

Guest Editors' Introduction:

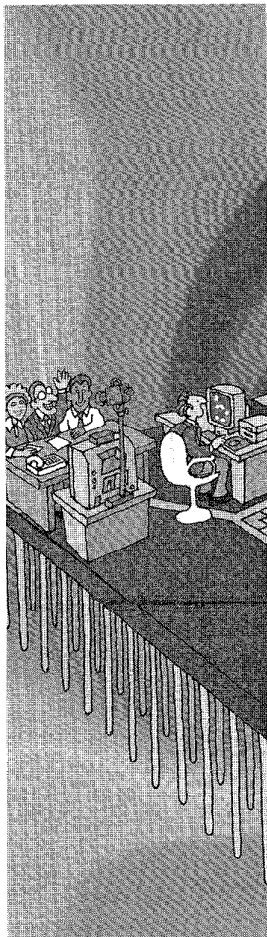
MEDIA PROCESSING: A NEW DESIGN TARGET

Ruby B. Lee

Hewlett-Packard

Michael D. Smith

Harvard University



Digital information processing used to involve only numbers and text. Graphical user interfaces (GUIs)—with pull-down menus, icons, and windows—extended this to two-dimensional graphics. Now we are seeing a transition to media-rich digital information. This includes images, video, audio, 2D and 3D graphics, animation, text, and numbers—collectively called multimedia information. Media processing—a shorter term for digital multimedia information processing—is the decoding, encoding, interpretation, enhancement, and rendering of digital multimedia information.

Multimedia information has existed for a long time, but in the past it involved analog processing. When we digitally represent images, video, audio, graphics, and animation, we reduce them to binary bits, which can be processed by digital processors.

Real-time digital video and audio, however, place certain minimum performance constraints on information devices and computers. For example, displaying MPEG-1 standard compressed video requires a device to decode and render 30 video frames per second, each frame comprising on the order of 10^5 pixels. The processing complexity is on the order of 10^3 operations per pixel per second, requiring on the order of 10^2 MOPS (million operations per second). MPEG-2 has frames that contain four times as many pixels and more complex decoding, requiring on the order of 1 GOPS (billion operations per second). Dealing with multimedia information thus requires a quantum leap in total computation, storage, and transmission, heralding a new era for computer designers.

Along with the need for higher performance, there is a paradoxical price constraint on media-processing computers. Video and audio have long been the media of entertainment and communications, and consumers

expect relatively low prices for devices such as televisions, VCRs, telephones, and game machines. So media processing represents a challenging design target: higher performance computers at an ever lower cost.

Evolution of media-processing devices

Initially, a separate special-purpose device or chip processed each media type. Audio, video, and 3D graphics required separate boards, each with its own specialized processing chips and memory.

Special-purpose audio chips with algorithms implemented in hardware evolved to chips we could configure for more than one algorithm, and then to digital signal processors (DSPs), which we could program for different algorithms. However, each audio data stream needed a separate DSP. For example, we had to use a separate DSP for each channel of sound, and another for telephony. In addition, the programmability tended to be at a low level, requiring DSP programmers to understand the underlying hardware very well.

Video and 3D graphics require greater bandwidth and processing performance than low-cost DSPs for audio or modems can provide. Today, video support consists of either 1) separate special-purpose video chips^{1,2} for MPEG-1, MPEG-2, and H.261 standards; 2) multi-algorithm video chip sets that implement a group of relatively similar video algorithms; or 3) programmable video processors.

A video processor can be considered a very sophisticated DSP specialized for processing pixel data. Gutttag et al. give an example,³ a multiprocessor on a single chip. The TMS320C82 is a cost-reduced version, which can be programmed to support many interesting multimedia applications, including H.324 videoconferencing over existing telephone lines. (See Golston's article, this issue.)

As video, audio, and graphics use increases, designers have considered consolidating memory and processing requirements. A common memory allows dynamic redistribution of memory resources based on the workload or the multimedia data mix. Consolidated processing is feasible because processors that satisfy video requirements can often meet the needs of other media data types with minimal incremental cost. In addition, combining memory and processing resources can reduce the cost and footprint of a media-processing device. However, a common memory system must have access bandwidth equivalent to the sum of the access needs of the separate media data types it services. Similarly, a common processor must have computational bandwidth equivalent to the sum of their processing needs.

Along these lines, the next step in the media-processing evolution appears to be media coprocessors and media processors. A media coprocessor is a computing device, usually a single chip, that simultaneously supports processing requirements of different media data types. It works in conjunction with a general-purpose processor that handles such functions as memory management, access protection, text processing, and number crunching. Golston's article in this issue discusses a sophisticated DSP that can also be called a media coprocessor (as do works by Foley⁴ and Rathnam and Slavenburg⁵ elsewhere).

A media processor also simultaneously processes different media data types, but it functions as a general-purpose processor as well. Hansen's article presents this issue's example of a media processor.

An alternative to these approaches is adding multimedia acceleration features to a general-purpose processor.⁶ In this issue, articles by Peleg and Weiser, Tremblay et al., and Lee discuss such multimedia extensions.

From the point of view of general-purpose microprocessors, the support of new data types often evolves from special-purpose chips, to optional coprocessors, to a non-optional part of a microprocessor's instruction set architecture (ISA). This evolution path was true for floating-point data types. For graphics data types, many microprocessors, such as the i860, M88110, and PA-RISC, have already introduced specialized support for graphics at the ISA level.^{7,9} While some of these instructions also accelerate other forms of multimedia data, the designers specifically targeted graphics functionality. The design goal of adding support in microprocessors for all forms of multimedia data has come more recently.

In the next few processor generations, the development of media processors with general-purpose functionality is likely to converge with the evolution of general-purpose microprocessors with multimedia extensions. While these processor types' major and secondary design targets are reversed, both may eventually cover and streamline the needs of general-purpose processing and media processing in full.

Multimedia extensions for microprocessors

Early attempts at introducing multimedia extensions into microprocessors have resulted in many similarities, with some notable differences. The basic similarity is that they are all based on operating in parallel on lower precision data

Media processing represents a challenging design target: higher performance computers at an ever lower cost.

packed into higher precision words. While this concept has appeared sporadically in earlier computers such as the Illiac IV, it has not been formalized, especially in its current reincarnation in microprocessors. The article by Lee in this issue attempts to formalize subword parallelism as a general technique for operation parallelism in processor design.

With the PA-7100LC, Hewlett-Packard introduced a small set of multimedia acceleration extensions, MAX-1,⁶ which performed parallel subword arithmetic. Though the design goal was to support all forms of multimedia, the application that best illustrated its performance was real-time MPEG-1,¹⁰ which was achieved with high-level C software, using macros to directly invoke MAX-1 instructions.

Next, Sun introduced VIS, a much larger set of multimedia extensions for UltraSparc microprocessors (see the article by Tremblay and coauthors in this issue). In addition to the parallel arithmetic instructions, VIS provides novel instructions specifically designed for reducing memory latency for algorithms that manipulate visual data. In addition, it includes a special-purpose instruction that computes the sum of absolute differences of eight pairs of pixels, similar to that found in media coprocessors such as Philips' Trimedia.⁵

Then, Hewlett-Packard introduced MAX-2 with its 64-bit PA-RISC 2.0 microprocessors.¹¹ MAX-2 added a few new instructions to MAX-1 for subword data alignment and rearrangement to further support subword parallelism. MAX-2 is interesting in its attempt to provide a minimalistic set of general-purpose media acceleration primitives. (See Lee's article in this issue.)

The MMX technology is a set of multimedia extensions for the Intel x86 family of processors (see Peleg's and Weiser's article in this issue). It lies between MAX-2 and VIS in terms of both the number and complexity of new instructions. The MMX designers have skillfully integrated a useful set of media acceleration instructions within the somewhat constrained register structure of the x86 architecture. MMX shares some characteristics of both MAX-2 and VIS, and also includes interesting new instructions, such as the parallel 16-bit multiply-accumulate instruction.

VIS, MAX-2, and MMX all have the same basic goal: to provide high-performance media processing on a general-purpose microprocessor. All three support the full set of subword-parallel instructions on 16-bit subwords, with a parallelism of four subwords per 64-bit register word. Differences exist in the type and amount of support they provide, some of which are driven by the needs of the target markets. For example, some support is provided for 8-bit subwords when

target markets include lower end multimedia (for example, games) in addition to higher fidelity multimedia (for example, workstations and medical imaging). An evaluation of the importance of the differences awaits the generation of comparable application performance data.

Software support

Currently, application developers have three common methods for accessing media-processing hardware within a system. They can

- invoke vendor-supplied, media-processing libraries;
- rewrite key portions of the application in assembly language using the media-processing instructions; or
- code in a high-level language (HLL) and use vendor-supplied macros that make available the functionality of the media-processing primitives through a simple function-call-like interface.

The simplest approach to improving media-processing application performance is to rewrite the system libraries to employ the media-processing hardware. The clear advantage of this approach is that existing applications can immediately take advantage of the new hardware without recompilation. However, the restriction of media-processing hardware to the system libraries also limits potential performance benefits. An application's performance will not improve unless it invokes the appropriate system libraries, and the overheads inherent in the general interfaces associated with system functions will limit application performance improvements. Even so, this is the easiest approach for a system vendor, and vendors have announced or plan to provide media-processing-enhanced libraries.

At the other end of the programming spectrum, an application developer can benefit from media-processing hardware by rewriting key portions of an application in assembly language. Though this approach gives a developer great flexibility, it is generally tedious and error prone. In addition, it does not guarantee a performance win (over code produced by an optimizing compiler), given the complexity of today's microarchitectures.

Recognizing the tedious and difficult nature of assembly coding, most media-processing hardware vendors have developed programming-language abstractions. These give an application developer access to the low-level media-processing primitives without having to actually write assembly language code. Typically, this approach results in a function-call-like abstraction that represents one-to-one mapping between a function call and a media-processing instruction.

There are several benefits to this approach. First, the compiler—not the developer—performs machine-specific optimizations such as register allocation and instruction scheduling. Second, this method integrates media-processing operations directly into the surrounding high-level code without an expensive procedure call to a separate assembly language routine. Third, it provides a degree of portability and isolation from the specifics of the underlying hardware implementation. If the media-processing primitives do not exist in hardware on the particular target machine, the

compiler can replace the media-processing macro with a set of equivalent operations.

The most common language extension for specifying media-processing primitives is to provide function-call-like macros within the C programming language. C compilers for the HP MAX-2, Intel MMX, MicroUnity, Sun VIS, and Philips Trimedia⁵ architectures support this approach. Several articles in this special issue give excellent code examples as illustrations. Each macro directly translates to a single media-processing instruction, and the compiler allocates registers and schedules instructions. This approach would be even more attractive to application developers if the industry agreed to a common set of macros, rather than having a different set from each vendor.

Future directions for compilers

While macros may be an acceptable—and even efficient—solution for invoking multimedia instructions within a high-level language, subword parallelism could be further exploited with automatic compilation from high-level languages to multimedia instructions. Some research directions toward automatic compilation techniques for media-processing architectures are discussed next.

Data streaming. One of the keys to a fast media-processing application is the efficient streaming of data into and out of the processor. Multimedia programs such as video decompression stress the data memory system in ways that the multilevel cache hierarchies of many general-purpose processors cannot handle efficiently. These programs are data intensive with working sets bigger than many first-level caches. Streaming memory systems and compiler optimizations aimed at reducing memory latency (for example, prefetching) have the potential to improve these applications' performance.¹² Current research in data and computation transformations for parallel machines¹³ may provide starting points for further gains in this area.

Subword parallelism. To fully automate the compilation process for media-processing architectures, the compiler must recognize opportunities for using subword-parallel (or packed-data) instructions. This requires techniques that identify the maximum width of the significant bits in the result of each operation. The compiler can obtain this information from the types of variables involved in the operation or from an analysis of the manipulation of these variables. (For example, if a variable is logically ANDed with 0xff, we know that all but the last 8 bits are zero.) Each of these information sources has its problems. With type information, a programmer's indiscriminate use of unnecessarily large data types may hide opportunities for subword-parallel instructions. Techniques such as Function Width Analysis,¹⁴ used in logic synthesis systems to find sequences of instructions amenable for conversion to programmable logic, may identify opportunities for subword parallelism. However, code sequences within loop structures with no clear iteration bound may make the compiler estimation of the function width too pessimistic.

Vectorization. Media-processing techniques have parallels in the compilation of numerical codes for vector machines. One of the greatest challenges for this new generation of "vectorizing" optimizations is that the media-processing applica-

tions are written in C, not Fortran. C's use of pointers makes the determination of alias information difficult. Improvements in interprocedural analysis or the use of programmer-specified directives would greatly improve the ability of the compiler to generate parallel or subword-parallel code.

THE EVOLUTION AND REFINEMENT of media-processing hardware has just begun. As programmable processors or coprocessors with media-processing enhancements gradually replace fixed-function, special-purpose devices, compiler support for these features will also improve. Today, an application developer who organizes program and data structures to exploit media-processing hardware achieves the best performance. Eventually, language extensions will probably emerge to support improved programmer efficiency without loss of application performance. Media processing—with its almost limitless appetite for computational power—provides an exciting new target for hardware and software design innovation. ■

References

1. T. Kondo et al., "Two-Chip MPEG-2 Video Encoder," *IEEE Micro*, Vol. 16, No. 2, Apr. 1996, pp. 51-58.
2. *CL480 and CL484 VideoCD Decoder User's Manual*, Part Number 92-0484-101, C-Cube Microsystems Inc., Milpitas, Calif., 1996.
3. K. Gutttag et al., "A Single-Chip Multiprocessor for Multimedia: The MVP," *IEEE Computer Graphics & Applications*, Vol. 12, No. 6, Nov. 1992, pp. 53-64.
4. P. Foley, "The Mpaact Media Processor Redefines the Multimedia PC," *Proc. Compton*, IEEE Computer Society Press, Los Alamitos, Calif., 1996, pp. 311-318.
5. S. Rathnam and G. Slavenburg, "An Architectural Overview of the Programmable Multimedia Processor, TM-1," *Proc. Compton*, IEEE CS Press, 1996, pp. 319-326.
6. R. Lee, "Accelerating Multimedia with Enhanced Microprocessors," *IEEE Micro*, Vol. 15, No. 2, Apr. 1995, pp. 22-32.
7. L. Kohn and N. Margulis, "Introducing the Intel i860 64-Bit Microprocessor," *IEEE Micro*, Vol. 9, No. 4, Aug. 1989, pp. 15-30.
8. K. Diefendorff and M. Allen, "Organization of the Motorola 88110 Superscalar RISC Microprocessor," *IEEE Micro*, Vol. 12, No. 2, Apr. 1992, pp. 40-63.
9. C. Dowdell and L. Thayer, "Scalable Graphics Enhancements for PA-RISC Workstations," *Proc. Compton*, IEEE CS Press, 1992, pp. 122-128.
10. L. Gwennap, "New PA-RISC Processor Decodes MPEG Video," *Microprocessor Report*, Vol. 8, No. 1, Jan. 24, 1994, pp. 16-17.
11. R. Lee and J. Huck, "64-bit and Multimedia Extensions for the PA-RISC 2.0 Architecture," *Proc. Compton*, IEEE CS Press, 1996, pp. 152-160.
12. D. Zucker, M. Flynn, and R. Lee, "Improving Performance for Software MPEG Players," *Proc. Compton*, IEEE CS Press, 1996, pp. 327-332.
13. J. Anderson, S. Amarasinghe, and M. Lam, "Data and Computation Transformations for Multiprocessors," *Proc. Fifth ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, ACM, New York, 1995, pp. 166-178.
14. R. Razdan and M.D. Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units," *Proc. 27th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, IEEE, Piscataway, N.J., 1994, pp. 172-180.



Ruby B. Lee is chief architect for multimedia architecture and senior architect for processors and systems in the Computer Systems Organization at Hewlett-Packard. She was instrumental in developing the architecture of several generations of PA-RISC processors and systems. She is also a consulting professor of electrical engineering at Stanford University. Her current interests are in operation parallelism, media processing, and system architecture.

Lee holds a BA from Cornell University, and an MS in computer science and a PhD in electrical engineering from Stanford University. She holds 12 patents in processor architecture, pipeline design, cache hints, branch optimizations, and multimedia architecture and algorithms. She is a member of the IEEE, ACM, Phi Beta Kappa, and Alpha Lambda Delta.



Michael D. Smith is an assistant professor of electrical engineering and computer science in the Division of Applied Sciences at Harvard University. His research focuses on the experimental realization of innovative compilation techniques and novel computer architectures to improve the capability and performance of computer systems. Earlier, he worked for Honeywell Information Systems, where he designed CPU boards and VLSI chip sets for a minicomputer product line.

Smith has a BS degree in electrical engineering and computer science from Princeton University, an MS degree in electrical engineering from Worcester Polytechnic Institute, and a PhD in electrical engineering from Stanford University. He is a member of the IEEE and the ACM, and is the recipient of a 1994 NSF Young Investigator Award.

Address questions concerning this special issue to Ruby B. Lee at Hewlett-Packard, 19410 Homestead Rd., MS 43UG, Cupertino, CA 95014; rblee@cup.hp.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 150

Medium 151

High 152