International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:3, No:11, 2009

Performance Analysis of Digital Signal Processors Using SMV Benchmark

Erh-Wen Hu, Cyril S. Ku, Andrew T. Russo, Bogong Su and Jian Wang

***Abstract*—**Unlike general-purpose processors, digital signal processors (DSP processors) are strongly application-dependent. To meet the needs for diverse applications, a wide variety of DSP processors based on different architectures ranging from the traditional to VLIW have been introduced to the market over the years. The functionality, performance, and cost of these processors vary over a wide range. In order to select a processor that meets the design criteria for an application, processor performance is usually the major concern for digital signal processing (DSP) application developers. Performance data are also essential for the designers of DSP processors to improve their design. Consequently, several DSP performance benchmarks have been proposed over the past decade or so. However, none of these benchmarks seem to have included recent new DSP applications.

In this paper, we use a new benchmark that we recently developed to compare the performance of popular DSP processors from Texas Instruments and StarCore. The new benchmark is based on the Selectable Mode Vocoder (SMV), a speech-coding program from the recent third generation (3G) wireless voice applications. All benchmark kernels are compiled by the compilers of the respective DSP processors and run on their simulators. Weighted arithmetic mean of clock cycles and arithmetic mean of code size are used to compare the performance of five DSP processors.

In addition, we studied how the performance of a processor is affected by code structure, features of processor architecture and optimization of compiler. The extensive experimental data gathered, analyzed, and presented in this paper should be helpful for DSP processor and compiler designers to meet their specific design goals.

***Keywords*—**digital signal processors, DSP benchmark, instruction level parallelism, modified cyclomatic complexity, performance analysis.

E-W. Hu is with the Dept. of Computer Science, William Paterson University, Wayne, NJ 07470 USA (Phone: 973-720-2196, fax: 973-720-2973, e-mail: hue@wpunj.edu)

C. S. Ku is with the Dept. of Computer Science, William Paterson University, Wayne, NJ 07470 USA (Phone: 973-720-2960, fax: 973-720-2973, e-mail: kuc@wpunj.edu)

A. T. Russo was a CS major student in the Dept. of Computer Science, William Paterson University. He is now with ET International, 1501 Cash Mill Rd Newark, DE 19711 (Phone: 845-774-9675, e-mail: xavier776@yahoo.com)

B. Su is with the Dept. of Computer Science, William Paterson University, Wayne, NJ 07470 USA (Phone: 973-720-2979, fax: 973-720-2973, e-mail: sub@wpunj.edu)

J. Wang is with Wireless Speech and Data Processing, Nortel Networks, 2351 Blvd. Alfred-Nobel, St. Laurent, QC Canada, H4S 2A9 (Phone: 514-818-2541, e-mail: jiwang@nortelnetworks.com)

I. INTRODUCTION

THE rapid growth enjoyed by the DSP processor industry over the past few years is expected to continue at a double-digit annual rate [1]. To address the need for a wide variety of applications, various types of DSP processors based on different architectures have been introduced to the market [2]-[5]. As a result, performance measurement of DSP processors has become an important issue for both processor manufacturers and application developers. Manufacturers rely on performance data to improve processor design, and application developers use them to select a processor that best meets their design criteria.

Over the past decade, several performance benchmarks such as BDTI [6], EEMBC [7], MediaBench [8] and MiBench [9] have been proposed for DSP processors. However, none of these benchmarks seems to have included most recent applications. Moreover, none seems to have investigated how the performance of a processor can be affected by code structure, features of processor architecture and optimization of compiler. Such information can be important in gaining insight into how a DSP processor and its compiler handle various applications and thus helpful in improving processor design.

To address the aforementioned concerns we have recently developed and proposed a new benchmark [10] based on the Selectable Mode Vocoder (SMV), a speech-coding program from the third generation (3G) wireless applications [11]. SMV is known as the 3GPP2 standard speech codec for the CDMA2000 wireless network. Although it has been replaced by the fourth generation vocoder (4GV) recently, SMV remains to be one of the most efficient and sophisticated speech codecs capable of operating in modes at different bit rates. Also, the SMV fixed point C code consists of most of the common speech coding functions which exist in other CDMA wireless speech vocoders.

The new benchmark, referred to as the SMV benchmark in this paper, consists of eight kernels written in C language. Our goal in this paper is to use the new benchmark to measure the performance of popular DSP platforms. Since it is difficult to isolate the effect of the compiler, performance of a DSP platform is defined as the combined performance of both the DSP processor and its compiler [12]. In order to gain insight into how DSP compilers and DSP processors might be improved, we also studied how the performance of a processor is affected by the code structure of the benchmark programs;

certain architectural features of the processors including issue width, number of the available registers and functional units, support for hardware loop controls, branch delays, and load latency; and the characteristics of DSP compilers such as their ability to exploit instruction level parallelism (ILP).

We conducted extensive series of experiments on five popular DSP processors including Texas Instruments' TIC64, TIC62 [13] and TIC55, and StarCore's SC 110 and SC140 DSP processors. All kernels of the new benchmark are compiled by the compilers of the respective DSP processors and then executed on their simulators.

This paper is organized into five sections. In section II, we briefly review the rationale for our choosing the SMV as the new benchmark. In the same section, we also define the selection criteria for the benchmark and our methodology in building it (in section II). In section III, we discuss our experimental procedure and present the experimental results. In section IV, we discuss our experimental results and relate them to practical design considerations for both the DSP processors and their compilers. In section V, we provide our concluding remarks.

II. THE SMV BENCHMARK PROGRAM

As mentioned in the previous section, the new SMV benchmark consists of eight kernels chosen from the SMV program [14] for 3G voice wireless cell phone application. Provided below are brief summaries of A) the rationale based on which the SMV is selected as our new DSP benchmark, B) the criteria used in identifying and building our benchmark programs, and C) the methodologies adopted in conducting our experiment and analyzing the experimental results.

A. Rationale for selecting the SMV as the new DSP benchmark

1.Wireless communication is a major DSP application. In dollar amount, it accounts for almost three quarters of the entire digital signal processor market [15]. To measure the performance of DSP processors, some manufacturers still use the 2G wireless applications as their benchmarks. To the best of our knowledge, there has been no report on using the newer 3G wireless application to measure the performance of DSP processors.

2. Certain studies raise questions about the efficiency of DSP processors in their handling of complex and large applications. For example, [16] used a few DSP kernels and multimedia applications including speech coding and compression programs as the benchmark to compare the performance between the general-purpose Pentium II processor and the TIC62 DSP processor. They found that although TIC62 performs well on DSP kernels, its performance on more complicated applications is relatively poor largely due to the inability of its compiler to exploit the instruction-level parallelism among instructions in the application programs containing frequently occurred control-dependent data dependencies. Therefore, how to improve the performance of DSP processors on complex applications such as speech coding and the more recent 3G wireless application programs needs to be investigated. Moreover, Moore's Law is expected to hold true for at least till the end of the decade. Consequently, chips will continue to become larger and more complex, thus moving the embedded systems in the direction of embedded computing [17]. This trend suggests that software will become larger and more complex in the years to come, necessitating the use of larger and newer applications as benchmarks for more meaningful evaluation of the performance of DSP platforms.

B. Benchmark selection criteria

To facilitate the selection of kernels as benchmarks from hundreds of functions in the SMV application, we define certain selection criteria as described below.

1. Kernels must have long execution time, so we select only those functions that account for a significant portion of the total execution time of the SMV application. With the exception of some small built-in library functions, such functions tend to be leaf functions. Besides their long execution time, the code for selected leaf functions can be more easily exploited for instruction level parallelism (ILP) because they do not call other functions. Therefore, leaf functions are useful in measuring optimizing compilers ability to extract ILPs.

2.There are certain groups of functions exhibiting similar code structure. Within each group, only the most frequently executed one is selected to represent the entire group. In addition, the selected function must also meet criterion 1 mentioned above.

3.Only those functions that are at a certain level of complexity are (considered) candidates for selection because almost all DSP processors can efficiently handle simple functions. In this paper, complexity of a function is measured by the number of DSP operations in its innermost loop. In addition, complexity is also measured in terms of a new software metric: the modified cyclomatic complexity to be defined shortly in this section.

4. Only those functions having different code structures are candidates for selection. Since the major task of our SMV benchmark is intended to evaluate various types of DSP platforms, using a diverse set of functions with different code structures allows the performance of a wide variety of DSP platforms to be assessed.

C. Methodology for building the new benchmark

In the following, we summarize the methodology used to build the new SMV benchmark.

1. Our work is based on SMV v3.5fx program downloaded from [18]. SMV v3.5fx is written in C and it is executed in a simulated DSP environment on a PC with the help of a set of C library functions. For years, profiling tools have been routinely used to gather program execution profiles. Such profiling information is often used to improve the design of compilers and processor architectures [19]-[23]. In conducting our profiling experiment with the new benchmark, we

attempted to use but found the popular UNIX profiling tool *gprof* [24] unsuitable for our purpose because the result gathered by *gprof* does not truly reflect the execution profiles of the real DSP processors we have been investigating [10]. Noticing that in the original SMV program there are certain embedded small functions used for tallying the weighted DSP operations, we modified and extended the functionality of these functions and used them to gather values of WMOPs, which stands for Weighted Million of Operations and represents the true clock cycles of execution when the SMV program runs on real DSP processors. With these modified tools, we successfully applied them to nearly 100 functions and to more than 250 loops [25].

2. As mentioned in the previous section, functions selected as benchmark must meet the complexity criterion. Cyclomatic complexity (CC) measurement was introduced in 1976 [26], 27] and has since been applied extensively in software engineering to calibrate and measure the complexity of programs. To adapt it to the analysis of more complex code structure in this study that involves ILP, we modified its definition and used the notation CC* and its values to evaluate and compare the diversity and complexity of these functions, and to determine which of these functions are to be selected as kernels.

The structure of any program can be graphically depicted by a control flow graph. In the graph, CC = e – n + p + 1, where e, n and p denote the number of edges, nodes, and connected components, respectively. In fact, CC is actually the number of independent paths through the control flow graph. It can be proved that CC is equal to the number of conditions (loops and branches in a program) plus one. If there are two programs, one with two sequential loops and one with two nested loops, the CCs for these two programs are the same since both programs contain two conditions only. Considering the fact that it is more difficult to optimize nested loops and branches at instruction level, we take nested levels into account and define a modified cyclomatic complexity, CC*, which equals the number of loops and branches in the program, plus the number of nested levels of loops and branches.

3. SMV v3.5fx program consists of two major parts: the decoder and the encoder [28]. We studied the encoder part only because both parts have similar functions and their code structures are also similar. Furthermore, our profiling data showed that the encoder part accounts for 86% of the WMOPs or total execution time of the whole SMV program [25]. Considering the fact that the encoder contains more than 300 functions and most of them are infrequently called upon [10], we therefore focus on the more frequently referenced low-level leaf functions and used the total combined WMOPs and CC* of these functions as criteria to select kernels: the former represents the total number of DSP operations and therefore reflects the total execution time of those leaf functions and their similar functions, and the latter serves as an indicator of their code complexity.

4. To test the new benchmark, we used a sample audio file as the input to the SMV program and gathered the inputs to and outputs from the eight selected kernel functions in the program. These inputs and outputs are then used in the driver programs to test the kernels.

TABLE I
EIGHT KERNELS OF SMV BENCHMARK

No.	Name	CC*	C code lines	No. of loops	Levels of nested loops	WMOPs	Combined WMOPs	WMOPs of major innermost loop	WMOPs ratio
F1	FLT_filterAP_fx	3	11	2	2	776	1173	538	69%
F2	LPC_Chebps_fx	1	16	1	1	196	196	139	71%
F3	LPC_autocorrelation_fx	4	10	3	2	166	826	164	99%
F4	FCS_Excit_Enhance_fx	12	30	8	2	174	219	172	99%
F5	LSF_Q_New_ML_search_fx	32	68	13	3	939	939	859	91%
F6	c_fft_fx	15	52	6	2	288	381	254	88%
F7	FCB_add_sub_contrib_phi	6	21	1	1	129	183	101	78%
F8	PIT_LT_Corr_Rmax_fx	10	44	4	2	562	562	360	64%

TABLE II
MAJOR INNERMOST LOOPS IN KERNEL FUNCTIONS

No.	Loop count		Cond. branch	C code lines	DSP operations in source code			
	Max	Min			Arith/ logic	Mult / MAC	Mem. fetch	Total
F1	9	9	0	2	0	1	4	5
F2	3	3	0	11	7	4	1	12
F3	16	1	0	1	0	1	4	5
F4	80	34	0	5	3	1	5	9
F5	10	10	0	8	6	2	4	12
F6	32	1	0	13	17	4	10	31
F7	8	1	2	9	5	0	4	9
F8	80	80	0	1	0	1	2	3

Table I shows the eight kernel functions selected from the SMV program as the new benchmark. Notice that the total combined WMOPs of the eight kernels account for 58% of the entire execution time of the SMV encoder program. For each of the eight functions in the table, we refer to the innermost loop with the largest value of WMOPs as the major innermost loop. Dividing the value of the WMOPs of the major innermost loops of a function by its WMOPs, we obtain the WMOPs ratio, the percentage of the major innermost loop's WMOPs of the whole kernel function. As shown in the last column of Table I, the WMOPs of those major innermost loops account for nearly 80% of WMOPs of whole set of eight

kernel functions. The detailed descriptions of these kernels are presented in [10]. Table II presents some characteristics of the major innermost loop of eight kernel functions, which can be described in terms of loop behavior and program structure.

III. EXPERIMENTS

We conducted our experiment on five popular DSP processors including Texas Instruments' TIC64, TIC62 and TIC55, and StarCore's SC 110 and SC140. All kernels are compiled by the compilers of the respective DSP processors and subsequently run on their simulators. Table III lists the major hardware components of these DSP processors. In Section IV, we discuss how these architectural features of the DSPs such as issue width; numbers of registers, memory ports, and function units; hardware loop control; branch delay; and load latency might impact on the performance of a processor.

Table IV displays the numbers of clock cycles and Table V lists code sizes of the eight kernel functions of the SMV benchmark.

TABLE III
COMPARISON OF DSP ARCHITECTURES

Processor	TIC64	TIC62	TIC55	SC140	SC110
Issue width	8	8	2	6	3
Data path	6 ALU, 2 multipliers (2 16x16 or 4 8x8)	6 ALU, 2 multipliers (2 16x16)	2 MAC, 1 ALU, 1 Shifter	4 MAC/ALU, 1 Shifter	1 MAC/ALU, 1 Shifter
Data registers	2 x 32	2 x 16	4 ACC	16	16
Memory ports	2 (can be double bandwidth)	2	1	2 (can be double bandwidth)	2 (can be double bandwidth)
AGU	no	no	1 ALU, 8 addr. reg., 8 general purpose reg.	2 AAUs, 16 addr. reg.	2 AAUs, 16 addr. reg.
Pipeline depth	11	11	7	5	5
Branch delay	6 cycles	6 cycles	5 cycles	4 cycles	4 cycles
Load latency	5 cycles	5 cycles	1 cycles	2 cycles	2 cycles
Hardware loop support	no	no	for 3 nested levels	for 3 nested levels	for 3 nested levels

TABLE IV
NUMBER OF CLOCK CYCLES

No.	TIC64	TIC62	TIC55	SC140	SC110
F1	2994	14632	7825	5446	5619
F2	51	59	65	35	40
F3	8664	12920	16155	24040	24085
F4	7981	23983	13422	23432	23726
F5	44522	84153	83806	44367	67633
F6	10681	16247	8119	9334	11523
F7	231	315	140	138	139
F8	32960	54486	24188	/ *	/ *

* Compilers of SC140 and SC110 have bugs for function F8

TABLE V
CODE SIZE (bytes)

No.	TIC64	TIC62	TIC55	SC140	SC110
F1	1104	672	197	194	212
F2	336	480	189	240	268
F3	532	736	185	582	496
F4	1856	1504	360	465	524
F5	1480	2688	443	1264	1408
F6	1496	1344	327	1120	1200
F7	300	512	106	322	352
F8	1984	2528	381	/	/
AM	***1077***	***1308***	***274***	***598***	***637***

*AM represents Arithmetic Mean

TABLE VI
NUMBERS OF INSTRUCTIONS AND INSTRUCTION GROUPS OF THE MAJOR INNERMOST LOOPS IN KERNEL FUNCTIONS

No.	OPs	TIC64		TIC62		TIC55		SC140		SC110	
		G	I	G	I	G	I	G	I	G	I
F1	5	4.5*	13.9*	8	12	4	5	2	5	2	5
F2	12	13	17	13	23	13	15	10	13	10	11
F3	5	3	13	3	11	5	6	4	9	4	8
F4	5	9	34	18	23	4	5	3	5	3	5
F5	12	4.5*	11.2*	4	24	6	7	8	14	11	14
F6	31	40	53	78	81	31	38	44	82	66	85
F7	9	25	32	36	36	18	24	14	15	14	14
F8	3	6	17	3	10	2	3	/	/	/	/
AM	***9.1***	***13.5***	***22.0***	***20.4***	***27.5***	***10.4***	***12.9***	***12.1***	***20.4***	***15.7***	***20.3***

* TIC64 compiler unrolls major innermost loop fully

In order to gain deeper insight into the combined performance of the DSP system in relation to processor architecture and code structure, we use both dynamic and static methods in data gathering. The dynamic data are obtained through the use of compilers and simulators while the static data for the major innermost loops of the eight kernel functions are obtained by examining the generated assembly code for those functions using software de-pipelining technique reported in [29]. Table VI shows information about the major innermost loop for each of the eight kernel functions.

In the table, OPs stands for the number of DSP operations of the major innermost loop in C source code; I and G represent the numbers of instructions and instruction groups in the major innermost loop in native DSP assembly code, respectively. It is noted that all assembly instructions in each instruction group are executed in parallel within the same clock cycle.

IV. DISCUSSION

To facilitate the performance comparison of five different DSP processors, we normalize the number of clock cycles and code size to those of TIC62 as shown in Table VII and VIII, respectively. Since performance gain is often achieved at the expense of increasing the code size, it is of interest to know the relationship between performance and code size and such a relationship is presented in Table IX in terms of the product of normalized clock cycles and code size. Note that while Tables VII and IX use weighted arithmetic mean (WAM) with the combined WMOPs of kernel functions as the weights, Table VIII uses arithmetic mean.

To further investigate the effect of architectural and hardware features of a DSP processor on its performance, we use the approach reported in [30] and define three new parameters Ks, Kp, and Kc that are based on OPs, I, and G as described below.

- Ks = I/OPs as shown in Table X, which corresponds to the code size of a kernel function, because the larger the number of assembly instructions for the same number of DSP operations, the larger the code size.
- Kp = I/G as shown in Table XI, which reflects the amount of instruction-level parallelism; larger Kp means more instructions can be executed in parallel.
- Kc = G/OPs as shown in Table XII, which provides a measure of execution speed of a kernel function; the smaller the number of instruction groups requires less execution time for a given piece of code. By definition, Kc = Ks / Kp.

1. From Tables IV and VII, we observe that the number of clock cycles of TIC64 is much better than that of TIC62; the weighted arithmetic mean of the former is only about 50% of the latter. From Tables V and IIX, we note that TIC64's code size is also less than that of TIC62. The major reason for the superiority of TIC64 over TIC62 is because of TIC64 has a much more efficient compiler, as it is capable of performing better optimization such as more sophisticated software pipelining and unrolling the innermost loops. It is worthwhile to note that TIC55, a non-VLIW DSP, has better performance in clock cycles than TIC62; this is because TIC62 compiler generates much larger number of instructions and has less ability to exploit instruction level parallelism as shown in Tables X, XI and XIII. Tables V and VIII show that TIC55 has the smallest code size among all DSP processors; this is due to its non-VLIW architecture and some of its powerful specialized instructions typical of traditional DSP processors.

In Tables VII, VIII and IX, we note that both StarCore SC110 and SC140 are quite good in both clock cycles and code size. There are two major reasons; first, both of them have some powerful specialized instructions such as zero-overhead loop control instructions and double word load instructions, which lead to less number of DSP overall assembly instructions as shown in Table X. Second, the number of DSP operations in major innermost loops in SMV benchmark kernels is not very large, both SC110 and SC140 have sufficient function units to support moderate values of Kp as shown in Table XI.

TABLE VII
NORMALIZED CLOCK CYCLES

No.	TIC64	TIC62	TIC55	SC140	SC110
F1	0.20	1.00	0.53	0.37	0.38
F2	0.86	1.00	1.10	0.59	0.68
F3	0.67	1.00	1.25	1.86	1.86
F4	0.33	1.00	0.56	0.98	0.99
F5	0.53	1.00	1.00	0.53	0.80
F6	0.66	1.00	0.50	0.57	0.71
F7	0.73	1.00	0.44	0.44	0.44
F8	0.60	1.00	0.44	/	/
WAM	***0.58***	***1.00***	***0.71***	***0.62***	***0.69***

TABLE VIII
NORMALIZED CODE SIZE

No.	TIC64	TIC62	TIC55	SC140	SC110
F1	1.12	1.00	0.29	0.32	0.29
F2	0.73	1.00	0.39	0.56	0.50
F3	0.70	1.00	0.25	0.67	0.79
F4	1.20	1.00	0.24	0.35	0.31
F5	0.53	1.00	0.16	0.52	0.47
F6	1.12	1.00	0.24	0.89	0.83
F7	0.42	1.00	0.21	0.69	0.63
F8	0.78	1.00	0.15	/	/
AM	***0.82***	***1.00***	***0.21***	***0.49***	***0.46***

TABLE IX
NORMALIZED CLOCK CYCLES * CODE SIZE

No.	TIC64	TIC62	TIC55	SC140	SC110
F1	0.23	1.00	0.16	0.12	0.11
F2	0.63	1.00	0.43	0.33	0.34
F3	0.47	1.00	0.31	1.25	1.47
F4	0.40	1.00	0.13	0.34	0.31
F5	0.29	1.00	0.16	0.28	0.38
F6	0.74	1.00	0.12	0.51	0.59
F7	0.31	1.00	0.09	0.30	0.28
F8	0.47	1.00	0.07	/	/
WAM	***0.44***	***1.00***	***0.19***	***0.39***	***0.43***

TABLE X
Ks = I/OPs

No.	TIC64	TIC62	TIC55	SC140	SC110
F1	2.1	2.4	1.0	1.0	1.0
F2	1.4	1.9	1.3	1.1	0.9
F3	2.6	2.2	1.2	1.8	1.6
F4	2.2	4.6	1.0	1.0	1.0
F5	0.9	2.0	0.6	1.2	1.2
F6	1.7	2.6	1.2	2.6	2.7
F7	3.6	4.0	2.7	1.7	1.6
F8	5.7	3.3	1.0	/	/
AM	***2.6***	***2.9***	***1.2***	***1.5***	***1.4***

TABLE XI
Kp = I/G

No.	TIC64	TIC62	TIC55	SC140	SC110
F1	3.1	1.5	1.3	2.5	2.5
F2	1.3	1.8	1.2	1.3	1.1
F3	4.3	3.7	1.2	2.3	2.0
F4	3.7	1.3	1.3	1.7	1.7
F5	2.5	6.0	1.2	1.8	1.3
F6	1.3	1.0	1.2	1.9	1.3
F7	1.3	1.0	1.3	1.1	1.0
F8	2.8	3.3	1.5	/	/
AM.	***2.5***	***2.4***	***1.3***	***1.8***	***1.5***
AM / # of issues	***0.31***	***0.31***	***0.63***	***0.30***	***0.52***

TABLE XII
Kc = G/OPs

No.	TIC64	TIC62	TIC55	SC140	SC110
F1	0.7	1.6	0.8	0.4	0.4
F2	1.1	1.1	1.1	0.8	0.8
F3	0.6	0.6	1.0	0.8	0.8
F4	1.8	3.6	0.8	0.6	0.6
F5	0.4	0.3	0.5	0.7	0.9
F6	1.3	2.5	1.0	1.4	2.1
F7	2.8	4.0	2.0	1.6	1.6
F8	2.0	1.0	0.7	/	/
AM	***1.2***	***1.8***	***1.0***	***0.9***	***1.0***

TABLE XIII
COMPILER OPTIMIZATION IN MAJOR INNERMOST LOOPS

No.	Kernel Functions	TIC64	TIC62	TIC55	SC140	SC110
F1	FLT_filterAP_fx	fully unrolling	software pipelining	no software pipelining	software pipelining	software pipelining
F2	LPC_Chebps_fx	software pipelining	software pipelining	no software pipelining	no software pipelining	no software pipelining
F3	LPC_autocorrleation_fx	software pipelining	software pipelining	no software pipelining	software pipelining	software pipelining
F4	FCS_Excit_Enhance	sophisticated software pipelining	poor software pipelining	no software pipelining	no software pipelining	no software pipelining
F5	LSF_Q_New_ML_search	fully unrolling	software pipelining	no software pipelining	software pipelining	software pipelining
F6	c_fft_fx	no software pipelining, few ILP	no software pipelining, few ILP	no software pipelining	no software pipelining	no software pipelining
F7	FCB_add_sub_contrib	no software pipelining	no software pipelining, no ILP	no software pipelining	no software pipelining, no ILP	no software pipelining, no ILP
F8	PIT_LT_Corr_Rmax_fx	software pipelining	software pipelining	no software pipelining	/	/

2. Table XIII summarizes how DSP compilers on optimize the major innermost loops. Provided below is a detailed analysis of how each kernel function's clock cycles and code size are related to the structure of the program, and the features of the DSP processors and compilers. The functions are identified by their actual names in the SMV application.

F1—FLT_filterAP_fx: It is a very short function having a simple 2-level nested loop with many statements between outer and inner loops; its innermost loop contains only two lines and a small loop count. From Table VII, VIII and XIII we note that TIC62 does perform software pipelining on innermost loop and produces long prelude and postlude, which leads it has big clock cycle; TIC64 fully unrolls innermost loop, and has much less clock cycle with slight bigger code size; TIC55's major innermost loop has three instructions only without any optimization. As a result, its code size is small and its execution takes a small number of clock cycles.

F2—LPC_Chebps_fx: It is a simple function having a single *for*-loop with a large loop body of many DSP operations, TIC55 compiler does not perform software pipelining, and therefore it has bigger clock cycles as shown in Table VII.

F3—LPC_autocorrelation_fx: It is a very simple function; its major innermost loop has one C statement only. All processors except TIC55 perform software pipelining on it and achieve high instruction-level parallelism as shown in Table XI. As to TIC55, its compiler generates only a small number of assembly instructions for the function so neither its clock cycles is high nor its code size large as shown in Tables VII and VIII.

F4—FCS_Excit_Enhance: It has three 2-level nested loops plus three separate single level loops. TIC62 performs software pipelining but does so very poorly on the major innermost loop; almost no instruction level parallelism available in the code has been exploited. On the other hand, the compiler of TIC64 effectively performs software pipelining on all innermost loops, thus reducing clock cycles of the major innermost loop by nearly two third at a small

expense of increasing the code size, as shown in Tables VII and VIII.

F5—LSF_Q_New_ML_search: The function is rather complicated as it contains a 3-level nested loop. The loop count of its major innermost loop is 10; however its second level loop has a much larger loop count. TIC62 performs software pipelining on its innermost loop, even though the software pipelined loop body is pretty compacted, the overall speed up is offset by the long prelude and postlude. TIC64 unrolls the major innermost loop fully; although the innermost loop body is less compacted than TIC62, both the number of clock cycles and code size are much better than TIC62 as shown in Tables VII, VIII and IX.

F6—c_fft_fx: It is a bit more complicated than the usual FFT program with a moderate CC* value, none of DSP compilers perform software pipelining on innermost loop of this function. Comparing with TIC62, TIC64 has less clock cycle because it has twice as many registers as the other and it has better scheduling to avoid load latency. However, the performance of both DSPs is far behind the theoretical result reported in [31]. On the other hand, as shown in Tables X and XI, TIC55 performs well and has small number of clock cycles because both of its Ks and Kp are small due to its more powerful specialized instructions.

F7—FCB_add_sub_contrib: It has a single level loop with nested conditional branches that restrict the instruction-level parallelism; none of the compilers performs software pipelining. Therefore, with its more powerful specialized instructions, TIC55 performs better in terms of both speed and code size; this is shown in Tables VII and VIII.

F8—PIT_LT_Corr_Rmax_fx: This function has a 3-level nested loop. The innermost loop is very simple with one MAC operation only. However, the second level loop is quite complicated. Even though TIC64 uses a double bandwidth memory fetch instruction, the length of its software pipelined major innermost loop is 5, which is much larger than 2 of TIC55's. This is due to the specialized powerful MAC instruction capable of performing two memory fetch and the MAC instruction simultaneously. Consequently, TIC55 has the best results both in clock cycles and code size as shown in Tables VII and VIII, respectively.

3.Table XIV lists the average DSP operations in the innermost loops of the SMV application [25] and our eight kernel functions, which shows that the total number of DSP operations in innermost loops is limited. Table XV presents the maximum number of instructions in instruction groups. Based on our experimental data, the instruction level parallelism represented by the average Kp = I/G and the maximum number of instructions in instruction groups for the eight kernel functions is far below the number of issues in all VLIW DSP processors. This is shown in Tables XI and XV.

Table XIV also shows that memory fetch operation takes large portion and multiplication operations take small portion of the total number of DSP operations. It helps explain why some DSP processors (such as TIC64, SC140 and SC110) with double memory fetch bandwidth have better performance than TIC62.

TABLE XIV
AVERAGE NUMBER OF DSP OPERATIONS IN INNERMOST LOOPS

	Major functions in SMV encoder	Kernels
ALU	2.1	2.8
Mult	0.8	1.2
Memory	3.1	3.7
Total	***6***	***7.7***

TABLE XV
MAXIMUM NUMBER OF INSTRUCTIONS
IN INSTRUCTION GROUPS

No.	TIC64	TIC62	TIC55	SC140	SC110
F1	5	2	2	3	3
F2	3	5	2	4	2
F3	5	6	2	4	3
F4	6	2	2	3	3
F5	4	7	2	4	2
F6	3	2	2	4	2
F7	4	1	2	2	1
F8	3	5	2	/	/
AM	***4***	***4***	***2***	***3***	***2***
$\frac{AM}{\#\ of\ issues}$	***0.52***	***0.47***	***1.00***	***0.57***	***0.76***

V. CONCLUSION

1. Our experiment with the new SMV benchmark shows that, in comparison with the VLIW type, there are certain advantages of the traditional DSP processor architecture. Although we did not consider the power consumption, which has much less code size and the speed performance is also comparable with VLIW DSP architecture. We suggest that VLIW DSP processors should have more balanced ratio of various kinds of function units and keep some powerful instructions under the support of hardware.
2. SMV benchmark has some kernel functions which are quite different from traditional DSP kernels; however their major innermost loops are relatively short. It implies either too many function units are unnecessary or more sophisticated loop optimization approaches are needed. For example the optimal loop unrolling combining with software pipelining.
3.More sophisticated software pipelining methods for complicated loops such as FFT and with nested conditional branches are needed.

ACKNOWLEDGEMENT

This research was supported in part by the ART (Assigned Release Time for Research) program, Office of the Provost, William Paterson University. E-W. Hu, C. S. Ku, and B. Su would like to thank the ART awards of William Paterson University. This research was also supported in part by grants from the Center for Research, College of Science and Health, William Paterson University.

REFERENCES

[1] W. Strauss, Forward Concepts' *Press 55*, www.fwdconcepts.com, April 2007
[2] D. Katz and R. Gentile, How to Choose an Embedded Media Processor, DSP Design Line April, 10, 2007
[3] N. Dutt and K. Choi, Configurable Processors for Embedded Computing, *IEEE Computer*, Jan. 2003
[4] E. Tan and W. Heinzelman, DSP architectures: past, present and futures, ACM SIGARCH Computer Architecture News Vol. 31, Issue 3, 2003
[5] C. Kozyrakis and D. Patterson, Vector vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks, *Proc. of MICRO-35*, 2002
[6] The BDTImark2000™: A Summary Measure of DSP Speed, www.bdti.com, Sept. 2004
[7] EEMBC Brings Embedded Benchmarking out of the Pits, 2000, www.eembc.org
[8] C. Lee et al., MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems, Proc. Of *MICRO-30*, 1997
[9] M. Guthaus, etc., MiBench: A free, commercially representative embedded benchmark suite, *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, December 2001
[10] E. Hu et al, New DSP Benchmark based on Selectable Mode Vocoder (SMV), *Proc. of the 2006 International Conference on Computer Design*, June 2006
[11] CDMA Enhancements Build on a Strong Foundation, www.cdg.org, 2003
[12] M. Genutis, E. Kazanavièius, and O.Olsen, Benchmarking in DSP, ISSN 1392-2114 ULTRAGARSAS, Nr.2(39). 2001.
[13] Code Optimization for TI C62xx / C64xx, CHRONIX tutorial, www.chronix.co.jp/chronix/syouhin/visioncomponents/pdf/Code_Optimization.pdf, 2005
[14] M. Chalamalasetti, Selectable Mode Vocoder (SMV), www.bsnl.in, Feb. 2003
[15] W. Strauss, Forward Concepts' *DSP Market Bulletin*, www.fwdconcepts.com, Jan. 2008
[16] D. Talla et al, Evaluating Signal Processing and Multimedia Applications on SIMD, VLIW, and Superscalar Architectures., Proc. Of *ICCD'00*, 2000
[17] J. Fisher etc., Moving from Embedded Systems to Embedded Computing, Keynote addressing, *CASES03*, 2003
[18] www.3gpp2.org,
[19] L. Codrescu and E. Plondke, A Characterization of Branch Behavior in DSP Application, Proc. Of the International Signal Processing Conference (ISPC03), 2003
[20] E. Fernandes and V. Barbosa, Monitoring the Structure and Behavior of Programs, Proc. of MPCS'02, April, 2002
[21] M. Smith, Overcoming the Challenges to Feedback-Directed Optimization, *Proc. of the ACM SIGPLAN Workshop on Dynamic and Adaptive Compilation and Optimization (Dynamo'00), 2000.*
[22] S. Jinturkar etc., Profile Directed Compilation in DSP Applications, Proc. of *the International Conference on Signal Processing Applications and Technology (ICSPAT'98, 1998)*
[23] D. Wall, Limits of Instruction-Level Parallelism, Proc. of ASPLOS-IV, 1991.
[24] S. Graham etc., gprof: A Call Graph Executin Profiler. Proc. of *SIGPLAN notices*, Vol. 17, No.6, 1982.
[25] B. Su et al., Analysis of Loop Behavior of Selectable Mode Vocoder (SMV) and Its Impact of Instruction Level Parallelism, *Proc. of GSPx 2005*.
[26] T. McCabe, A Complexity Meqsure, *IEEE Tran. On Software Engineering,* 2(4):308-320, 1976
[27] Software Engineering Institute, Cyclomatic Complexity, Software Technology Roadmap, Carnegie Mellon University, http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.htm, 2005
[28] S. Ahmadi, Tutorial on the Variable-Rate Multimode Wideband Speech Codec, CommsDesign, Sept. 2, 2003
[29] B. Su et al, Software De-Pipelining Technique, *Proc. Fourth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM2004)*, 2004
[30] B. Su et al, A new Source-Level Benchmarking for DSP Processors, Proc. of *the International Conference on Signal Processing Applications and Technology (ICSPAT'03)* 2003.
[31] J. Sankaran et al, Optimized implementation of the FFT algorithm on the TMS320C62x and the TMS320C64x DS, *Proc. of the 3rd Workshop on Optimizations for DSP andEmbedded Systems (ODES-3)*, March 20, 2005