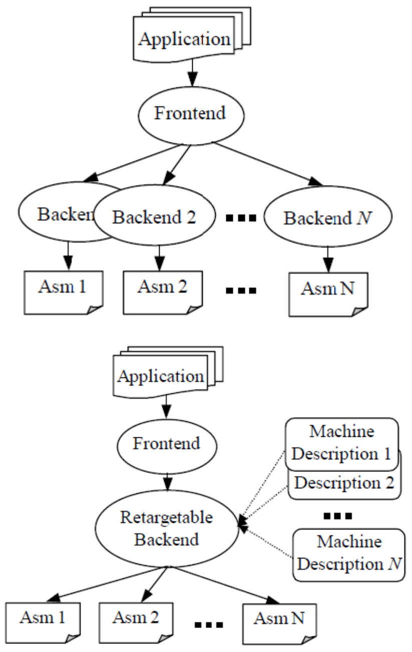


Figure 1: The difference between non-retartegable and retargetable

tion given in a dedicated language contains the retargeting information. All information required for code generation is automatically derived from this description. The specification does not require in-depth compiler knowledge and hence, can be performed by an experienced user.

Developer retargetable: Retargeting is also based on an external target description. However, the specification requires extensive compiler expertise usually possessed only by very experienced users or compiler designers.

2 Concept of DSE Retargetable Compiler VLIW Architecture

2.1 Design Space Definition

With the OpenIMPACT simulation environment, the source code of each application that will be implemented on target architecture posseses its own MAKEFILE . At the first time, Architecture Processor Model can be developed using HMDES architecure description languages (ADLs). In Retargetable Compiler, source code will be converted to Lcode assembly with optimization corresponding with processor model defined in HMDES. Simulating process in this environment can use superscalar, EPIC or VLIW architecture with according to target definition in Retargetable simulator[9]

2.2 DSE with OpenIMPACT Compiler

OpenIMPACT compiles the original source code into an assembly intermediate representation (IR) called Lcode.

The Lcode produced is optimized for ILP, but not for a specific machine. The compilation and simulation tools for this evaluation were provided by the OpenIMPACT compiler, produced by group of Wenmei Hwu at the University of Illinois [3]. The OpenIMPACT environment includes a trace-driven simulator and an ILP compiler. The simulator enables both statistical and cycle-accurate trace-driven simulation of a variety of parameterizable architecture models, including both VLIW and in-order superscalar datapaths. The Open IMPACT compiler supports many aggressive compiler optimizations including procedure inlining, loop unrolling, speculation, and predication. IMPACT organizes its optimizations into three levels:

- Classical(O) : performs only traditional local optimizations .
- Superscalar/Superblock(S) : adds procedure inlining, loop unrolling, and speculative execution.
- Hyperblock(HS) : adds predication (conditional execution).

The OpenIMPACT compiler enables an architecture independent analysis through their low-level intermediate representation, Lcode. The Lcode representation is essentially a large, generic instruction set of simple operations like those found on typical RISC architectures, but not biased towards any particular architecture. Such a generic instruction set enables architecture-independent evaluation.

2.3 DSP TI Prossesor Model

Information about the machine needed by the compiler is broken down into six types of information, each of which has an associated hierarchy of sections. Register information

captures the types and overlap of registers. Format information specifies what operands are allowed by each type of operation. Resource information captures the resource usage patterns. Latency information specifies the latencies for the sources and destination of operations. Operation information specifies the opcodes of the machine

and associates each opcode with a format, resource usage and latency information. Compiler specific information captures other information needed by the compiler[8].

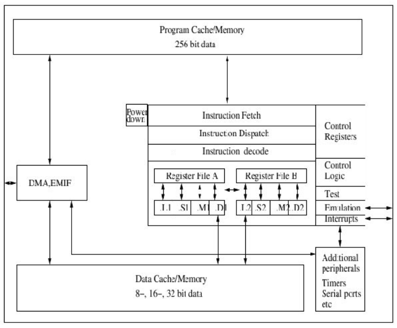


Figure 2: The Structure of TMS320 C6201 architecture

With using HMDES ADLs [6], we can define slot for TI model using SECTION Resource as part of Resource information like bellow :

```
{
    $for(I in $0..${LAST_SLOT})
    (slot${I}  (slot(${I}));}

    $for (I in $1..${WIDTH})
    (  decoder${I}  ( );}
Lunit1();Lunit2();Munit1();Munit2();
Dunit1();Dunit2();Sunit1();Sunit2();
}
```

2.4 VLIW Processor

VLIW processor use a long instruction word that is a combination information about the machine needed by the compiler is broken down into six types of information, each of which has an associated hierarchy of sections. Register information several operations combined into one single long instruction word. This allows a VLIW microprocessor to execute multiple operations in parallel [1] [2] [4][6].

3 Methodology

DSE is important in computer architecture design because no single design is optimal in every aspect. DSE also helps to determine the optimal combination of hardwired components and programmable elements. With HMDES ADLs, we define DSP TI processor model with variant in number of IALU, FALU, Memory Unit, and Branches ilsutrated in table2. For this experiment used model 1 (IALU=FALU=Memory Unit= Branches=1) until model 8.

Table 2: Configuration of VLIW DSP TI Processor Model

Comp\Model	1	2	3	4	5	6	7	8
IALU	1	2	3	4	5	6	7	8
FALU	1	2	3	4	5	6	7	8
Memory unit	1	2	3	4	5	6	7	8
Branch unit	1	2	3	4	5	6	7	8

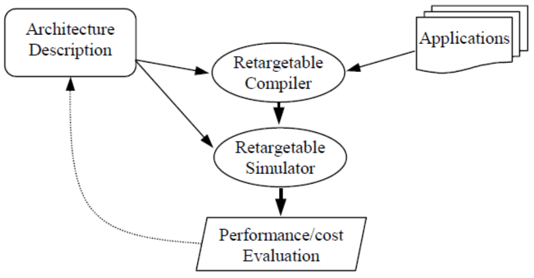
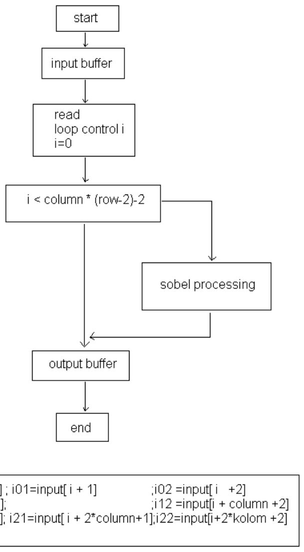


Figure 3: DSE Retargetable Compiler Methodology



```
i00=input[ i              ] ; i01=input[ i + 1]          ;i02 =input[ i   +2]
i10=input[ i +  column   ];                              ;i12 =input[i + column +2]
i20=input[ i + 2*column ]; i21=input[ i + 2*column+1];i22=input[i+2*kolom +2]
```

Figure 4: Process flow diagram loops on Using Sobel Edge Detection and . Sobel Kernel Algorithm

Figure 5: Grayscale bmp with 20x20 for input (left) and output(right) images

Sobel application choose be simulated by simulator because there is loop processing that can be observed related with unroll loop mode in the OpenIMPACT compilator and simulator toolchain. The simulator executes the code and produces performance metrics such as cycle count (total execution). The metrics are analyzed and used to guide the tuning of the architecture description. The process iterates until a satisfactory cost-effective architectural trade-off is found.

4 Experiment Result

In figure 5, total instruction that be generated by compiler is 11618 instruction and total execution (cycles) in processor model 4 (4905 cyles) better than others for Classical Optimization.

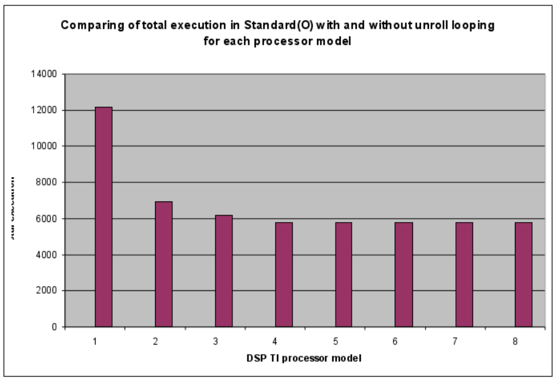


Figure 6: Total execution for each processor model in Classical for Sobel application

In figure 6, total instruction that be generated by compiler is 11540 instruction and total execution (cycles) in processor model 3 (4905 cycles) better than others for Superblock Optimization without unroll looping.

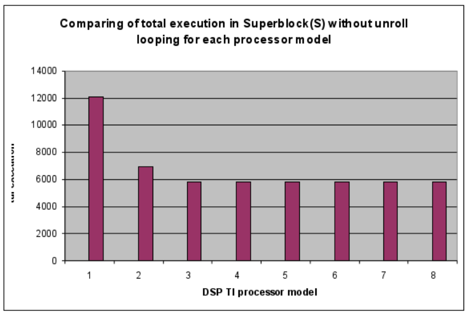


Figure 7: Total execution for each processor model in Superblock without unroll for Sobel application

With unroll looping mode for Superblock as shown at figure 9, total execution for each processor model is relative less than without unroll at the same optimization unless at model 1.

In figure 9, total instruction that be generated by compiler is 11820 instruction and total execution (cycles) in processor model 3 better than others for Superblock Optimization without unroll looping.

With unroll looping mode for Hyperblock as shown at figure 9, total execution for each processor model is relative less than without unroll at the

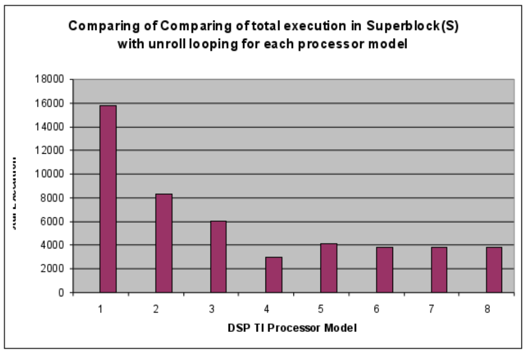


Figure 8: Total execution for each processor model in Superblock with unroll for Sobel application

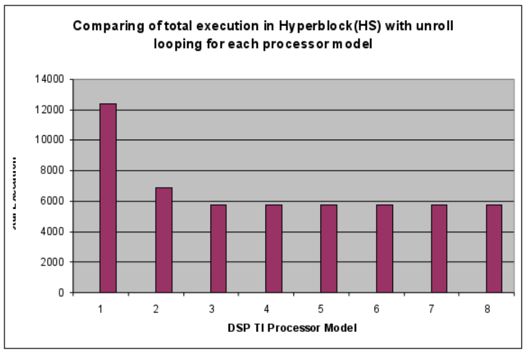


Figure 9: Total execution for each processor model in Hyperblock without unroll for Sobel application

same optimization unless model 1.

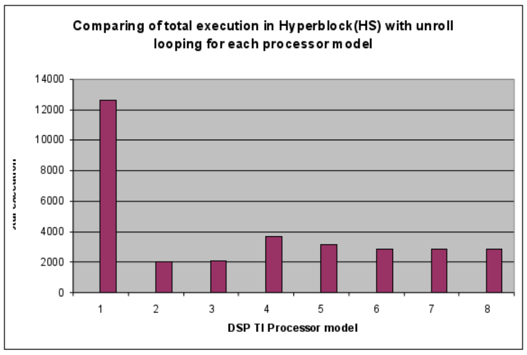


Figure 10: Total execution for each processor model in Hyperblock with unroll for Sobel application

5 Conclusion and Perspective

In this papers, we have presented a VLIW architecture design space exploration methodology using the OpenIMPACT compiler and implemented it for sobel application. We have proposed criteria based on optimization tools to analyze performance which allows us to get better setting for VLIW DSP

TI Processor model. Our results show that Superblock optimization with unroll and model prossesor number of 4 is better setting for obtain extract significant amount of ILP (Instruction Level Parallelism) from sobel application.

In future work, we hope to embed VLIW architecture on reconfigurable component FPGA using the OpenIMPACT compiler, because modern FPGA chips, with their large memory capacity and reconfigurable potential, are opening new frontiers in rapid prototyping of embedded systems.

References

[1] F. Yang D. Saptono, V. Brost and E. Prasetyo. Concept and development of modular vliw processor based on fpga. *ICSEM2010*, 2010.

[2] F. Yang D. Saptono, V. Brost and E. Prasetyo. Design space exploration for a custom vliw architecture : Direct photo printer hardware setting using vex compiler. In *In Proc of the 4th International Conference of SITIS 2008*, November 2008.

[3] Gelato.org. Impact advanced compiler technology, 2009.

[4] P. Faraboschi J.A. Fihser and C. Young. *Embedded Computing : A VLIW Approach to Architecure, Compilers and Tools*. 2005.

[5] W. Mei Hwu J.C Gyllenhaal and B.R Rao. Hmdes version 2.0 specification technical report impact-96-3. 1996.

[6] Weng Fook Lee. *VLIW Microprocessor Hardware Design For ASIC and FPGA*. Mc Graw Hill Proffesional, 2008.

[7] R. Leupers. *Retargetable Code Generation for Digital Signal Processors*. Kluwer Academic Publishers, 1997.

[8] M. Pandaivone V. Brost, F. Yang and N. Farrugia. Multiple modular vliw processors based on fpga. *Journal of Electronic Imaging, SPIE*, 16(2):110, April-June 2007.

[9] P. Shankar Y.N Srikant. *Compiler Design Handbook, Optimizations and Machine Code Generation*. CRC Press, 2003.