ISSN 1392-2114 ULTRAGARSAS, Nr.2(39). 2001.

Benchmarking in DSP

M. Genutis, E. Kazanavičius, O.Olsen

Kaunas University of Technology,

*DSP Laborotatory*Communication Department, Aalborg University*

Introduction

Benchmarking is a way to measure performance of a computer system. More specifically, *benchmark* is a program used to quantitatively evaluate computer hardware and software resources. To get a better picture of a computer system, engineers define *benchmark suites* - sets of benchmarks. By choosing a suitable benchmark for a system it is possible to test if it behaves the way we expect. In this paper we will discuss what has been done in benchmarking the DSP systems.

A. What Makes DSP Processors Tick?

DSP applications imply differences between DSP and general-purpose processors. Most DSP applications require high performance in repetitive, computation- and data-intensive tasks. Thus, the following architecture features prevail in DSP processors:

1. *Fast Multiply-Accumulate*. The multiply-accumulate operation is useful in algorithms that involve computing a vector product, such as digital filters, correlation, and Fourier transforms. The multiply-accumulate operation (MAC) is usually performed in a single instruction cycle. To achieve this functionality, DSP processors include one or more multipliers and accumulators integrated into the main arithmetic processing unit.
2. *Multiple-Access Memory Architecture*. It is the ability to complete several accesses to memory in a single instruction cycle. This allows the processor to fetch an instruction while simultaneously fetching operands for a previous instruction or store the result of the previous instruction to memory.
3. *Specialized Addressing Modes*. Efficient handling of data arrays and other common data types is provided through dedicated address generation units. Special addressing modes called *circular* or *modulo* addressing are often supported to simplify the use of data buffers.
4. *Specialized Execution Control*. Because DSP algorithms involve repetitive computations in small loops, most DSP processors provide special support for efficient looping (zero overhead looping).
5. *Peripherals and Input/Output Control*. On-chip peripherals, like analog-to-digital converters, allow for small, low-cost system designs. Similarly, I/O interfaces tailored for common peripherals allow simple interfaces to off-chip I/O devices.

B. Why to Benchmark DSPs?

Why should we benchmark DSPs? First, users need accurate comparisons of DSP processors. However, as architectures diversify, it becomes more difficult to compare them. Simple MIPS, MOPS and other metrics as we will see further, no longer help. Second, benchmarks help users answer the following question: does a particular DSP platform is suitable for user's application? Users need to know, how fast is the processor, is it suitable for real-time tasks, how much power it consumes, how easy it is to write and maintain software, what is the memory usage.

Benchmarking methodologies

In the past benchmarking was performed by DSP processor vendors such as Texas Instruments, Motorola, and Analog Devices. Programs chosen to benchmark were executed on the vendor's platform and execution time was measured. Later independent benchmarks appeared, such as BDTIMark, EDN Benchmarks, DSPStone.

A DSP system consists of a processor, a compiler and a DSP application. Thus, we can distinguish the following components that can be benchmarked:

1. Processor
2. Compiler
3. Platform (Processor and Compiler)

C. Benchmarking the Processor

Since we're benchmarking the processor alone, we cannot use the compiler (if we use compiler-generated code, we’re measuring compiler performance too). The benchmark must be written in assembly language.

D. Benchmarking the Compiler

Benchmarking DSP processors involves benchmarking the compiler as well. Compiler converts high-level language source code into assembly code. Thus, performance of the benchmark depends upon the quality of the compiler. Unfortunately, it's still a lot to be expected from DSP compilers. DSP developers tend to write a lot of assembly code. Thus, DSP software tends to be non-portable and hardly maintainable.

To evaluate compiler efficiency, a reference method was suggested[13]. According to that method, a benchmark program is compiled. Cycle count and memory usage are measured. Then the compiler-generated assembly code is

compared with hand-made code. This comparison tells us how effective the compiler is in using the hardware.

E. Benchmarking the Platform

Platform benchmarks are written in a high level programming language and measure the performance of both the compiler and processor.

F. Benchmarking Approaches

The following approaches are used to benchmark DSPs:

1. Metrics (MIPS, MOPS, MFLOPS, etc)
2. Complete DSP Applications (v.90 modem, GSM-EFR transcoder, Viterbi encoder/decoder)
3. DSP algorithm kernels (FIR filter, FFT)

Let's look at the pros and cons of each of the above approaches.

1) Metrics

MIPS and MFLOPS are frequently used, but are they meaningful? Metrics approach is widely criticized in literature [2], [5], [10], [13]. Metrics lost significance when RISC architectures were introduced. It's not worth counting instructions executed during a period of time since different processors accomplish different amount of job with a single instruction. Metrics followers should precisely define what they mean by *instruction* or *operation*. Metrics cover quantitative issues and they don't evaluate architectural features of processors.

2) Complete DSP Applications

These are real-world working DSP applications such as v.90 modem, GSM-EFR transcoder, Viterbi encoder/decoder. Usually they consist of several thousands lines of C source code. They require assembly hand optimizations. It's expensive to create such a benchmark - it consumes a lot of time and efforts. Such a benchmark measures whole system, not only the processor. Since the application consumes a lot of program memory, memory system and peripherals are tested as well.

3) DSP Algorithm Kernels

They are code fragments extracted from real DSP programs. Kernels are believed to be responsible for most of the execution time. They have small code size and long execution time. They consist of small loops which perform number crunching, bit processing etc. A few examples of kernels:

1. Matrix product
2. Convolution
3. FIR, IIR, LMS filters
4. FFT

G. What to Measure?

The following parameters are usually measured when benchmarking DSPs:

1. Cycle Count
2. Program Memory Usage
3. Data Memory Usage
4. Program execution time
5. Power consumption
6. Cache hit/miss ratio (if the cache exists)

They are sufficient to compare DSP processors from the user's point of view. If the user needs speed, then cycle count and program execution time matters. If user programs are large and access memory a lot, then program and data memory usage must be taken into account. Power consumption is important in small widgets that incorporate DSP chips. If the system has hard real-time constraints, then cache hit/miss ratio measurements are relevant.

H. How to Measure: Use of Geometric Mean

Suppose initially that we want to average 5.2, 6.3, and 4.7. The arithmetic mean is (5.2 + 6.3 + 4.7) / 3 = 5.4. The geometric mean is $\sqrt[3]{5.2*6.3*4.7} = \sqrt[3]{153.97} = 5.36$.

Now suppose some dubious optimizations are performed on one of the tests, so that the third result becomes, say 19.7. The arithmetic mean is now (5.2 + 6.3 + 19.7) / 3 = 10.4. However, the geometric mean becomes $\sqrt[3]{5.2*6.3*19.7} = \sqrt[3]{645.37} = 8.64$. Using the arithmetic claim, we can claim to have almost doubled performance from 5.36 to 10.4, but what we really did was drastically improve the result for just one test and leave the rest unchanged.

Thus, if we want the final figure to reflect overall improvement, the geometric mean is a better measure; it's less sensitive to changes in just one component of the results. This fact makes the geometric mean useful for combining the results of several benchmarks.

On the road to deliver unbiased benchmarks

About 13 years ago we didn't have DSP benchmarking authorities. General purpose benchmarks were used to benchmark DSPs. Companies had to manipulate benchmark results to get on top. DSP world needed unbiased and meaningful benchmarks. Because of the lack of other benchmarks, designers, vendors and analysts relied on Dhrystone MIPS for a microprocessor's performance results. MIPS is a synthetic and extremely abused benchmark. When a vendor presents it's processor's MIPS, you can always assume that the vendor performed the test under the best conditions for their particular processor.

We must face the fact that it's difficult to design a benchmark for a processor in the context of an embedded application. The difficulty stems from the fact that one benchmark cannot effectively represent the variety of embedded applications.

In 1988 EDN standardized some common benchmarks, compiled and analyzed the resulting data on 18 DSPs and 12 benchmarks. Benchmarks were implemented in assembly language and were a good start in benchmarking DSP processors. This came as a result of 1981 EDN benchmark survey.

In 1993 university DSPStone project came up with valuable results on evaluating DSP compiler performance. C compiler generated code was compared with hand-coded

assembly code. Speed and memory usage overhead were measured. Results showed that DSP compilers are still in an infant stage - DSP developers prefer writing applications in assembly.

In 1997 EDN Embedded Microprocessor Benchmark Consortium (EEMBC) was established. Its primary goal was to develop real-world benchmarks with precise rules for reporting. Unbiased benchmark tests are EEMBC's mission. 21 competing microprocessor companies took part in establishing EEMBC (now there are 28). Tool vendors and developers are excluded from EEMBC membership.

At about the same time as EEMBC, BDTIMark benchmark suite appeared. Its main purpose is to measure DSP processor execution speed. That's why their benchmarks are written in assembly language.

Dhrystone benchmarks

In 1984, Dhrystone benchmark authors wrote the initial 1.0 version in Ada and have subsequently updated the benchmark. Later the benchmark was converted to C. The most recent 2.1 version is free and can be downloaded via the Internet. Dhrystone program performs no directly useful action. Instead, it belongs to synthetic benchmarks -- code with particular behavioral characteristics rather than programs that implement algorithms.

Version 2.1 of Dhrystone contains 103 high-level statements within the main loop, which executes repeatedly during a benchmark run. User chooses the number of iterations at runtime. At the end of the benchmark run, Dhrystone prints the absolute time required per iteration; the number of iterations per second through the main loop; and the performance, measured in iterations per second, relative to a baseline machine. The baseline machine is DEC VAX 11/780, which was in wide use when authors created the benchmark.

The first major limitation of Dhrystone is its size: because it is small, it's easy to work with. When authors created Dhrystone, caches for instructions or data were a rarity in embedded system design. Since then, caches have become commonplace. Dhrystone's strong locality allows caching to significantly boost performance on the benchmark. Even a small cache can contain most or all of the information that each iteration uses.

The second major limitation of Dhrystone arises from its execution profile, which is the proportion of overall execution time it spends in each function. A program in which each function makes roughly the same contribution to the total has a ``flat'' profile. In contrast, a program for which just a few functions account for a significant proportion of overall execution time has a ``sharp'' profile.

On most embedded CPU architectures, Dhrystone's profile is sharp, and it spends as much as 30 to 40% of its execution time in just two functions: strcpy and strcmp. Dhrystone performs rather specialized and more intense string handling than that found in many embedded system workloads. If user's embedded firmware does little specialized sting handling, Dhrystone results could be inaccurate.

Various other criticisms of Dhrystone include the argument that synthetic nature doesn't resemble real-world DSP applications. For example, Dhrystone has an unusually low number of instructions per procedure call and is therefore oversensitive to the implementation efficiency of procedure calls and returns.

Similarly, Dhrystone's call sequences are nested only three or four levels deep. Thus, most register-window RISC machines would never spill or fill their register windows and thus would never need to save or restore registers from off-chip memory.

All these lines of argument essentially proceed towards the same conclusion: the small and portable Dhrystone benchmark doesn't use the CPU in the same way as a large, complex piece of embedded software does.

EDN's DSP benchmarks

This suite came out in 1988. 12 common benchmarks were run on 18 DSPs. DSP users were surveyed about their applications and DSP manufacturers were asked about their own benchmarks. Benchmark suite resulted in a collection of kernel benchmarks, fragments of code taken from real-world applications: FIR, IIR filters, FFT, dot product, and matrix multiplications.

EDN specifically defined the execution time and memory for the participating manufacturers as follows [11]:

"The execution time is the time it takes to execute the given benchmark. This does not include the time it takes to initialize the system or create lookup tables. It does include the time to initialize those items (registers, pointers, etc) that are required each time the routine is run."

"The total memory required is the total number of words that the program requires. This includes executable code, initialization code, filter coefficients, delay data, twiddle factors, bit-reversal lookup tables, and any other item that consumes some of the available memory resources of the DSP."

One of conclusions was that speed isn't the most important factor. If DSP is used mostly for filter applications, the speed at which a DSP can execute FFT is probably irrelevant. There were no winners in this survey, however, DSP evaluation process was enlightened.

EEMBC benchmarks

EEMBC was established in 1997. The benchmarks consist of suites of tests written in C. They encompass applications in the automotive/industrial, consumer, networking, office-automation, and telecommunication industries. Within each suite, individual tests measure one or more processor functions, allowing to determine which functions are appropriate for the application. For each test, vendors must report runtime characteristics that include compiler versions and switches, processor clock and bus speed, wait states and cache size. Furthermore, the vendor

must clearly document any code changes that were implemented to improve the benchmark performance; this documentation ensures that the exact test is repeatable and unbiased.

EEMBC strategy is to extract processor-intensive code segments of the most popular embedded applications, i.e. DSP kernel benchmarks are used. EEMBC benchmarks lie in between synthetic and real world based benchmarks.

If the code of the benchmark fits into the cache, it doesn't test the processor's external bus structure. To resolve this drawback, EEMBC studied how its benchmark programs behave within the context of the application. In some cases, an algorithm apparently runs repeatedly, but not continuously, during the operation of the application. Therefore, depending on a processor's cache size or on whether you use cache locking, the processor would repeatedly have to reload the algorithm into its cache. When appropriate, EEMBC wrapped "cache-thrashing"' code around the benchmark algorithm to simulate the real-world effect and exercise a processor's bus interface unit as it loads code and data from external memory into the on-chip cache.

BDTIMARK – a DSP benchmark suite

BDTI is a consulting company that specializes in independent analysis of DSP technology. Their benchmark suite compares the speed of DSP processors. Benchmarks are written in assembly language to avoid compiler interference in benchmark results.

Writing a complete DSP application in assembly is time consuming. That's why BDTI adopted an algorithm kernel model. BDTI benchmarks comprise 11 DSP algorithm kernels that have been selected from those most commonly used, including FIR and IIR filters, an LMS filter, a convolutional encoder, and an FFT. Benchmark algorithms are programmed in assembly and carefully hand-optimized to make full use of processor's potential.

The BDTIMark is an overall speed metric that distills processor's execution time results on all 11 BDTI benchmarks into a single number. However, this approach hides the relative importance of each routine for a particular application. BDTI correctly points out that proper use depends on profiling an application to assign appropriate weights. However, the single number that they publish simply distills results using uniform weighting.

Benchmark routines rely on the chip's native data format. That is, a 16-bit fixed-point chip could end up with the same, or better, ranking than a much higher-end floating-point chip.

Performance in DSP applications is very important, but reality dictates other concerns such as chip and system cost, power consumption, memory usage and so forth. BDTIMark doesn't take this into account. BDTI results show that general-purpose processors can work as fast as DSP processors. But why to use DSPs then? Thus, benchmark results alone obscure practical reasons for choosing a DSP processor.

DSPSTONE benchmarking methodology

The purpose of the DSPStone project was to benchmark DSP compiler performance and efficiency. If DSP application developers decide to write software in C, they want to know, what will be the speed/memory overhead. DSPStone results can be used to find reasons in relative inefficiency of DSP compilers as well as possible ways to improve their performance.

DSPStone covers over 30 benchmark programs, organized into three benchmark suites (application, DSP-kernel and C-kernel). In order to measure results, a new, DSP-oriented benchmarking methodology is introduced [4]. It is based on the reference-code method where the metric distance between the hand-written assembly code and the assembly code generated by the compiler is measured.

The introduced methodology is applied on a set of five commercial DSP C compilers (Analog Devices 21xx, AT\&T 16xx, Motorola 56xxx, NEC 770xx and TI 320C5x).

The results show that a lot of work has to be invested into DSP compiler development in order to make them useful, not only for rapid prototyping, but also for production quality programming.

Conclusions

Users want to know if a particular processor is suitable for their application. Also, it is good to know which processors are the best up to day. Benchmarking these processors is a way to do it. Since DSP processors differ from general-purpose counterparts, it is good to have a DSP-oriented benchmarking methodology.

DSP benchmarks usually are one of the following: complete DSP applications, DSP algorithm kernels or synthetic code segments. They are written either in C or assembly, or both of them. Writing in assembly takes more time and efforts. The advantage is that compiler influence is excluded.

Parameters usually measured are: execution time, memory usage, cycle count, power consumption. Often benchmark results are summarized as a single number (for example, BDTIMark).

DSP user community agrees that MIPS or Dhrystone benchmarks are no longer a reliable source to judge processors. DSP user community needs an independent DSP benchmarking authority. Now we have at least two of them: EEMBC and BDTI. Their reports can be relied on when choosing a DSP processor.

This work was done Aalborg University.

References

1. **Cantrell Tom**. Yet Another Benchmark? Embedded Systems, 2000.
2. **Thomas M.** Comte and Wen mei W. Hwu. Benchmark Characterization. IEEE Computer. 1991. P. 48-56.
3. **Eyre Jennifer.** DSP Benchmarking Methodologies. Computer Design, March 1998.

4. Institute for Integrated Systems in Signal Processing. DSPStone: A DSP-Oriented Benchmarking Methodology. Technical report, Aachen University of Technology, 1994.
5. **Lapsley Phil, Bier Jeff, Shoham, Amit and Lee Edward A.** DSP Processor Fundamentals: Architectures and Features. IEEE Press, 1996.
6. **Levy Marcus.** EEMBC: on the Road to Delivering Unbiased Benchmarks. EDN Magazine. December 1997.
7. **Levy Marcus**. At Last: Benchmarks You Can Believe. EDN Magazine. May 1998.
8. **Levy Marcus.** A Peak Behind the Scenes at EEMBC. EDN Magazine. May 1999.
9. **Mann Daniel**. When Dhrystone Leaves You High and Dry. EDN Magazine. May 1998.
10. W**alter J.** Price. A Benchmark Tutorial. IEEE MICRO. 1989. P. 28-41.
11. **Shear David.** EDN's DSP Benchmarks. EDN. September 1988. P.126-134.
12. **Weicker Reinhold P.** An Overview of Common Benchmarks. IEEE. 1990. P. 65-75.
13. **Zivojnovic Vojin, Velarde Juan Martinez, Schlager Christian and Meyr Heinrich**. DSPStone: A DSP-Oriented Benchmarking Methodology. In Proceedings of International Conference on Signal Processing: Applications and Technology '94 – Dallas, 1994.

M. Genutis, E. Kazanavičius, O. Olsen

DSP efektyvumo matavimai

Reziumė

Darbo tikslas – apžvelgti įvairius diskretinių signalų procesorių (DSP) efektyvumo nustatymo metodus.

Darbe aptariami DSP efektyvumo kriterijai, metodų skirtumai, parodyta matavimų svarba, realizuojant DSP algoritmus įvairiuose procesoriuose.

Darbas atliktas bendradarbiaujant su Danijos Aalborgo universitetu.

Pateikta spaudai 2001 04 23