Submitted to the International Workshop on Compiler and Architecture Support for EmbeddedComputing Systems — CASES'98.
Other work related to this project can be found at http://www.eecg.toronto.edu/~pc/research/dsp/

A Comparison of Traditional and VLIW DSP Architectures for Compiled DSP Applications

by
Mazen A. R. Saghir, Paul Chow, and Corinna G. Lee
Department of Electrical and Computer Engineering
University of Toronto

1 Introduction

The high performance and low cost of traditional programmable DSPs are achieved through the use of architectural features optimized for the computationally-intensive operations in the inner loops of common DSP algorithms. Tightly-encoded instruction sets reduce instruction storage and bandwidth requirements, but limits the use of on-chip registers or accumulators. The specialized architectural features and tightly-encoded instruction sets make DSPs difficult targets for high-level language (HLL) compilers. Since commercial DSP compilers are often less sophisticated than optimizing compilers targeting general-purpose processors, DSPs must still be programmed in assembly language to achieve the most efficient and compact code. With the proliferation of DSPs as programmable cores in embedded systems-on-a-chip, the growing complexity of DSP applications, and the continued reduction in time-to-market cycles, more compiler-friendly DSP architectures and better DSP compiler technology are needed.

One alternative to traditional DSP architectures, used in new media processors and DSPs such as the Philips TriMedia [1] and the Texas Instruments TMS320C6x [2], is the very long instruction word (VLIW) architecture [3]. VLIW architectures use multiple functional units to exploit fine-grained parallelism. Typically, each functional unit executes the operations specified in the corresponding field of a long instruction word. VLIW architectures also have simple control units since they rely on their compilers to expose and exploit parallelism, and to schedule parallel operations into long instructions. However, a drawback of VLIW architectures is their high instruction storage requirements, unless they use some instruction compaction or encoding. Although other architectural styles, such as superscalar, SIMD, and multiprocessor architectures, may also be used for DSP applications, VLIW architectures are easier targets for HLL compilers; are more flexible in exploiting different forms of parallelism; and can easily be customized to meet the cost constraints of a target application.

In this paper, we compare the performance and area cost of VLIW architectures with traditional architectures for compiled DSP applications. Using a very conservative model, we show that VLIW architectures achieve at least a factor of 1.3 – 2 times the performance of traditional DSP architectures. From an estimation of the effects of the restrictive use of registers in traditional DSP architectures, we argue that the actual performance gain is at least a factor of 1.8 – 2.8. We also show that through the efficient encoding and storage of long instructions, instruction storage requirements can be reduced significantly. This yields VLIW architectures that occupy only 1.13 times the area of traditional DSP architectures.

The remainder of this paper is organized as follows. Section 2 provides an overview of the experimental tools and methodology used in this study. It describes our VLIW model architecture, optimizing C compiler, and suite of DSP benchmarks. It also describes how we modeled traditional DSP architectures, and how we compared their performance and cost to corresponding VLIW architectures. Section 3 discusses the impact of the different architectural styles on performance and cost. Finally, Section 4 presents the main conclusions of the paper.

2 Experimental Tools and Methodology

2.1 UTDSP: a VLIW Model DSP Architecture

To study the performance and area cost of VLIW DSP architectures, we developed the UTDSP. In its basic form, the UTDSP consists of nine functional units, which include two memory units for accessing two single-ported data-memory banks; two address units for performing address calculations; two integer units; two floating-point units; and a control unit for executing control operations. It also contains three separate register files, each containing 32 registers, for storing address, integer, and floating-point data, respectively. Long instructions in the UTDSP consist of nine operation fields that each control the execution of one of the functional units. Since we assume each functional unit executes a machine operation in a single clock cycle, the UTDSP can execute up to nine independent machine operations every clock cycle. To measure the execution performance of the UTDSP and gather run-time statistics, we also developed an instruction set simulator. The simulator runs the code generated by our optimizing C compiler.

2.2 Optimizing C Compiler

Our optimizing compiler consists of a C-language front-end and an optimizing back-end. The front-end, based on the Stanford University SUIF compiler [4], translates application programs written in C to the basic machine operations of our model architecture. It also applies traditional scalar optimizations and performs register allocation. The back-end exploits the DSP-specific features of our model architecture, and schedules parallel machine operations into long instructions.

2.3 DSP Benchmarks

To measure the impact of using VLIW architectures on performance and cost, we developed a suite of DSP benchmarks consisting of 12 kernels and 10 applications. The kernels are based on six common DSP algorithms, and for each algorithm, we implemented a version processing a small data set and another processing a large data set. The applications are taken from the areas of speech processing, image processing, and data communications.

2.4 Modeling Traditional DSP Architectures

To model traditional DSP architectures, we constrained the number of functional units and registers in the UTDSP to reflect the datapaths of two commercial DSPs. We also modified the compiler back-end to only generate combinations of parallel operations that are supported by the instruction sets of both DSPs. Furthermore, we constrained the number of registers available to the compiler to reflect the number of registers used in each DSP. Although the instructions generated by the modified compilers are still VLIW instructions — and could therefore be executed on our instruction-set simulator — each instruction corresponds, approximately, to an existing tightly-encoded instruction in the DSP being modeled. However, unlike the restrictions typically imposed by DSPs on the use of registers, which complicate automatic code generation [5], our modified compilers do not restrict the use of registers. The performance results of our modeled DSPs are therefore optimistic. For example, for two kernels, the code generated by a commercial DSP compiler [6] targeting one of the DSPs we modeled was 1.4 – 2 times slower — assuming single-cycle instruction execution — than the corresponding code generated by our modified compiler. Since both compilers were able to exploit the parallelism in the inner loops of the kernels, and since the quality of the code generated by our compiler is comparable to hand-coded assembly [7], the results highlight the significance of restricting the use of registers on performance. The effects of such restrictions should therefore be remembered when comparing the performance results of our VLIW architecture and the modeled DSP architectures.

The first DSP we modeled, *DSP_mot*, is based on the fixed-point Motorola DSP56001 [8], while the second DSP, *DSP_ad*, is based on the floating-point ADSP-21020 [9]. Both DSPs are also representative of other popular DSPs [10],[11],[12],[13]. To study the performance and cost of VLIW architectures with restricted parallelism and more parallelism, we increased the number of registers used in *DSP_mot* and *DSP_ad* to match those of the UTDSP architecture. The resulting models are called *UTDSP_mot* and *UTDSP_ad*, and are true VLIW versions of the *DSP_mot* and *DSP_ad* architectures in that they have the same functional units, but are not restricted by the instruction set architecture in exploiting available parallelism. Furthermore, since they have more registers, they make it easier for the compiler to generate more efficient code.

2.5. Measuring Area Cost

To measure the cost of different architectures, we developed a first-order area-estimation model for the UTDSP datapath. The model uses technology-independent estimates of the area occupied by different datapath components and a set of architectural parameters for describing a datapath. By specifying different parameters, different datapaths, such as the *DSP_mot*, *DSP_ad*, *UTDSP_mot*, and *UTDSP_ad*, can be modeled, and the area of their datapaths can be estimated.

3 Results and Analysis

3.1 Impact on Performance

Figure 1 shows the average performance of the traditional and VLIW DSP architectures relative to the base *DSP_mot* architecture, for both the kernel and application benchmarks, respectively. The generally poor performance of the traditional DSPs is mainly due to the smaller number of data registers and the restricted instances of parallelism supported by their instruction-set architectures. The smaller number of data registers increases the number of memory instructions in the code, which also increases execution time. In addition, the limited instances of supported parallelism do not always match those occurring in the applications, causing both DSPs to miss opportunities to exploit other instances of parallelism. One reason for the better overall performance of *DSP_mot* and *DSP_ad* on the kernels is the higher incidence of parallelism in the code matching that supported by their instruction sets. Finally, the higher performance of *DSP_ad*, relative to *DSP_mot*, is due to its larger number of data registers and its more flexible instruc-

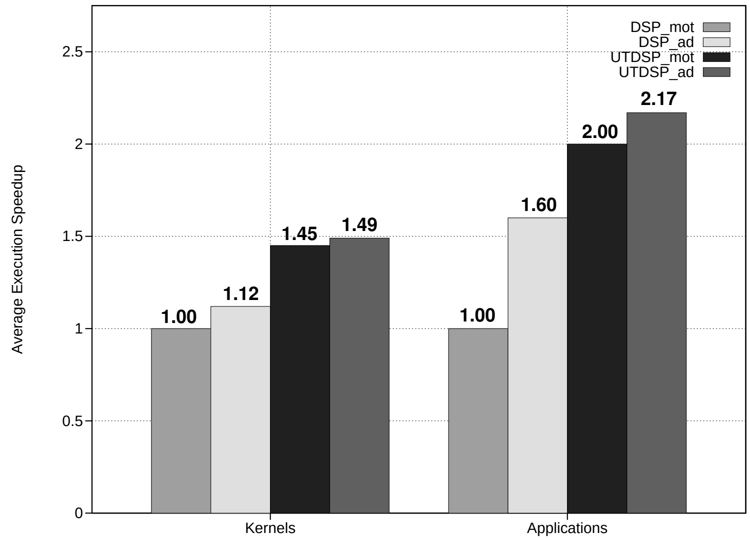


Figure 1 Average execution speedup relative to *DSP_mot*.

tion set that enables it to exploit more instances of parallelism.

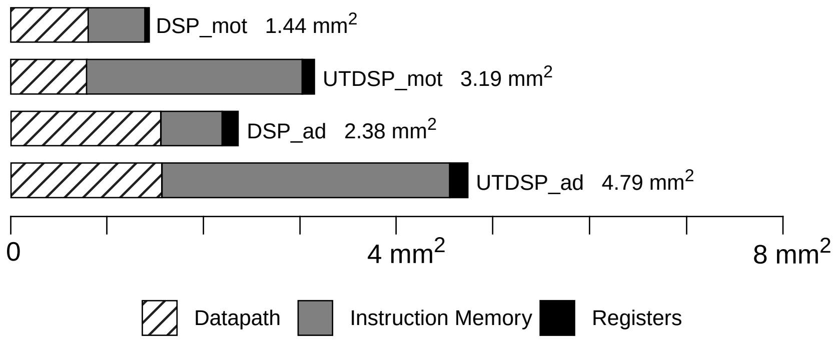
Figure 1 also shows that, for both the kernels and the applications, the relative performance of the VLIW architectures is significantly higher than that of the traditional architectures. This is mainly due to their larger register sets, which enable the compiler to generate more efficient code and eliminate additional memory instructions. To a lesser extent, it is also due to the greater flexibility of their instruction set architectures in exploiting parallelism. The combined effect of these two factors enable the VLIW architectures to execute 1.3 – 2.0 times faster than the traditional architectures. Since these results do not take into account the restrictions on the use of registers in tightly-encoded instruction sets, the VLIW architectures are likely at least 1.8 – 2.8 times faster than the traditional architectures, based on the simple comparison with a commercial compiler described in Section 2.4.

3.2 Impact on Area Cost

Table 2 shows the area estimates of the different DSP architectures. The estimates are calculated for a feature size of $\lambda = 0.25$ microns, and instruction storage requirements for the different DSPs are based on the average number of instructions generated for the application benchmarks.

Table 2 shows that registers occupy a small percentage of total chip area for all the architectures. For *UTDSP_mot* and *UTDSP_ad*, registers occupy only 4% of the area. Similarly, for *DSP_mot* and *DSP_ad*, registers occupy 3% and 7% of the area, respectively. Since the impact on overall cost is small, using more registers helps the compiler produce more efficient code.

Table 2 also shows that instruction memory occupies a significant proportion of the total area for all architectures. This is especially true for the VLIW architectures, since all long instructions are assumed to be stored in an unencoded format. Since the VLIW architectures have the same datapaths as their corresponding traditional architectures, their greater instruction storage requirements cause them to occupy 2.0 – 2.2 times the area of the traditional architec-



DSP	Total Area (mm^2)	Registers		Instruction Memory		Datapath		Average Instructions Stored
		Area (mm^2)	% Total Area	Area (mm^2)	% Total Area	Area (mm^2)	% Total Area	
DSP_mot	1.44	0.04	3	0.59	41	0.82	56	1068
UTDSP_mot	3.19	0.11	4	2.26	71	0.82	25	688
DSP_ad	2.38	0.17	7	0.63	27	1.58	66	576
UTDSP_ad	4.79	0.21	4	3.00	63	1.58	33	514

Table 1: Area estimates for λ = 0.25 µm.

tures. On examining the contents of instruction memory, we found that 70% of the space, on average, is occupied by NOPs. This suggests that a significant reduction in instruction storage requirements, and hence cost, can be achieved simply by using more efficient instruction encoding and storage techniques. In fact, reducing the instruction storage requirements of the VLIW architectures by 70% reduces their overall area to an average of 1.13 times the area of the traditional architectures.

4 Conclusions

In this paper, we compared the performance and area cost of traditional and VLIW DSP architectures for compiled DSP applications. We show that VLIW architectures can provide an improvement in performance of about 1.8 – 2.8 compared to traditional architectures. This is due to having more flexibility for parallelism and less restrictive register usage. With proper instruction encoding, the cost, in terms of increased area, is about 13%.

5 References

[1] Brian Case, "Philips Hopes to Displace DSPs with VLIW," *Microprocessor Report*, Vol. 8, No. 16, pp. 12–15, MicroDesign Resources, December 5, 1994.

[2] Jim Turley and Harri Hakkarainen, "TI's New 'C6x DSP Screams at 1,600 MIPS," *Microprocessor Report*, Vol. 11, No. 2, pp. 14–17, MicroDesign Resources, February 17, 1997.

[3] Joseph Fisher, "Very Long Instruction Word Architectures and the ELI-512," *Proceedings of the 10th International Symposium on Computer Architecture*, pp. 140–150, IEEE, 1983.

[4] Robert P. Wilson, et. al., "SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers," http://suif.stanford.edu/suif/suif1/suif-overview/suif.html, 1994.

[5] Ashok Sudarsanam and Sharad Malik, "Memory Bank and Register Allocation in Software Synthesis for ASIPs," *Proceedings of the International Conference on Computer-Aided Design*, pp. 388–392, IEEE/ACM, 1995.

[6] Markus Levy, "C Compilers for DSPs Flex Their Muscles," EDN Magazine, pp. 93–107, June 5, 1997.

[7] Vijaya K. Singh, *An Optimizing C Compiler for a General-Purpose DSP Architecture*, M.A.Sc. Thesis, University of Toronto, 1992.

[8] Motorola, *DSP56002 Digital Signal Processor User's Manual*, http://www.mot.com, 1990.

[9] Analog Devices, *ADSP-21020 User's Manual*, Second Edition, http://www.analog.com, 1995.

[10] Texas Instruments, *TMS320C5x User's Guide*, http://www.ti.com, 1993.

[11] Analog Devices, *ADSP-2100 Family User's Manual*, Third Edition, http://www.analog.com, 1995.

[12] Motorola, *DSP96002 IEEE Floating-Point Dual-Port Processor User's Manual*, http://www.mot.com, 1989.

[13] Texas Instruments, *TMS320C3x User's Guide*, http://www.ti.com, 1989.