Impact of Source-Level Loop Optimization on DSP Architecture Design

Bogong Su
Dept. of Computer Science
William Paterson University
Wayne, NJ 07470, USA
bsuwpc@frontier.wilpaterson.edu

Jian Wang
Wireless Speech and Data
Nortel Networks, Montreal
H3E 1H6, QC, Canada
jiwang@nortelnetworks.com

Erh-Wen Hu
Dept. of Computer Science
William Paterson University
Wayne, NJ 07470, USA
ewh@frontier.wilpaterson.edu

Andrew Esguerra
Dept. of Computer Science
William Paterson University
Wayne, NJ 07470, USA
njd44@cs.wilpaterson.edu

Abstract

How to select an architecture which has better performance cost ratio, how to design a balanced architecture which can fit most DSP programs and how to generate a custom-fit DSP processor are important topics now. This paper presents a novel methodology based on source-level loop optimization which can avoid expensive retargetable compiler and simulator, and exhaustive experiments.

1. Introduction

Many new DSP architectures have been announced recently [Eyre98, Step98]. How to select an architecture which has better performance cost ratio, how to design a balanced architecture which can fit most DSP programs and how to generate a custom-fit DSP processor become important topics. The popular method now is to conduct exhaustive experiments on an architecture model in which the number of various function units (such as ALUs, multipliers, etc.) is adjustable [Fis96, Sag98]. This method requires an expensive retargetable compiler system and considerable amount of experiment time.

We present a novel methodology - using source-level loop optimization [Su99] to estimate what kind of architectures and how many resources are good for most DSP kernels and many DSP applications. Our motivations are as follows: 1. Working at source-level eliminates the need for expensive compiler and the lengthy experiment time. 2. Loop execution is the major component in DSP kernels and most DSP applications, therefore we can get a good estimate of architectural requirement.

2. Architecture Model

Figure 1 shows our simplified VLIW architecture model where one register file is connect to several ALUs, multipliers, and memory ports. Since our preliminary work is focused on these major resources, we made the following assumptions in our experiments:

1. One cluster. Clustering is an important problem in real design [Fis96], it closely relates to the cost and performance, in particular, when the number of ALUs and multipliers are large.
2. All function units have same latency. As our basic approach is software pipelining [Lam86, Rau81], as long as there is no loop carried dependency in a loop, this assumption will not affect the throughput of software pipelining. Our study confirms that the percentage of loops having loop carried dependency in DSP programs is indeed quite small.
3. Some special architectures of conventional DSP such as one-cycle MAC, multiple data memory banks, multiple data address generators, zero-overhead looping can be easily converted to function units such as adders, multipliers of VLIW architecture.

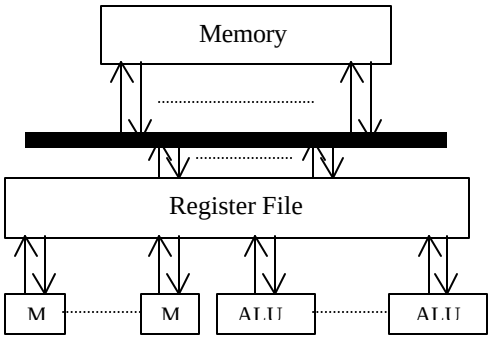

Figure 1 Architecture Model

3. Methodology

Our basic method is as follows:

1. Decompose each C statement in the innermost loop of DSP source programs to elementary operations.
2. Build data dependency graph of elementary operations.
3. Apply Modulo Scheduling software pipelining [Lam86, Rau81] on elementary operations to find the loop body with minimum length. For loops with no loop-carried dependency, the minimum length is determined by resources. If resources are unlimited, the minimum length is one. For a small number of loops with loop-

resources or by the data dependencies. In particular, many loops in our collected DSP programs are quite small, they don't have enough operations to make the hardware resources busy thus limit the increase of speedup. On the other side, loop unrolling will increase the code size and the number of required registers. It is beyond this paper.

Our experimental results in this paper are based on typical VLIW architectures. The future DSP architectures may be integrated with some special conventional DSP architectural features like circle buffering and hardware looping. Obviously, the introduction of these new architectural features may enhance the performance of a VLIW based DSP processor. Our future work is to study the impact of these new features on a VLIW based DSP architecture design.

Our collection of DSP programs has not completed yet, we will collect more programs, in particular, those new programs in multimedia and communication applications. For those users who have special application programs or focus on some kernel programs, they can still use our methodology to conduct experiment on their own application programs or change the weight of some kernel programs, then can select the DSP architecture which is suitable for their specific application.

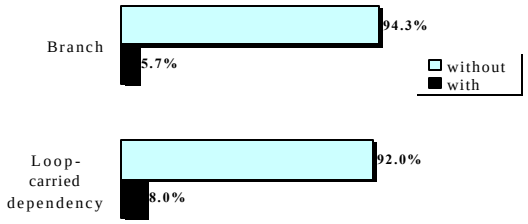

Figure 5 Branch and Loop-carried dependency in innermost loops

9. Conclusion

In this paper, we propose a new approach based on source-level loop optimization to predict the speedup and suggest to use the ratio of the average number of operations in critical resource loops as the criterion to select architecture. It will be a useful tool for the manufactures to design a new DSP architecture and for the users to select DSP products. Our approach does not need expensive retargetable compiler system and exhaustive experiments, its implementation is easy and can even be conducted manually for simple applications.

ACKNOWLEDGMENTS

This work was partially supported by the 1998-1999 ART Award and undergraduate research grants from William Paterson University of New Jersey. We kindly thank Prof. H. Meyr for providing DSPSTONE code and Dr. Corinna Lee for providing UTDSP benchmarks. We also like to thank Edisberto Rosado for his effort in analyzing the benchmark programs and compiling data for this work.

References:

[Emb95] P. Embree, *C Algorithms for Real-Time DSP*, Prentice Hall, 1995.
[Emb99] P. Embree and D. Danieli, *C++ Algorithms for Digital Signal Processing*. Prentice Hall, 1999.
[Eyre98] J. Eyre and J. Bier, "DSP Processors Hits the Mainstream", *IEEE Computer*, Aug. 1998.
[Fis96] J. Fisher, P. Faraboschi and G. Desoli, Custom-Fit Processors: Letting Applications Define Architectures. *Proc. of MICRO-29*, 1996. 324-335.
[Hal99] T. Halfhill, StarCore Reveals Its First DSP, *Microprocessor Report*, May 10, 1999
[Lam87] M. Lam, A Systolic Array Optimizing Compiler, Ph.D thesis, May 1997.
[Lev97] M. Levy, C Compilers for DSPs – Flex their Muscles, *EDN*, June 5, 1997.
[Pre92] W. Press etc., *Numerical Recipes in C*, Cambridge University Press, 1992
[Rau81] B. R. Rau and C. D. Glaeser, Some Scheduling Techniques and an Easily Schedulable Horizontal Architecture for High Performance Scientific Computing. *Proc. of MICRO-14*, Oct. 1981.
[Sag98] M. Saghir, P. Chow, and C. Lee, A Comparison of VLIW and Traditional DSP Architectures for Compiled Code. downloaded from UT. 1998.
[Step98] Stephan Ohr, "TI defends its DSP turf", *EDTN Network*, Sept. 1, 1998.
[Stot99] E. Stotzer and E. Leiss, Modulo Scheduling for the TMS320C6x VLIW DSP Architecture, Proc. of LCTES'99, May 5, 1999
[Su99] B. Su, J. Wang, and A. Esguerra, Source-Level Loop Optimization for DSP Code Generation, Proc. of ICASSP'99, March 1999.
[Ziv94] V. Zivojnovie etc., DSPStone – A DSP-oriented Benchmarking Methodology, Proc. of ICSPAT, 1994.

carried dependency, the minimum length is determined by the strongly connected circles in those loops. We use If-conversion for global cases in loops. We do not do loop unrolling, as it will increase the size of register file and instruction memory dramatically.

4. Process statistically the number of various kinds of resources needed to reach the loop body with minimum length.
5. Apply Modulo Scheduling software pipelining and change the number of critical resources to obtain the relation between speedup and the number of critical resources.

4. DSP Benchmarks

We have conducted experiment on 90 innermost loops in 45 DSP kernels and 5 DSP applications [Emb95, Emb99, Lev97, Pre92, Sag98, Ziv94] as shown in Table 1.

Table 1 DSP Benchmarks

Type	Description	No. of programs	No. of innermost loops
Matrix/vector calculation	Basic matrix/vector manipulations	12	17
Statistics	Basic statistical calculations	1	4
Least square routines	LMS signal enhancement	2	6
Quantization	Quantization of samples	1	2
FFT	Forward and inverse fast Fourier transforms	2	5
FIR	FIR filtering operations	4	4
IIR	IIR filtering operations	1	1
Median filter	Median filter operations	1	2
DFT/IDFT	Forward and inverse discrete Fourier transforms	3	7
Convolution	Convolutions routine	3	6
Lattice	Lattice filter	1	2
Interpolations	Interpolation using FFT and IFFT	2	4
Window routines	Spectral side lobe reduction	5	6
Vocoder code book	Speech compression	1	1
Image processing	JPEG compression and filtering operations	3	3
Correlation	Correlation of two vectors	1	1
Wavelet	Discrete wavelet transform	1	1
Signal generation	Summation of sinusoids	1	2
Adaptive filter	ARMA adaptive IIR filtering	1	4
Complex FFT	Doppler Radar signal processing	1	3
Music processing	Kaiser window pitch shift/sample rate change	1	4
Music synthesis	Voice generation	1	2
FFT spectral estimate	Speech spectral analysis	1	3
	Total:	50	90

5. Loop Iterations

Figure 2(a) and 2(b) present the number of loop iterations in the innermost loops of DSP benchmarks and the number of loop nested levels. Most innermost loops have big enough iteration numbers, it means that (1) The software pipelining can hide the execution time of its prelude and postlude and improve efficiency significantly, (2) the execution time of innermost loops takes big percentage of the whole DSP benchmark programs, especially for those loops which have more than two nested levels, Figure 2(b) shows that more than 35% of loops studied are nested. It means that the results of our study have enough accuracy.

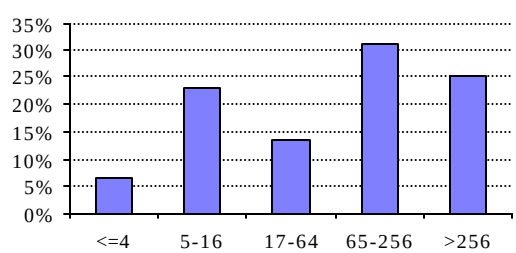

Figure 2(a) The Number of Loop Iterations

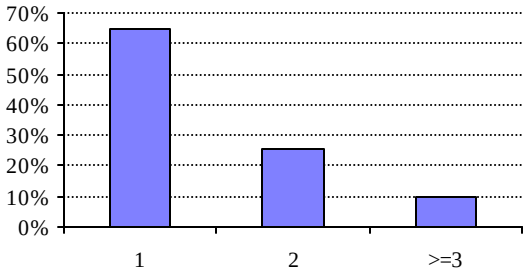

Figure 2(b) The number of loop nested levels

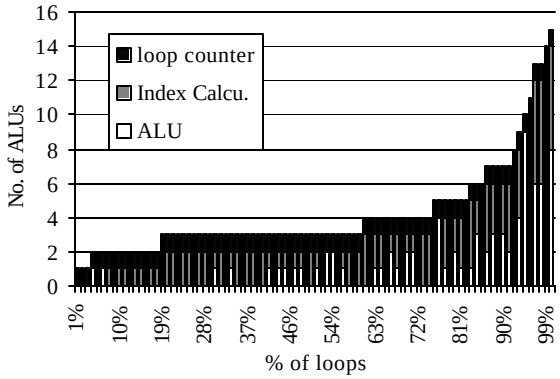

Figure 3(a) Requirement for ALUs

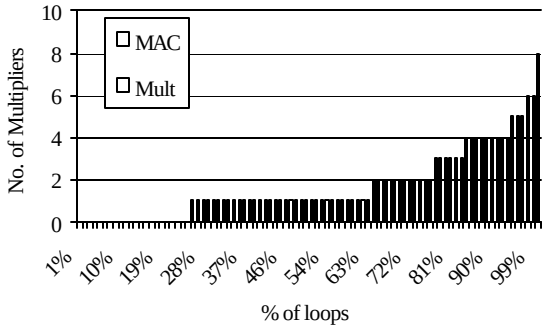

Figure 3(b) Requirement for Multipliers

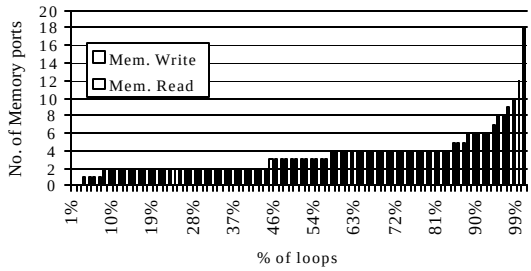

Figure 3(c) Requirement for Memory ports

6. Resource Requirements for Minimization of Loop Length

Figure 3(a) shows the distribution of the number of ALUs required by the innermost loops in DSP benchmarks to reach the minimum loop length, which considers the total requirement from ALU operations, calculations for address generation, and loop counting. We find that 8 ALUs can fit more than 90% loops, and15 ALUs fit all loops.

Figure 3(b) shows the distribution of the number of multipliers required by the innermost loops in DSP benchmarks to reach the minimum loop length, where consideration is given to both of the multiplication and MAC operations. It is noted that 2 multipliers fit 76% of loops,, 4 multipliers fit 94% loops and 8 multipliers fit for all loops.

Figure 3(c) shows the distribution of the number of memory ports required by the innermost loops in DSP benchmarks to reach the minimum loop length, where consideration is given to both of the memory reads and writes. It is noted that 4 memory ports fit more than 80% of loops, 6 memory ports can fit more than 90% loops, however some loops have many memory accesses, we must have 18 memory ports to fit all loops.

7. Balanced Architectures

One of the basic problems of the design of DSP architecture is to determine the number of major function units, such as ALUs, Multipliers, and memory ports. An intuitive approach is to use the ratio of the average number of operations of those function units based on the study of benchmark programs. Table 2 shows that the average numbers of ALU, multipliers, and memory operations are 4.2, 1.6 and 3.5 respectively, which leads to that the ratio of the number of these three major function units is 3:1:2. We shall call the ratio 3:1:2 Ratio_1.

However, we realize that the result of software pipelining is determined by the critical resource only, not by all resources. Therefore we must use the average number of operations of those loops in which that function unit is the critical resource. Table 2 gives the percentage of critical resource loops and the new average numbers of operations. It shows that ALU is the critical resource in most loops (78%). Multiplier is the critical resource in 6% loops only, however its average number of operations is almost the same as that of ALU. Corresponding to these new average numbers we obtain the Ratio_2, it is 3:3:4.

Table 2 Determination of Architecture Ratio

	ALU	Mult	Mem
Average Ops	4.2	1.6	3.5
Ratio_1	3	1	2
Critical Resource Loops	78%	6%	17%
Average Ops in critical resource loops	4.3	4.2	5.4
Ratio_2	3	3	4

By applying software pipelining and changing the number of critical resources we obtain the relation between speedup and the number of critical resources. We then change the architecture ratio of the number of critical resources such as of the number of ALUs to the number of multipliers to the number of memory ports to obtain the relation between the speedup and different architectures as shown in Figure 4. In this figure, the ratio such as 3:1:2 means that the number of ALUs : the number of

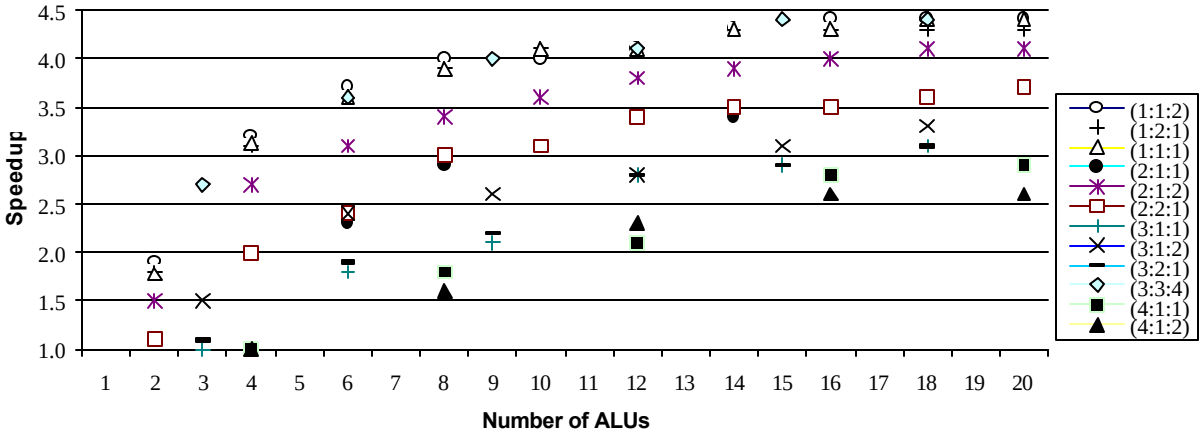

Figure 4 Relation between speedup and different architectures

Multipliers : the memory ports is 3:1:2. From Figure 4, we have the following observations:

(1) The speedup of Ratio_2(3:3:4) architecture is much better (30-40%) than that of Ratio_1(3:1:2) architecture.

(2) Ratio (1:1:1) architecture almost reaches the best speedup, and Ratio_2 (3:3:4) is very close to it both of speedup and cost. Therefore we can say that the ratio average numbers of operations of critical resources is a good heuristic in DSP architecture design.

(3) Even Ratio (1:1:1) architecture has almost highest speedup, but its cost is significant. Ratios (1:1:2) and (1:2:1) are better than it slightly but with higher cost. Figure 4 presents some ratios with higher proportion of ALU, it means less cost due to less proportions of multiplier and memory ports. We must make tradeoff between the performance and cost. In fact, some current VLIW DSP processors such as TI C6[Stot99] and SC140 of Star*core[Hal99] have higher proportion of ALU and limited memory ports. TI C6 has (3:1:1) ratio and its average speedup is 1.8. SC140 has more flexible architecture with 2 memory ports, hardware loop counter, 2 address arithmetic units, and 4 function units which can perform MAC or ALU operations, its speedup ranges from 1.4 to 2.2.

8. Discussion

Scalability problem has been faced by the designers of computer architecture and optimizing compiler recently. From Figure 5 we realize that just few DSP programs have loop-carried dependency, also Figure 4 shows that up to 12 ALUs the speedup is almost proportional to the number of function units. It means that the scalability should not be a serious problem. We will conduct further study on those loops with loop-carried dependency.

Figure 5 shows that very few DSP programs have branch in the innermost loop. i.e. the global case will not be a big problem in DSP architecture and compiler design.

Our methodology assumes no loop unrolling, however unrolling can improve the speedup when the parallelism is not limited either by the hardware