Microprocessors vs. DSPs: Fundamentals and Distinctions

Bjorn Hori and Jeff Bier
Berkeley Design Technology, Inc.
+1 510-665-1600
www.BDTI.com

Introduction

The number and variety of products that include some form of digital signal processing has grown dramatically over the last ten years. Digital signal processing has become a key component in many consumer, communications, medical, and industrial products. These products use a variety of hardware approaches to implement the required signal processing, ranging from off-the-shelf general-purpose microprocessors (GPPs) to fixed-function custom integrated circuits (ASICs). Off-the-shelf programmable processors are often the preferred choice because they avoid the need for designing a custom chip and because they can be reprogrammed in the field, allowing product upgrades or fixes. They are often more cost-effective (and less risky) than custom chips, particularly for low- and moderate-volume applications, where the development cost of a custom ASIC may be prohibitive. Digital signal processors (or DSPs) are microprocessors specially designed for signal processing applications. In the 1980's and early 1990's, typical DSPs provided signal processing performance that was significantly better than what was achievable with typical GPPs. Since then, the performance gap between DSPs and GPPs has narrowed; today many GPPs are capable of handling serious digital signal processing applications.

While both GPPs and DSPs provide sufficient processing power for many signal processing applications, each type of processor brings with it important strengths and weaknesses that can have an enormous influence on how effectively the processor meets the needs of a particular application. In this article, we examine the key attributes that distinguish DSP algorithms and applications and show how these attributes have led to the architectural specialization found in DSP processors. We then investigate how GPP and DSP architectures compare with regard to these key DSP algorithm attributes. In addition, we examine the architectural techniques employed in high-performance GPPs and DSPs, which have led to rapid gains in signal processing performance in both classes of processor.

DSP and GPP Architecture Fundamentals

Before we examine the key attributes of digital signal processing algorithms, we'll broadly classify the processors we will be discussing. We've already introduced the terms GPP and DSP, but we would like to refine our definitions of these two classes of processor to include both "basic" and "high-performance" variants of each class.

The most common GPPs are microcontrollers (or MCUs). Microcontrollers are designed for cost sensitive embedded applications and are available in 4-, 8-, 16-, and 32-bit varieties. As a group, MCUs account for the vast majority of all microprocessors sold. Our focus will be on 32-bit

MCUs, such as ARM's ARM7 architecture. We will refer to this type of MCU as a "basic GPP." The highest performance GPPs are central processing units (or CPUs) designed for personal computers, work stations, network servers, and the like, such as Intel's Pentium family and Motorola's PowerPC family. We will refer to this class of GPP as a "high-performance GPP."

"Basic DSPs" are architecturally similar to the first DSPs developed in the 1980s. Basic DSPs feature highly specialized architectures designed to perform very specific signal processing tasks efficiently. Akin to MCUs, basic DSPs account for majority of DSPs sold today. Their combination of low cost, good energy efficiency, and in many cases adequate signal processing performance makes them the ideal choice for a broad range of consumer products. Texas Instruments' TMS320C54x family of DSP processors typifies the basic DSP. "High-performance DSPs are generally used in relatively cost-insensitive applications where very high signal processing performance is required, such as medical equipment, military applications, and communications infrastructure. Analog Devices' TigerSHARC typifies the high-performance DSP.

Basic DSPs and basic GPPs are in many respects quite similar; for example, they typically execute a single instruction per clock cycle, have limited amounts of on-chip memory, and operate at lower clock rates, generally in the range of 75-160 MHz. But beyond these similarities, there are vast differences: basic DSPs use compound, multi-operation instructions, whereas basic GPPs typically use simple, single-operation RISC instructions. Generally, several RISC instructions are required to perform the work of one compound DSP instruction, and this ultimately translates into a performance advantage for basic DSPs.

In the higher cost and performance range, DSPs and GPPs employ various techniques to increase both the clock rate and the amount of work achieved per clock cycle. Thus, increased performance is a function of both increased parallelism and increased clock rates. Later, we will consider many of the architectural characteristics of high-performance DSPs and GPPs, and how these classes of processors compare both to each other and to their more basic counterparts.

Signal Processing Algorithms Mold DSP Architectures

From the outset, DSP processor architectures have been molded by digital signal processing algorithms. For nearly every feature found in a DSP processor, there are signal processing algorithms whose computation is eased in some way by inclusion of this feature. Therefore, perhaps the best way to understand DSP architectures is to first examine typical signal processing algorithms and identify how their computational requirements have influenced the architectures of DSP processors. Initially, we'll limit our scope to basic DSPs and GPPs; later, we'll broaden the scope to include high-performance DSPs and GPPs. As a case study, we will consider one of the most common signal processing algorithms, the finite impulse response (FIR) filter.

Fast Multipliers

The FIR filter is mathematically expressed as $y[n]=\sum_{k=0}^{N-1} h[k]x[n-k]$, where $x[n]$ is a vector of input data and $h[n]$ is a vector of filter coefficients. To produce an output sample, a set of recent input samples is multiplied by a set of constant coefficients. Hence, the main component of the FIR filter algorithm is a dot product: a series of multiplications where the multiplication products are summed, or "accumulated." These operations are not unique to the FIR filter algorithm; in fact, multiplication (often combined with accumulation of products) is one of the most common operations performed in signal processing—convolution, infinite impulse response (IIR) filtering, and Fourier transforms all also involve heavy use of multiply-accumulate (or "MAC") operations.

Early microprocessors implemented multiplications via a series of shift and add operations, each of which consumed one or more clock cycles. In 1982, however, Texas Instruments (TI) introduced the first commercially successful "DSP processor," the TMS32010, which incorporated specialized hardware to enable it to compute a multiplication in a single clock cycle. As might be expected, faster multiplication hardware yields faster performance in many digital signal processing algorithms. Today, hardware multipliers are standard in both basic GPPs and DSPs. The multiplier in a basic DSP is usually combined with an adder to facilitate single-cycle multiply-accumulate operations; the resulting hardware is referred to as a "MAC unit."

Multiple Execution Units

Digital signal processing applications typically have very high computational requirements in comparison to other types of processor applications. This is because they generally execute math-intensive signal processing algorithms (such as FIR filtering) in real time on signals sampled at data rates of 10-100 kHz or higher. Hence, basic DSP processors often include several independent execution units that are capable of operating in parallel—for example, in addition to the MAC unit, they typically contain an arithmetic-logic unit (ALU), shifter, and address generation unit. Basic GPPs in contrast, are generally not capable of parallel operations.

Efficient Memory Accesses

Executing MAC operations at a rate of one MAC per clock cycle requires more than just a single-cycle MAC unit. It also requires the ability to fetch the MAC instruction and the operands from memory in a single cycle. Hence, good digital signal processing performance requires high memory bandwidth—higher than is typically supported on a basic GPPs, which usually rely on a single-bus connection to memory and can only make one access per clock cycle. To address the need for increased memory bandwidth, designers of early basic DSP processors developed memory architectures that could support multiple memory accesses per cycle. The most common approach (which is still used in many DSP architectures) was to use two or more separate banks of memory, each of which was accessed by its own bus and could be read or written during every clock cycle. Often, instructions were stored in one memory bank while data was stored in another. With this arrangement, the processor could fetch both an instruction and a data operand in parallel in every cycle. Since many digital signal processing algorithms (such as FIR filters) consume two data operands per instruction (e.g., a data sample and a coefficient), some architects have adopted an extension to this idea: divide data memory into two separate banks (e.g., X data and Y data), each allowing one access per instruction cycle—thus enabling the processor to execute a MAC in a single cycle. In contrast, a basic GPP requires one instruction cycle per memory access and per arithmetic operation. Thus in the context of sustained sequential MAC operations, a basic GPP typically requires several instruction cycles to complete each MAC operation—even if a single-cycle multiplier is available.

Basic DSPs often further support high memory bandwidth requirements via dedicated hardware for calculating memory addresses. These address generation units operate in parallel with the DSP processor's main execution units, enabling the processor to access data at new locations in memory (for example, stepping through a vector of coefficients) without pausing to calculate the new address. In contrast, basic GPPs do not provide separate address generation units and must rely on the main arithmetic logic unit (ALU) to perform address calculations. This typically requires one instruction cycle per address update, widening the performance gap between basic DSPs and GPPs in the context of tasks such as sustained sequential MAC operations.

Memory accesses in digital signal processing algorithms tend to exhibit very predictable patterns; for example, for each sample in an FIR filter, the filter coefficient vector is accessed sequentially from start to finish; when processing the next input sample, accesses start over from the beginning of the coefficient vector. (This is in contrast to other types of computing tasks, such as database processing, where accesses to memory are less predictable.) Address generation units in basic DSP processors take advantage of this predictability by supporting specialized addressing modes that enable the processor to access data efficiently in patterns commonly found in digital signal processing algorithms. The most common addressing mode is register-indirect addressing with post-increment, which is used to automatically increment the address pointer for algorithms where repetitive computations are performed on a series of data stored sequentially in memory. Without this feature, the processor would spend instruction cycles incrementing the address pointer, as is the case with basic GPPs. Many DSP processors also support "circular addressing," which allows the processor to access a block of data sequentially and then automatically wrap around to the beginning address—exactly the pattern used to access coefficients in FIR filtering, for example. Circular addressing is also very helpful in implementing first-in, first-out buffers, commonly used for I/O and for FIR filter input data delay lines. Another specialized DSP addressing mode is bit-reversed addressing, which is used in certain fast Fourier transform (FFT) implementations. Basic GPPs do not support specialized DSP addressing modes like circular or bit-reversed addressing. These addressing modes, if needed, must be implemented in software, or alternative algorithms must be selected that do not require these addressing modes.

Data Format

Most DSP processors use a fixed-point numeric data type instead of the floating-point format most commonly used in scientific applications. In a fixed-point format, the binary point (analogous to the decimal point in base 10 math) is located at a fixed location in the data word. In contrast, in floating-point formats numbers are expressed using an exponent and a mantissa; the binary point essentially "floats" based on the value of the exponent. Floating-point formats allow a much wider range of values to be represented and virtually eliminate the risk of numeric overflow in many applications.

Digital signal processing applications typically must pay careful attention to numeric fidelity (e.g., avoiding overflow). Since numeric fidelity is far more easily maintained using a floating-point format, it may seem surprising that most DSP processors use a fixed-point format. In many applications, however, DSP processors face additional constraints: they must be inexpensive and provide good energy efficiency. At comparable speeds, fixed-point processors tend to be cheaper and less power-hungry than floating-point processors because floating-point formats require more complex hardware to implement so DSP processors predominately use a fixed-point numeric format. Basic GPPs also generally use fixed-point numeric formats, for many of the same reasons that DSPs do. Sensitivity to cost and energy consumption also influences the data word width used in DSP processors. DSP processors tend to support the shortest data word that will provide adequate precision in the target applications. Most fixed-point DSP processors support 16-bit data words because 16-bit width is sufficient for many digital signal processing applications. (A few fixed-point DSP processors support 20, 24, or 32 bits to enable higher precision in some applications, such as high-fidelity audio processing.) Basic GPPs generally support 32-bit fixed-point data, offering a better match for applications requiring higher precision than the DSPs supporting 16-bit format.

To ensure adequate signal quality while using fixed-point data, DSP processors typically include specialized hardware to help programmers maintain numeric fidelity throughout a series of computations. For example, most DSP processors include one or more "accumulator" registers to hold the results of summing several multiplication products. Accumulator registers are typically wider than other registers; the extra bits, called "guard bits," extend the range of values that can be represented and thus helps avoid overflow. In addition, DSP processors usually include good support for saturation arithmetic, rounding, and shifting, all of which are useful for maintaining numeric fidelity. Basic GPPs, in contrast, usually have uniform register widths, i.e., they do not provide larger accumulator style registers, and do not support saturation arithmetic or rounding. To some extent, however, the larger native data word size typical of GPPs (32 bits vs. 16 bits for DSPs) reduces the need for saturation, scaling, and rounding in the first place.

Zero-Overhead Looping

Digital signal processing algorithms typically spend the vast majority of their processing time in relatively small sections of software that are executed repeatedly; i.e., in loops. Hence, most DSP processors provide special support for efficient looping. Often, a special loop or repeat instruction is provided which allows the programmer to implement a for-next loop without expending any clock cycles for updating and testing the loop counter or branching back to the top of the loop. This feature is often referred to as "zero-overhead looping." In contrast, basic GPPs do not provide special loop oriented operations, beyond normal conditional branch instructions.

On-chip Integration

To gain low-cost, high-performance input and output, most DSP processors incorporate specialized on-chip peripherals, such as buffered synchronous serial ports, and streamlined I/O handling mechanisms, such as low-overhead interrupts and direct memory access (DMA). This allows data transfers to proceed with little or no intervention from the processor's computational unit. Basic GPPs in contrast, typically incorporate more generalized peripherals, like universal asynchronous serial interfaces and often do not include a DMA controller.

Specialized Instruction Sets

DSP processor instruction sets have traditionally been designed with two goals in mind. The first is to make maximum use of the processor's underlying hardware, thus increasing efficiency. The second goal is to minimize the amount of memory space required to store DSP programs, since digital signal processing applications are often quite cost-sensitive and the cost of memory contributes substantially to overall chip and/or system cost. To accomplish the first goal, basic DSP processor instruction sets generally allow the programmer to specify several parallel operations in a single instruction, typically including one or two data fetches from memory (along with address pointer updates) in parallel with the main arithmetic operation. With the second goal in mind, instructions are kept short (thus using less program memory) by restricting which registers can be used with which operations and restricting which operations can be combined in an instruction. To further reduce the number of bits required to encode instructions, DSP processors often have only a small number of registers divided into special-purpose groups. DSP processors may use mode bits to control some features of processor operation (for example, rounding or saturation) rather than encoding this information as part of the instructions.

The overall result of this approach is that basic DSP processors tend to have highly specialized, complicated, and irregular instruction sets. Combined with multiple memory spaces, multiple buses, and highly specialized arithmetic and addressing hardware, these instruction sets make basic DSP architectures very poor compiler targets. While a compiler can certainly take C source code and generate assembly code for a basic DSP, this compiler-generated code is usually far less efficient than hand written assembly code. Digital signal processing applications typically have very high computational demands coupled with strict cost constraints, making efficient code essential. For these reasons, when selecting a processor, programmers often need to consider the quality of compiler-generated code and the ease or difficulty of writing key routines in assembly language.

Basic GPPs, with their simple single-operation RISC instructions, regular register sets, and simpler memory models are excellent compiler targets. Programmers who write software for basic GPPs typically don't have to worry much about the ease of use of the processor's instruction set (as they tend to be relatively benign), so programmers generally develop applications in a high-level language, such as C or C++. The main motivation to hand write assembly for a basic GPP is to gain access to instructions the compiler does not support, although this scenario is rare in the context of basic GPPs.

High Performance Processors

Enhancements to Basic Architectures

Processor architects who want to improve performance beyond the gains afforded by faster clock speeds and must find a way to get more useful work out of every clock cycle. One approach taken with DSPs is to extend their architectures by adding parallel execution units, typically a second multiplier and adder. These hardware enhancements are combined with an extended instruction set that takes advantage of the additional hardware by allowing more operations to be encoded in a single instruction and executed in parallel. We refer to this type of processor as an “enhanced basic DSP processor,” because it is based on the basic DSP processor architectural style rather than being an entirely new approach. With this increased parallelism, enhanced basic DSP processors can execute significantly more work per clock cycle—for example, two MACs per cycle instead of one. Enhanced basic DSP processors typically have wider data buses to allow them to retrieve more data words per clock cycle to keep the additional execution units fed. They may also use wider instruction words to accommodate specification of additional parallel operations within a single instruction. Increases in cost and power consumption due to the additional hardware and architectural complexity are largely offset by increased performance (and, in some cases, by the use of more advanced fabrication processes), allowing these processors to maintain cost-performance and energy consumption similar to those of basic DSPs.

Although basic DSPs provide superior signal processing performance compared to basic GPPs, many applications require a mixture of control-oriented software and digital signal processing software. An example is the digital cellular phone, which must implement both supervisory tasks and voice-processing tasks. In general, basic GPPs provide good performance and code density in controller tasks and poor performance and code density in digital signal processing tasks. Hence, basic GPPs can be an attractive option for applications that combine some signal processing tasks with extensive control-oriented tasks. In an effort to provide a “best of both worlds” solution, vendors such as Hitachi, ARM, and Intel have created DSP-enhanced versions of their basic GPPs. Each vendor has employed a different approach to adding digital signal processing functionality to its existing GPP designs, borrowing and adapting the architectural features common among basic DSP processors. Many of these DSP-enhanced GPPs achieve signal processing performance that is comparable to that of basic DSP processors while maintaining many positive attributes of basic GPPs, such as good performance on control-oriented tasks. DSP-enhanced GPPs can be as an alternative to using a basic DSP in combination with a basic GPP.

High-Performance DSPs

Enhanced basic DSP processors provide improved performance by allowing more operations to be encoded in every instruction, but because they follow the trend of using specialized arithmetic and addressing hardware along with complex, compound instructions, they suffer from some of the same problems as basic DSPs: they are difficult to program in assembly language and they are unfriendly compiler targets. With the goals of achieving high performance and creating architectures that lend themselves to the use of compilers, newer high-performance DSP processors use a "multi-issue" approach. Multi-issue processors achieve a high level of parallelism by issuing and executing instructions in parallel groups rather than one at a time. The advantage of the multi-issue approach is a processor with a significant increase in parallelism with a simpler, more regular architecture and instruction set that lends itself to efficient compiler code generation.

TI was the first vendor to use a multi-issue approach in a mainstream commercial DSP processor. TI's VLIW (very long instruction word) TMS320C62x, introduced in 1996, was dramatically faster than other mainstream DSP processors available at the time. Other vendors have since followed suit, and now all of the major DSP processor vendors (TI, Analog Devices, and Motorola) employ multi-issue architectures for their latest high-performance processors.

In a VLIW architecture, the assembly language programmer (or code-generation tool) specifies which instructions will be executed in parallel. Hence, instructions are grouped at the time the program is assembled, and the grouping does not change during program execution. VLIW architectures provide multiple execution units, each of which executes its own instruction. The TMS320C62x, for example, contains eight independent execution units. VLIW processors typically issue a maximum of between four and eight instructions per clock cycle, which are fetched and issued as part of one long super-instruction—hence the name "very long instruction word."

When a processor issues multiple instructions per cycle, it must be able to determine which execution unit will process each instruction. Traditionally, VLIW processors have used the position of each instruction within the super-instruction to determine to where the instruction will be routed. Newer VLIW architectures do not use positional super-instructions, however, and instead include routing information within each sub-instruction. This has the benefit of reducing program memory use, as super-instructions that use every execution slot have the burden of needing to explicitly encode NOP operations in unused execution slots. In contrast to basic and enhanced basic DSPs, VLIW DSPs tend to use simpler instructions that typically encode one or two operations. These simpler instructions ease instruction decoding and execution, allowing VLIW DSPs to execute at higher clock rates than basic or enhanced basic DSP processors.

Although their instructions typically encode only one or two operations, most current VLIW DSPs use wider instruction words than basic DSP processors—for example, 32 bits instead of 16. There are a number of reasons for using a wider instruction word. In VLIW architectures, a wide instruction word may be required in order to specify which functional unit will execute the instruction. Wider instructions allow the use of larger, more uniform register sets (rather than the small sets of specialized registers common among basic DSP processors), which in turn enables higher performance. Relatedly, the use of wide instructions allows a higher degree of consistency and regularity in the instruction set; the resulting instructions have few restrictions on register usage and addressing modes, making VLIW processors better compiler targets (and easier to program in assembly language). There are disadvantages, however, to using wide, simple instructions. Since each VLIW instruction is simpler than a basic DSP processor instruction, VLIW processors tend to require many more instructions to perform a given task. Combined with the typically wider instruction words than those on basic DSP processors, this characteristic results in relatively high program memory usage. High program memory usage, in turn, may result in higher chip or system cost because of the need for additional ROM or RAM. Furthermore, to support execution of multiple parallel instructions, VLIW processors must have sufficient instruction decoders, buses, registers, and memory bandwidth. VLIW processors typically use either wide buses or a large number of buses to access data memory and keep the multiple execution units fed with data. The added hardware tends to increase energy consumption and processor cost.

Since they tend to use simple instructions, the architectures of VLIW DSP processors are in some ways more like a general-purpose processor than a highly specialized basic DSP architecture. VLIW DSP processors often omit some of the features that were, until recently, considered virtually part of the definition of a "DSP processor." For example, VLIW DSPs do not typically include zero-overhead looping instructions; they require the processor to execute instructions to perform the operations associated with maintaining a loop. This does not necessarily result in a loss of performance, however, since VLIW processors are able to execute many instructions in parallel. The operations needed to maintain a loop, for example, can be executed in parallel with several arithmetic computations, achieving the same effect as if the processor had dedicated looping hardware operating in the background.

High Performance GPPs

Like high-performance DSPs, high-performance GPPs also employ multi-issue architectures that execute multiple instructions in parallel. Many of the multi-issue concepts discussed in the context of high-performance DSPs hold true for high-performance multi-issue GPPs. For example, both types of processors issue multiple independent instructions in parallel, both provide multiple execution units to execute the parallel instructions, and both are effective at improving performance compared to their single-issue counterparts. But high-performance GPPs, by and large, are not VLIW architectures, but rather are superscalar architectures. The difference in the way these two types of architectures schedule instructions for parallel execution is important in the context of using them in real-time digital signal processing applications.

Superscalar processors contain specialized hardware that determines at run-time which instructions will be executed in parallel based on inter-instruction data dependencies and resource contention. This shifts the burden of scheduling parallel instructions from the programmer or tools to the processor. An important benefit of the superscalar approach is the ability to maintain binary compatibility with previous generations of the same processor family. For example, some superscalar processors simply duplicate the execution units of an existing single-issue GPP or DSP processor, producing two execution paths, each one identical to the single execution path of the earlier single-issue processor. In the worst case, software written and compiled for the earlier single-issue processor runs on the new multi-issue processor using a single execution path, while the other path remains idle, resulting in performance similar to the original processor (assuming a similar clock rates and underlying microarchitectures). In the best case, the scheduling hardware can issue two instructions in parallel in every instruction cycle, effectively doubling throughput, and potentially improving performance by a factor of two. Superscalar processors typically issue and execute fewer instructions per cycle than VLIW processors—usually between two and four.

A challenge for programmer using a superscalar processor is that superscalar processors may group the same set of instructions differently at different times in the program's execution; for example, it may group instructions one way the first time it executes a loop, then group them differently for subsequent iterations. Because superscalar processors dynamically schedule parallel operations, it may be difficult for the programmer to predict exactly how long a given segment of software will take to execute. The execution time may vary based on the particular data accessed, whether the processor is executing a loop for the first time or the third, or whether it has just finished processing an interrupt, for example. This uncertainty in execution times can pose a problem for digital signal processing software developers who need to guarantee that real-time application constraints will be met in every case. Measuring the execution time on hardware doesn't solve the problem, since the execution time is often variable. Determining the worst-case timing requirements and using them to ensure that real-time deadlines are met is another approach, but this tends to leave much of the processor's speed untapped. Dynamic features also complicate software optimization. As a rule, DSP processors have traditionally avoided dynamic features (such as superscalar execution, dynamic caches, and dynamic branch prediction) for just these reasons. In general, high-performance multi-issue DSPs are VLIW architectures, whereas high-performance multi-issue GPPs are superscalar architectures. This distinction isn't arbitrary, but rather is driven by key requirements related to each class of processor: execution time predictability in DSPs and compatibility requirements in GPPs.

Common Characteristics

VLIW and superscalar processors often suffer from high energy consumption relative to their more basic single-issue counterparts; in general, multi-issue processors are designed with an emphasis on speed rather than energy efficiency. These processors often have more execution units active in parallel than single-issue processors, and they require wide on-chip buses and memory banks to accommodate multiple parallel instructions and to keep the multiple execution units supplied with data, all of which contribute to increased energy consumption.

Because they typically have had high memory usage and energy consumption, VLIW and superscalar processors have historically mainly targeted applications which have very demanding computational requirements but are not very sensitive to cost or energy efficiency. For example, a VLIW processor might be used in a cellular base station, but not in a portable cellular phone. This trend, however, is changing, as portable devices increasingly require more computational power and processor vendors continually strive to produce energy efficient versions of their high-performance architectures. For example, Motorola's StarCore SC140-based VLIW MSC8101 has sufficiently low energy consumption to enable its use in portable products.

SIMD

High-performance DSPs and GPPs often further increase the amount of parallelism available by incorporating single-instruction, multiple-data (SIMD) operations. SIMD is not a class of architecture itself, but is instead an architectural technique that can be used within any of the classes of architectures we have described. SIMD improves performance on some algorithms by allowing the processor to execute multiple instances of the same operation in parallel using different data. For example, a SIMD multiplication instruction could perform two or more multiplications on different sets of input operands in parallel in a single clock cycle. This technique can greatly increase the rate of computation for some vector operations that are heavily used in signal processing applications. In the case where SIMD is included in a multi-issue architecture, the resulting parallelism can be quite high. For example, if two multiplication instructions can be issued per instruction cycle and these two instructions are each four-way SIMD instructions, the result is eight independent multiplications per clock cycle.

High-performance GPPs such as Pentiums and PowerPCs have been enhanced to increase signal processing performance via the addition of SIMD-based instruction-set extensions, such as MMX and SSE for the Pentium and AltiVec for the PowerPC. This approach is good for GPPs, which typically have wide resources (buses, registers, ALUs), that can be treated as multiple smaller resources to increase parallelism. For example, Motorola's PowerPC 74xx with AltiVec has very powerful SIMD capabilities; its wide, 128-bit vector registers are logically partitioned into four 32-bit, eight 16-bit, or sixteen 8-bit elements. The PowerPC 74xx can perform eight 16-bit multiplications per cycle. Using this approach, high-performance GPPs are often able to achieve performance on DSP algorithms that is better than that of even the fastest DSP processors. This surprising result is partly due to the effectiveness of SIMD, but also because high-performance GPPs operate at extremely high clock speeds in comparison to DSP processors; they typically operate at upwards of 2500 MHz or more, while the fastest DSP processors are typically in the 600-1000 MHz range.

On DSP processors with SIMD capabilities, the underlying hardware that supports SIMD operations varies widely. Analog Devices, for example, modified its basic floating-point DSP architecture, the ADSP-2106x, by adding a second set of execution units that exactly duplicate the original set. The resulting architecture, the ADSP-2116x, has two sets of execution units that each include a MAC unit, ALU, and shifter, and each has its own set of operand registers. The augmented architecture can issue a single instruction and execute it in parallel in both sets of execution units using different data—effectively doubling performance in some algorithms.

Alternatively, some DSP processors split their execution units into multiple sub-units, similar to high-performance GPPs. Perhaps the most extensive SIMD capabilities in a DSP processor to date are found in Analog Devices' TigerSHARC processor. TigerSHARC is a VLIW architecture, and combines the two types of SIMD: one instruction can control execution of the processor's two sets of execution units, and an instruction can specify a split-execution-unit (e.g., split-ALU or split-MAC) operation that will be executed in each set. Using this hierarchical SIMD capability, TigerSHARC can execute eight 16-bit multiplications per cycle, like the PowerPC 74xx.

Making effective use of processors' SIMD capabilities can require significant effort on the part of the programmer. Programmers often must arrange data in memory so that SIMD processing can proceed at full speed (e.g., arranging data so that it can be retrieved in groups of two, four, or eight operands at a time) and they may also have to re-organize algorithms, or write additional code that reorganizes data in registers to make maximum use of the processor's resources. SIMD is only effective in algorithms that can process data in parallel; for algorithms that are inherently serial (that is, algorithms that tend to use the result of one operation as an input to the next operation), SIMD is generally not of use.

Development Support

Development support refers to the software and hardware infrastructure that that is available to assist users in developing products based on the processor. This typically includes software development tools, software component libraries, hardware emulators, and evaluation boards. Development support varies significantly between GPPs and DSPs, between basic and high-performance processors, and from one processor to another within these categories.

As a group, DSPs have good to excellent signal-processing-specific tool support, while GPPs typically have poor signal-processing-specific tool support. The situation is similar in terms of third-party signal processing software components: DSP processors have poor to excellent support, depending on the processor, whereas GPPs typically have poor support (vendor supplied signal processing software, however, is improving for GPPs as vendors increasingly strive for competitive advantages, particularly in multimedia applications). For third-party software for tasks other than signal processing, DSP processors typically have poor support, whereas GPPs have poor to excellent support, depending on the specific processor.

An important class of third-party software is real-time operating systems. A typical DSP may have a small handful of real-time operating systems available for it, usually these don't include the most popular real-time operating systems. In contrast, GPPs often have extensive support from a wide range of real-time operating systems including some of the most popular real-time operating systems such as VxWorks from Wind River Systems.

Finally, DSP processor tools tend to have links to high-level signal processing tools like MATLAB. General-purpose processor tools typically lack these links, but instead have links to other kinds of high level tools, such a user interface builders.

On-Chip Integration

We use the phrase "on-chip integration" to refer to all of the elements on a chip besides the processor core itself. A key consideration with on-chip integration is the appropriateness of the on-chip integration to the application at hand. If the elements on the chip suit the needs of the application well, they can be an enormous benefit to the system designer. They result in reduced cost, lower design complexity, reduced power consumption, reduced product size and weight, and even reduced electromagnetic interference because fewer printed circuit board traces will be required.

In terms of on-chip integration, basic GPPs offer a wide range of on-chip integration options. Basic DSPs offer fewer on-chip integration options, but offer options that are better suited to the needs of specific signal processing applications. For example, a typical basic DSP would provide one or more synchronous serial ports, well suited for digital signal processing. These serial ports often include some intelligence, such as the ability to manage their own transfers to and from memory without intervention from the processor core.

High-performance DSPs usually have fewer on-chip integration elements than their basic DSP cousins. Their on-chip integration level can be characterized as moderate, and the elements included are usually not specific to a particular application. This is because high-performance DSPs are less narrowly focused on specific applications and because these processors are not intended for highly cost-sensitive applications.

High-performance embedded GPPs usually have moderate to extensive on-chip integration. This integration may be oriented to a specific application, but usually isn't tailored to the needs of specific digital signal processing applications. High-performance CPUs usually limit their on-chip integration to memory in the form of caches and memory management units to support PC operating systems, plus external bus interfaces. Because of their limited on-chip integration, using these processors in an embedded application usually requires significant external support and interface circuitry; this increases design complexity and can drive up cost and power consumption.

Conclusions

For applications with moderate signal processing requirements, basic DSPs and GPPs are often both viable candidates. As the signal processing requirements increase, it is more likely that a basic DSP processor will be necessary because at some point a basic GPP just won't have the needed horsepower. But even in the context of more moderate signal processing requirements, a basic DSP may be better suited in terms of energy efficiency, memory use efficiency, and on-chip integration. On the other hand, if the application has extensive functionality outside of signal processing tasks—things like user interfaces and communications protocols—then the basic GPP may be preferred because of its features, its efficiency in those types of tasks, and its generally superior development tools and support.

Applications with heavy signal processing requirements will invariably require a high-performance processor. But since high-performance GPPs often have signal processing performance equal to that of high performance DSPs, which one should you use?

Since high-performance GPPs are competitive with high-performance DSPs in terms of speed and memory use, other factors become important in selecting and differentiating between the two. Primary among these other factors are on-chip integration and development support. In neither of these areas does either category of processor have a consistent advantage. In integration, most high-performance DSPs and high-performance GPPs have somewhat limited on-chip integration, so which class of processor has a better match for a particular application will vary widely from application to application and with the particular processors under consideration. In general, PC CPUs, with their minimal on-chip integration, are at a disadvantage in embedded applications.

Though in general their tools are very good, signal-processing-specific development infrastructure is usually a weakness of high-performance GPPs used in signal processing applications. This is an area where high-performance DSP processors still hold a significant advantage, though vendors of high-performance GPPs and their tool vendors are making progress in signal-processing-oriented tool support.

Finally, high-performance GPPs introduce a lack of execution time predictability compared to high-performance DSPs, which results from their use of dynamic features such as superscalar instruction execution. This timing variability can make it difficult to ensure real-time behavior in some situations, and can make it difficult to optimize software, which is often required in performance-hungry signal processing applications.

Resources

Berkeley Design Technology, Inc., "DSP Processor Fundamentals: Architectures and Features," Berkeley Design Technology, Inc., 1996.

www.BDTI.com for numerous white papers and presentation slides on processor architectures, DSP software development, and processor benchmark performance.