How to estimate DSP processor performance

Phil Lapsley and Garrick Blalock
Berkeley Design Technology Inc.

Digital signal processing (DSP) is the application of mathematical operations to digitally represented signals. Because digital signals can be processed by cost-effective digital integrated circuits, DSP systems can economically accomplish complex tasks, such as speech synthesis and recognition, that would be difficult or impossible with conventional analog techniques.

The market for products based on DSP technology—wireless communication devices and PC multimedia peripherals, for example—is growing rapidly. Semiconductor makers have responded to this demand with a bewildering array of DSP processors. Selecting the best one for a given application presents a difficult and time-consuming challenge for DSP system designers.

Simple, familiar performance measures like MIPS (millions of instructions per second) and MOPS (millions of operations per second) are misleading and neglect factors like memory usage, power consumption, and application execution time. Complex alternatives—such as application benchmarks—suffer from limitations that virtually preclude fair comparisons.

Fortunately, a compromise methodology that combines algorithm kernel benchmarking with application profiling yields good estimates of processor performance weighted to the target application.

Advantages of DSP

The key advantage of DSP systems over those based on analog circuits is that the former benefit from the rapid advances being made in digital IC manufacturing processes; increasing density and speed have brought enormous performance gains as well as cost and size reductions. As a result, DSP ICs can combine digital control functions, such as user interfaces, with signal-processing circuitry. Also, they can interface with a wide selection of peripherals and other system components. Flexibility is another plus: unlike analog systems, which usually require hardware modificatons to effect changes in functionality, many systems based on DSP processors may be quickly reprogrammed in response to changing requirements.

DSP systems enjoy two further outstanding advantages over analog systems. They are far less sensitive to environmental conditions such as temperature, and they are insensitive to component tolerances. Barring catastrophic failure, they will behave the same in the snow as in the desert, environments that might upset analog systems. And properly functioning DSP systems will always produce the same output from the same (digital) inputs. In contrast, analog components are guaranteed to be accurate within a stated tolerance only, so two identically manufactured analog systems will respond slightly differently to the same inputs. These two advantages make DSP systems predictable, repeatable, and well suited for high-volume manufacturing.

DSP was once limited to high-end, cost-insensitive applications such as military sonar and medical imaging systems. But now, the cost-performance ratio of DSP processors makes digital signal processing the best solution for a world of consumer products and other cost-sensitive applications. DSP solutions today are ubiquitous in applications such as digital cellular phones, modems, disk drive servo control, PC multimedia, and home theater.

DSP processors

Because most signal-processing systems perform complicated mathematical operations on real-time signals, DSP processors are specially architected to accelerate the execution of repetitive, numerically intensive calculations. Common elements include:

- Circuitry to perform multiply-accumulate operations, useful in algorithms such as filtering.
- Multiple-access memory, which enables the processor to load multiple operands, such as a data sample and a filter coefficient, simultaneously and in parallel with an instruction.
- Various special memory-addressing modes and program-flow control features designed to speed the execution of repetitive operations.
- Special on-chip peripherals or I/O interfaces that enable the processor to interface efficiently with other system components, such as analog-to-digital converters and memory.

The focus here is on evaluating the performance of programmable DSP processors, such as Texas Instruments' TMS320C50 and Motorola's DSP56002. The evaluation methodology to be described may also be applied to determine the DSP performance of general-purpose processors.

Several manufacturers have enhanced the signal-processing abilities of their general-purpose processors by adding instructions and hardware accelerators. For example, Intel Corp. has developed the MMX extensions for the Pentium processor and Integrated Device Technology has added basic DSP functionality to its R4000 line of RISC processors. In the embedded systems arena, many microcontroller and embedded processor manufacturers are also adding DSP functionality.

0018-9235/96/$5.00©1996 IEEE

What is DSP processor performance?

Engineers frequently cite performance as their overriding concern when choosing a processor for a new DSP system design. After all, their choice must meet the number-crunching demands of real-time applications like digital cellular and PC multimedia. Yet tight constraints on the pricing of consumer products rule out paying for any unneeded performance.

These conflicting requirements often lead designers to search for the lowest-cost DSP processor that performs well enough for the application. That means quantifying the computational needs of an application and identifying DSP processors that satisfy them—a truly challenging task. Berkeley Design Technology, which specializes in the analysis of DSP technology, has developed a two-fold methodology of algorithm kernel benchmarking and application profiling.

DSP processor performance can be measured in many ways. The most common metric is the time required to do a certain amount of processing. In some applications, however, concerns such as memory usage or power consumption may be equally—or even more—important. An ideal technique for measuring performance would yield data on execution time, memory usage, and power consumption. In what follows, execution time will serve as the primary measure of performance, with memory usage and power consumption as secondary considerations.

Functions used for Berkeley Design Technology benchmarks

Function	Description	Application examples
Real block finite impulse response (FIR) filter	FIR filter that operates on a block of real (not complex) data	G.728 speech encoding, other speech processing
Complex block FIR filter	FIR filter that operates on a block of complex data	Modem channel equalization
Real single-sample FIR filter	FIR filter that operates on a single sample of real data	Speech processing, general filtering
Least-mean-square adaptive FIR filter	Least-mean-square adaptive FIR filter that operates on a single sample of real data	Channel equalization, servo control, linear predictive encoding
Infinite impulse response (IIR) filter	IIR filter that operates on a single sample of real data	Audio processing, general filtering
Vector dot product	Sum of the pointwise multiplication of two vectors	Convolution, correlation, matrix multiplication, multidimensional signal processing
Vector add	Pointwise addition of two vectors producing a third vector	Graphics, combining audio signals or images, vector search
Vector maximum	Discovery of the value and location of a vector's maximum value	Error-control coding, algorithms using block floating-point arithmetic
Convolutional encoder	Application of convolutional forward error-correction code to a block of bits	North American digital cellular telephone equipment (IS-54 standard)
Finite-state machine	A contrived series of control operations (test, branch, push, pop) and bit manipulations	Control operations appear in nearly all digital signal-processing applications
256-point, radix-2, in-place fast Fourier transform (FFT)	FFT conversion of a normal time-doman signal into the frequency domain	Radar, MPEG audio compression, spectral analysis

MPEG = Motion Pictures Experts Group

Customary methods

Traditional approaches to performance measurement use very simple metrics to describe processor performance. The most common performance unit, MIPS, is misleading because the amount of work done by an instruction can vary greatly from one processor to another. DSP processors, in particular, often have highly specialized instruction sets.

In other words, MIPS figures are useful only in the context of a single, known, processor architecture. Outside that context, there would have to be some gauge of instruction set efficiency—perhaps a ratio of the number of instructions the processor executes to the number of instructions other processors in the same class require to do the same work. MOPS suffers from a related problem—what counts as an operation and how many are needed to do useful work vary greatly from processor to processor.

Other oft-quoted performance-measurement units can also mislead. Because multiply-accumulate operations are central to many DSP algorithms, such as filtering, correlation, and vector multiplication, some manufacturers quote performance in multiply-accumulates per second (MACS). Most DSP processors, however, can complete one MAC per instruction cycle, making this unit equivalent to MIPS. Further, MAC measurements disregard the important data movement and processing required before and after multiply-accumulate operations.

Neither MIPS, MOPS, nor MACS can measure secondary performance issues like memory usage and power consumption. This is a severe limitation because execution time means little if memory requirements exceed system design constraints. Further, if large memory usage requires resorting to slower external memory, then the processor's speed may be reduced.

Likewise, in a portable application, a processor is unusable if it consumes more power than the available battery can supply. Power consumption varies with different instructions and data values. Although many manufacturers quote a "typical" power consumption at a given clock rate, such specifications are suspect without details on the precise instructions and data used in the measurement. Furthermore, the measurements do not account for special power-saving modes available when a processor (or a portion of it) is idle.

A common approach to benchmarking computer systems is to use complete applications, or even suites of applications. This approach is used by the Standard Performance Evaluation Corp. in

the popular SPEC95 benchmarks for general-purpose processors and systems. Examples of DSP applications might include speech coders (CELP, VSELP, GSM, and so on), modems (V.34, V.32bis, and so on), disk drive servo control programs, or PC-based multimedia systems. The approach works best in cases where the application software is portable—that is, when the application is coded in a high-level language like C. Unfortunately, because of the inefficiency of C compilers for the most cost-effective (fixed-point) DSP processors, and because of the demand for performance, high-volume DSP applications are largely coded in assembly language. Furthermore, when applications written in C are benchmarked, the compiler as well as the processor is benchmarked.

Suppose application benchmarks are coded in assembly. Even then, four problems remain. First, few applications are well enough defined to permit fair comparisons. For instance, two implementations of a standard modem may use different equalizers, one more complex than the other, depending on whether the goal is a high-quality solution or one that makes the least demands on the processor. Second, with most complex applications, it's virtually impossible to ensure that assembly language software is optimal, or even near optimal; thus, application implementations may be benchmarking the programmer, not the processor. Third, full application benchmarks tend to measure a system's performance, not just the processor's. Without very careful system specifications, isolating the performance of the DSP processor from other system components like external memory and microcontroller coprocessors is very difficult. Last, coding an entire application could take years of engineering time.

Algorithm kernel benchmarking

The twofold methodology of algorithm kernel benchmarking and application profiling is a practical compromise between oversimplified MIPS-type metrics and overly complicated application-based benchmarks. Algorithm kernels are the mathematical building blocks of most signal-processing systems and include functions such as fast Fourier transforms, vector additions, and filters. Algorithm kernels offer several compelling advantages as benchmarks:

- Relevance. Algorithm kernels can be selected by examining DSP applications and focusing on those portions that account for the largest share of the processing time.
- Ease of specification. By virtue of their modest size, algorithm kernels can be well-defined: a specification can state their input and output requirements, include test vectors to verify functional conformance, and indicate which algorithms and optimizations are allowable. For example, there are many techniques for implementing a fast Fourier transform (FFT). Without specifying the exact type of implementation, one cannot fairly compare two processors' FFT execution times.
- Optimization. Again because algorithm kernels are of a moderate size, a skilled programmer can write the code in assembly language and be fairly certain that his or her implementation is optimal, or very close to optimal, on a given processor.
- Ease of implementation. Because of their moderate size, algorithm kernels can be implemented in a reasonable amount of time, even with thorough optimization.

The general-purpose suite of algorithm kernels used in Berkeley Design Technology's DSP processor benchmarking is called the BDT Benchmarks [Table 1]. Although all of these benchmarks were chosen for their relevance to DSP applications, not all of them are traditional DSP functions. For example, the finite-state machine benchmark represents decision-making in control processing, and the convolutional encoder represents error-correction coding. As DSP applications become more complex and highly integrated, DSP processors are increasingly being called upon to execute these types of operations.

With one exception, the benchmarks are optimized for execution time. The exception is the finite-state machine, which is optimized for memory usage—usually of greater concern in control functions. Naturally, these particular algorithm kernels may not be the best choice for every application. Engineers may want to add or delete algorithm kernels to better represent the type of processing performed in their applications.

Measuring algorithm kernel execution

There are several ways to measure a processor's execution time on an algorithm kernel benchmark. Usually the most convenient method is with a cycle-accurate software simulator. Such a simulator models a processor's execution of instructions and keeps an accurate cycle count by making adjustments when factors such as pipeline interlocking or bus contention slow its operation. Software simulators offer a controlled, flexible, and interactive environment for testing and optimizing code. Some include support for macros or scripts that can automate performance measurement and functionality verification so that engineers can quickly see how code changes affect performance.

Hardware-based application development tools can also be used to measure execution time and are needed to gauge power consumption. An example is an emulator, which lets the user download code from a PC to the target processor. Using a debugger, most emulators allow the processor to step through the code line by line, or to run the code until a breakpoint is reached.

To measure power consumption, code can be run in continuous loops on hardware application development tools. Power consumption is measured by isolating the power going to the DSP processor from the power going to other system components, running a benchmark in a repeating loop, and using a current probe to record the time-varying input current under carefully controlled conditions.

Measuring benchmark performance on new processors without software or hardware development tools is a tedious and error-prone process. The time required to execute each instruction in the benchmark must be manually calculated and the benchmarks must be checked manually for functional correctness. Before this can be done, the processor documentation must be thoroughly understood, because pipeline interlocks or bus conflicts can slow execution.

Benchmark results

Figure 1 shows the execution time scores of several processors measured with the BDT FFT benchmark. The FFT is a computationally efficient algorithm for computing the discrete Fourier transform, which converts time-domain signals into their frequency-domain representations. The results illustrate how architectural features can impact a processor's performance.

The Texas Instruments TMS320C80, for example, owes much of its top performance to an arithmetic and logic unit that permits two 16-bit operations per instruction cycle. (Note that the TMS320C80 includes four separate on-chip DSP processors that operate in parallel; here, just one of the four processors is considered.) Similarly, the Analog Devices ADSP-2106x and the Zoran ZR3800x both include special instructions to support the FFT, improving their scores on this benchmark.

Of course, caution is the watchword when interpreting benchmark findings. For example, a processor's data word width affects both numerical accuracy and memory usage, and cannot be ignored when viewing benchmark results. A 24-bit processor may execute a finite-impulse-response filter in the same time as a 16-bit processor, but the benchmark results will show a 50 percent increase in data memory usage. This increased memory use is a result of the extended precision of the 24-bit data. In fact, since the 24-bit processor is calculating the filter result with 50 percent greater precision, more work has clearly been done. If the application needs high precision, the 24-bit processor may be an excellent choice. On the other hand,

if 16-bit precision is sufficient, then the 24-bit processor may be a poor candidate because it consumes more data memory.

Application profiling

The results of algorithm kernel benchmarks are useful but incomplete without an understanding of how the kernels are used in actual applications. A useful methodology for relating algorithm kernels to actual applications is to measure or estimate how often subsections of application code are executed—a technique called application profiling. Profiling can be done at varying levels of granularity, ranging from broad functional subsections to algorithm kernels and even individual instructions. At the algorithm kernel level, profiling looks at how often such kernels are executed when an application is run for a suitable period of time.

There is more than one way to profile at the algorithm kernel level. Code in high-level languages, such as C, is an excellent source of profiling information because algorithm kernels often can be identified as subroutines. If assembly code is available, profiling information may be extracted by running the code on an instruction-set simulator equipped with profiling capabilities, or by setting breakpoints in key sections of code to see how often each is executed. The same information may also be estimated by studying application specifications or block-level signal flow diagrams.

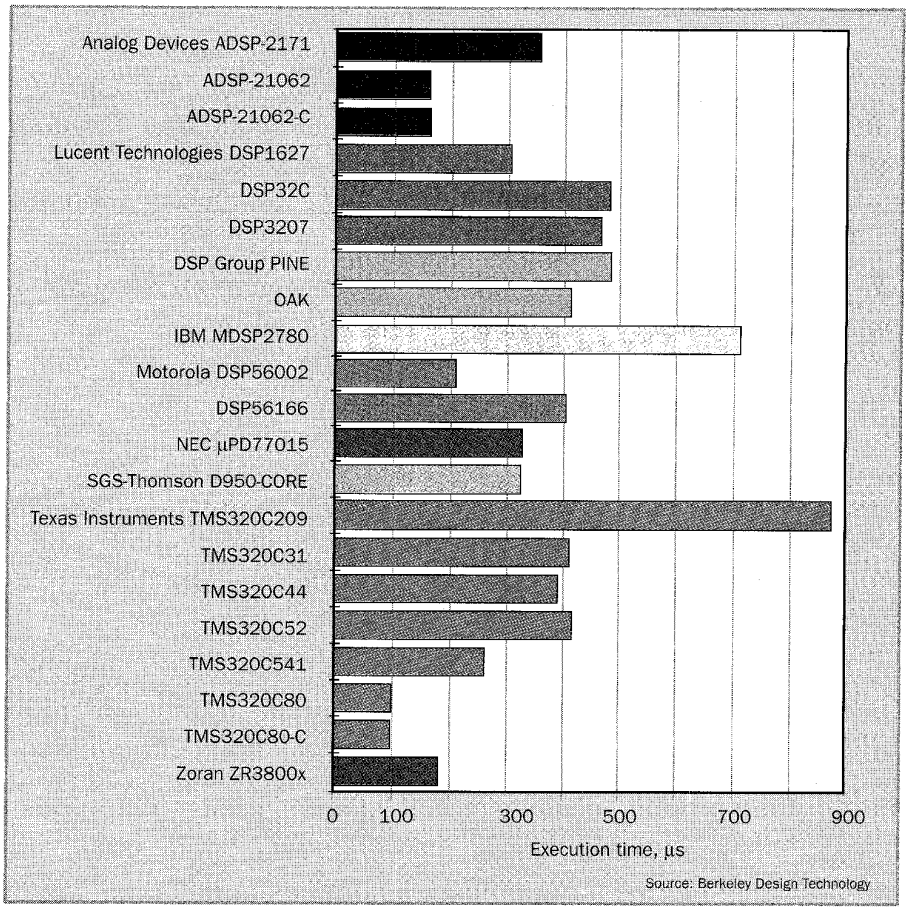
Application profiling allows designers to estimate the relative importance of each algorithm kernel benchmark in a particular application. Of course, it is not a perfect process. If the number of benchmarks is limited to a reasonable number, say 10 or 15, then in many cases there won't be an exact match between every algorithm found in a complex application and a benchmark. Engineers will have to approximate some of the application's processing with benchmarks that perform similar, but not identical, computations.

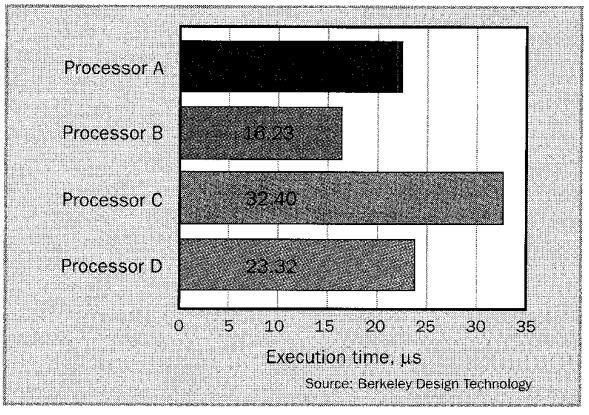
Note, too, that application profiling may not identify some of the optimizations that will be possible when assembly code is written. For example, a programmer may notice that a set of values computed in one algorithm kernel is also used in a later kernel. Reuse of the values may markedly reduce the amount of processing required in the second algorithm kernel.

A processor's performance on an application is estimated by combining the results of the benchmarks with the results of the application profiling. Multiplying the benchmark execution times by the number of occurrences of each benchmark (or a similar algorithm kernel) yields an estimate of the time it would take to execute the application.

A simple example of a 10-band graphic equalizer can be used to illustrate the approach described here. A stream of digitized audio samples enters the graphic equalizer at a known sampling rate. Occasionally, a control word telling the equalizer how much to attenuate each of the 10 frequency bands will also enter the system. Every time the equalizer receives a control word, it checks to see if any of the bandpass attenuation coefficients need to be changed. If they do, the equalizer updates the filter output gain parameters before proceeding with the filtering.

The finite-state machine (FSM) benchmark can be used to represent the decision-making control processing in this application; the



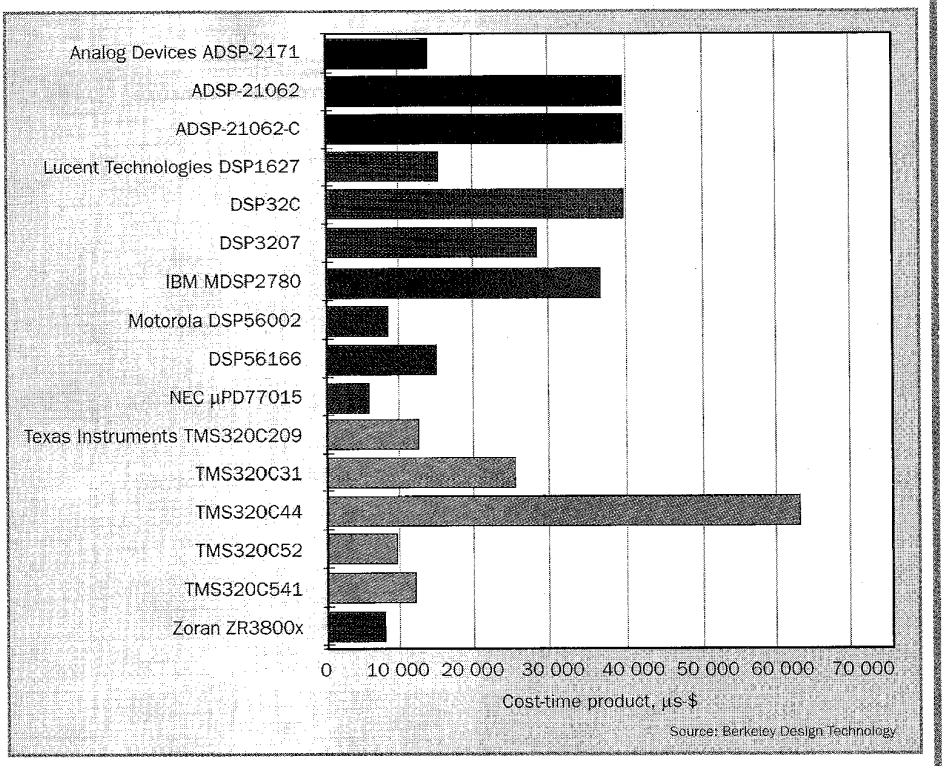


[1] Better paerformance on the Berkeley Design Technology fast Fourier transform benchmark is indicated by lower values of processor execution times. These results are based on the fastest version of each processor available in June 1995. For processors with on-chip cache, the "-C" indicates performance with the cache pre-loaded.

[2] Processor execution time estimates for the 10-band graphic equalizer application. A, B, C, and D represent four commercially available DSPs.

infinite impulse response (IIR) filter benchmark is a good match for the filter processing. Thus, an estimate of each processor's execution time on this application can be obtained by multiplying each processor's execution times on the FSM and IIR benchmarks by weighting factors reflecting the amount of each type of processing required, and then summing these products for each processor.

The results of such a calculation are shown in Fig. 2 for four commercially available DSPs. Each processor's total execution time can then be compared with the execution time available (based on the sampling rate) to determine whether the processor has enough performance for the application.



[3] Better cost-performance on the fast Fourier transform benchmark is indicated by lower values of processor cost-time products. Results here are based on the fastest version of each processor available in June 1995 and on quantity 1000 prices. For processors with on-chip cache, the "-C" indicates performance with the cache pre-loaded.

Other considerations

Although performance is a leading consideration, many other factors affect the choice of a DSP processor. The importance of effective application development tools, for instance, should not be overlooked. Without such tools, writing application software can be difficult, no matter how strong the processor's performance. Likewise, chip vendor and third-party application engineering support can be invaluable when problems arise. Additionally, designers cannot ignore the matter of physical size and must be sure to choose a processor that is available in an appropriate package.

Cost is another critical concern. There are two ways to view the ratio of cost to performance. In some instances, additional performance beyond the minimum required will remain unused. In such cases, designers typically seek the least expensive processor with enough performance to do the job. At other times, the excess performance may allow extra features to be added to the product. Or, the designer may want a line of code-compatible DSP processors with performance levels appropriate for different members of an entire product line. In this situation, a cost–execution-time product metric—the execution time of a processor multiplied by the unit cost—may be useful. The cost–execution-time product of several processors on BDT's FFT benchmark is shown in Fig. 3.

Designers must also remember that minimizing system cost may not always mean minimizing DSP processor cost. For example, one processor may use memory more efficiently than a slightly less expensive rival. If the lower memory usage can eliminate one memory chip from the system, the more expensive processor may yield the lowest overall system cost. Designers must also remember the cost of engineering time and weigh the effect of the quality of application development tools on product development schedules.

DSP systems will grow in sophistication and in their demand for computational performance. At the same time, semiconductor vendors will press ahead with developing more powerful DSP processors and integrating them with other system components, like microcontrollers and general-purpose microprocessors. With more complicated systems and a widening choice of processors, designers will undoubtedly need better estimates of a processor's DSP performance. The methodology outlined above will be an excellent starting place for calculating these estimates. ◆

To probe further

"How RISCy is DSP?" by Michael Smith (*IEEE Micro*, Vol. 12, no. 6, December 1992, p. 10) uses benchmark algorithms to compare reduced-instruction-set computer processors with DSP processors and proposes an optimized architecture for digital signal processing.

"Bringing Benchmarks up to SPEC" (*BYTE*, Vol. 21, no. 3, March, 1996, p. 145) discusses how the SPEC95 benchmarks are used to evaluate general-purpose microprocessors and systems.

"*EDN*'s 1996 DSP-Chip Directory" by Marcus Levy and Anne Coyle (*EDN*, Vol. 41, no. 5, March 1, 1996, p. 40) is a concise overview of commercial DSP processors.

Buyer's Guide to DSP Processors, by Phil Lapsley, Jeff Bier, Amit Shoham, and Edward A. Lee is a 924-page technical report published annually by Berkeley Design Technology, Fremont, Calif. The report discusses DSP benchmarking methodologies in detail and contains extensive benchmarking data for popular DSP processors. Excerpts from this report, as well as a pocket guide to DSP processors, are available on the World Wide Web at http://www.bdti.com.

DSP Processor Fundamentals: Architectures and Features (IEEE Press, 1996) is an introductory textbook in which Phil Lapsley, Jeff Bier, Amit Shoham, and Edward A. Lee discuss DSP processor architectures and features, and their relationship to performance.

About the authors

Phil Lapsley is vice president of Berkeley Design Technology Inc., Fremont, Calif., and is one of the company's founders. He has co-authored several industry reports on DSP technology.

Garrick Blalock, a marketing engineer at the company, writes benchmark code and contributes to BDT's published reports on DSP technology.