Real-Time Image Processing on a Custom Computing Platform

Peter M. Athanas
A. Lynn Abbott
Virginia Polytechnic Institute and State University

The authors explore the utility of custom computing machinery for accelerating the development, testing, and prototyping of a diverse set of image-processing applications.

Several aspects of image processing make it computationally challenging. A single image represents a data set of considerable size—typically 256K picture elements, or *pixels*, for a black-and-white image. Many tasks require that several operations be performed on each pixel in the image. Furthermore, when real-time operations are needed, they must be performed at live video rates, typically 30 images per second. To keep up with these capacious data rates and demanding computations in real time, the processing engine must provide specialized data paths, application-specific operators, creative data management, and careful sequencing and pipelining.

Hardware designers typically must perform extensive behavioral testing of a new concept before proceeding with an implementation. Due to the enormous processing time required to simulate a complex image-processing system, executing a VHDL model with a representative data set even on a fast workstation is not practical. Days, or even weeks, are commonly needed to simulate the processing of a single full-sized image. And since some applications process sequences of images, designers may need several hundred image simulations to adequately analyze only a few seconds of data. Because of this, they are often forced into a trade-off between how much testing can be afforded versus an acceptable risk in allowing a silicon iteration.

We discuss an alternative, automated approach: transforming the structural representation (or transforming a behavioral model) into a real-time implementation. With our system, a designer can proceed from a behavioral description of the image-processing task to a functioning prototype that can perform the task at full speed (rapid prototyping). Reconfiguration from one image-processing task to another does not require physical changes but is accomplished by downloading a hardware personalization database to a novel computing platform. Reconfiguration takes just seconds. A designer with this capability has

- a means for evaluating the performance of an experimental algorithm/architecture, and
- a working component that can be used in the development and testing of a much larger system.

We chose an experimental custom computing platform called Splash-2 to investigate this approach to prototyping real-time image-processing designs.[1] Custom computing platforms are emerging as a class of computers that can provide near application-specific computational performance. Designers can also configure them for a variety of tasks. Such platforms let designers customize specific operations for function and size, and data paths for individual applications.

0018-9162/95/$4.00 © 1995 IEEE

We developed a real-time image-processing system called VTSplash, based on the Splash-2 general-purpose platform. Splash-2 is an attached processor featuring programmable processing elements (PEs) and communication paths. The Splash-2 system uses arrays of RAM-based field-programmable gate arrays (FPGAs), crossbar networks, and distributed memory to accomplish the needed flexibility and performance. Even though Splash-2 was not designed specifically for image processing, its architectural properties are suited for the computation and data-transfer rates characteristic of this class of problems. The price/performance ratio of this system also makes it competitive with conventional real-time image-processing systems.

In this article, we explore the utility of custom computing machinery for accelerating the development, testing, and prototyping of a diverse set of image-processing applications. We first summarize architectural aspects of high-speed image processing. We next provide a synopsis of pertinent architectural features of the Splash-2 processor and describe its development environment. We then describe several image-processing tasks implemented on Splash-2 and conclude with a discussion of task performance.

ARCHITECTURAL ASPECTS OF IMAGE PROCESSING

Conventional general-purpose machines cannot manage the distinctive I/O requirements of most image-processing tasks; neither do they take advantage of the opportunities for parallel computation present in many vision-related applications. Parallel processing systems such as mesh computers or pipelined processors have been successfully applied to some image-processing tasks. Mesh architectures often provide very large speedup after an image is loaded, but overall performance often suffers severely from I/O limitations. Pipelined machines can accept image data in real time from a camera or other source, but historically they have proven difficult to reconfigure for various processing tasks. (The sidebar "Architectural considerations for image processing" further discusses the unique requirements of image-processing architectures.)

Image data are typically produced and conveyed in raster order, that is, pixels are presented serially, left-to-right for each image row, beginning with the top row. If a typical image frame is 512 rows × 512 columns of 8-bit pixels, the total data in a single frame is 262,144 pixels, or 2

ARCHITECTURAL CONSIDERATIONS FOR IMAGE PROCESSING

The goal of many image-processing tasks is to transform an input image into a new, enhanced version of the original. In some cases, each output pixel (or picture element) can be computed as a function of a small neighborhood of adjacent pixels from the input image. Figure A shows an example 3 × 3 neighborhood. Each output pixel depends on a different neighborhood in the original image. Conceptually, therefore, the output image is produced by sliding a 3 × 3 window over the input image, with an output pixel resulting for each new location of the window.

The choice of neighborhood operation determines the appearance of the output image. A weighted sum of neighborhood pixels, for example, could result in smoothed (low-pass filtered) or edge-enhanced (high-pass filtered) output images. Median filtering generates the median value of each neighborhood. Many other filter types are possible.[1-3]

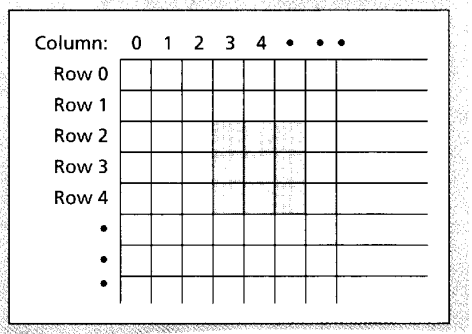

Figure A. Example image array. Each cell represents one pixel, which is commonly 8 bits for a monochrome image. The shaded area indicates a 3 × 3 neighborhood centered about pixel (3, 4).

Although the nine pixels of a 3 × 3 neighborhood are spatially localized in the physical image, this is not true in the signals produced by most video sources. For example, a typical video camera produces pixels in raster order, which means that the pixel values are generated serially beginning with row 0, followed by row 1, and so on. Figure B illustrates this process.

For processing purposes, the straightforward approach is to store the entire input image into local memory and access pixels as needed to produce the output image. However, this approach results in a latency of at least an entire image frame time before the processor can begin to generate output pixels. This latency can be reduced to less than the time of n rows (for an $n \times n$ neighborhood) in an architecture carefully designed to interleave memory reads and writes, effectively utilizing memory as a delay line. We used Splash-2 to implement both of these processing methods.

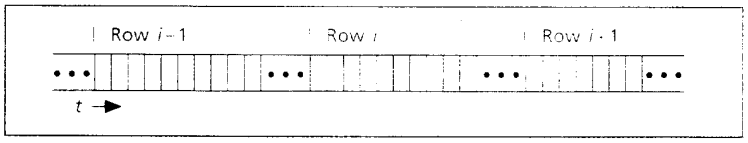

Figure B. Example image in raster order. Pixels are produced serially in row-major order. Highlighted pixels represent a single 3 × 3 image neighborhood.

References

1. R.M. Haralick and L.G. Shirapo, *Computer and Robot Vision*, Vol. I, Addison-Wesley, Reading, Mass., 1992.
2. B. Jähne, *Digital Image Processing*, Springer-Verlag, New York, 1991.
3. A. Rosenfeld and A. Kak, *Digital Picture Processing*, 2nd ed., Academic Press, New York, 1982.

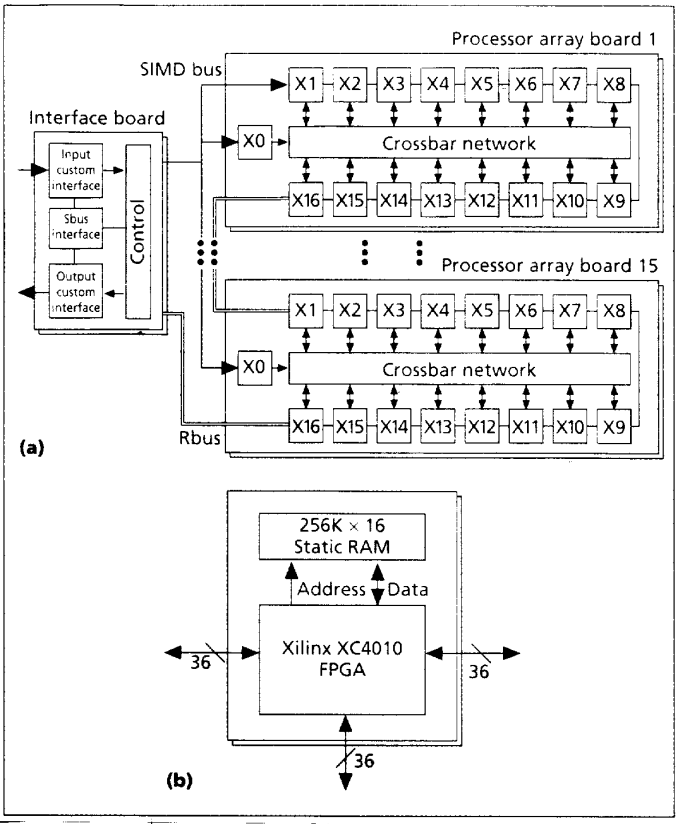

Figure 1. The Splash-2 architecture: (a) Each board contains 16 PEs (and one control processor) interconnected by neighbors and through a crossbar switch. Up to 15 processor array boards can be connected to an interface board; (b) Each PE consists of an FPGA and an SRAM.

megabits. This not only presents a computational challenge but also poses storage problems. Some image-related tasks, such as compression, tracking, and motion compensation, need information distributed in a single image (spatially distributed data) and also depend on data present in previous frames (temporally distributed data). Because of this, the processor must store and retrieve multiple frames quickly.

For simplicity, we assume that data represent monochrome light intensity values. Of course, pixel data are not restricted to represent only quantized brightness information. The applications discussed here are also relevant to other types of image data, such as

- color images;
- range images, for which each pixel represents a distance value;
- X-ray, ultrasound, or electron-microscope images, where each pixel depends on object density or other physical phenomena; and
- CT (computer tomography) images, for which each 2D (two-dimensional) image represents a reconstructed "slice" of density information within a 3D array.

CUSTOM COMPUTING HARDWARE

Here we review the properties of custom computing machines, using the Splash-2 platform as an example. We then show how the custom computing machine can be used as a component in a real-time processing system.

The Splash-2 platform

Splash-2 is a second-generation processor designed by the Supercomputing Research Center in Bowie, Maryland. It achieves high computational performance by executing an application in hardware customized to the needs of individual applications. A Splash-2 system consists of one to 15 Splash-2 array boards, an interface board, and a Sun SparcStation-2 host. Each array board contains 16 PEs, denoted as X1 through X16, arranged linearly and fully connected through a 16 × 16 crossbar switch. A seventeenth control element, X0, regulates the crossbar network. Figure 1a is a system block diagram.

Each PE within the Splash-2 array board (identified as X1 through X16 in Figure 1a and expanded in Figure 1b) consists of one FPGA and one fast static memory. The Xilinx XC4010 FPGA used in each PE consists of a 2D array of configurable logic blocks that can be connected internally with reconfigurable interconnection resources. Both the logic blocks and the interconnection resources are programmable through the host computer. Computational operations are implemented as logic circuits constructed within the FPGAs by the operation into individual blocks and then interconnecting them as required with the programmable switches. A fast 256K × 16 static RAM (SRAM) is attached to each FPGA, which allows one read or write access per clock cycle. Each PE has three 36-bit bidirectional data paths; one each to the left and right neighboring PEs and one to the crossbar switch. In addition, a 16-bit path exists between the FPGA and its SRAM. Several 1-bit signals support broadcasts, handshakes, and other special functions.

The input data stream to the Splash-2 processor array is provided by the interface board with a 36-bit SIMD bus to the X0 of each processor array board (and to the X1 of the first processor array board). The output data stream can be linked to multiple array boards by extending this stream from the X16 of one board to the X1 of the next. The output stream produced from the last array board is returned to the interface board. The control paths between the Sun SparcStation-2 host and the application program running on Splash-2 consist of a set of handshake registers (two on each Splash-2 array board), a global AND/OR mechanism, a broadcast signal, direct access to on-board memory, and an interrupt mechanism.

The crossbar network contains 16 36-bit bidirectional ports for augmenting interprocessor communications. Splash's crossbar switches can be used for both static and dynamic architecture adjustments. Static adjustments establish the data paths for fixed systolic-like tasks, while dynamic adjustments accommodate more complex data-movement paradigms. The control element X0 selects the interconnection structure used in any given clock cycle. The crossbar allows point-to-point, multicast, and broadcast communication between all PEs on each processing board and can readily change the topology to a mesh, linear array, hypercube, or other custom configuration. (Arnold, Buell, and Davis[1] provide more information on Splash-2.)

The image-processing platform

Figure 2 shows the VTSplash laboratory system we developed. A video camera or a VCR creates a standard RS-170 video stream. The signal produced from the camera is digitized with a custom-built frame-grabber card. This board not only captures images but also performs any needed sequencing or simple pixel operations before the data are presented to Splash-2. The frame-grabber card was built with a parallel interface that can be connected directly to the input data stream of the Splash-2 processor.

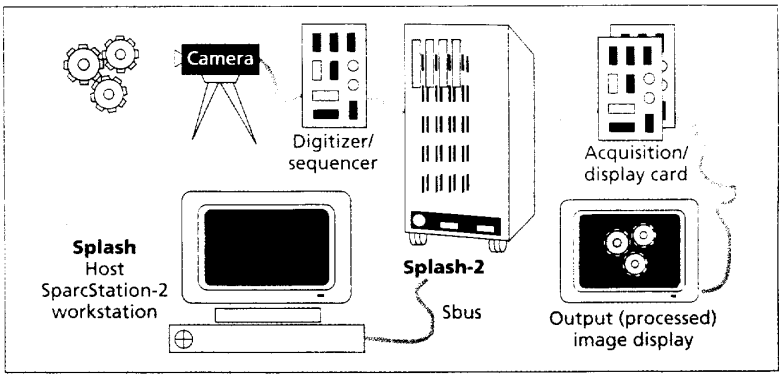

Figure 2. Components of the VTSplash laboratory system.

The VTSplash laboratory system uses two processor array boards. The output data produced by Splash-2—which can be a real-time video data stream, image overlay data, or some other form of information—is first presented to another custom board for converting the data to an appropriate format (if necessary). Once formatted, the data are then presented to a commercial image-acquisition/display card, which presents the images to a color video monitor. The SparcStation host configures Splash-2 arrays and sends runtime commands intermixed with the video stream (if needed). The laboratory system can be quickly reconfigured from one task to another in just a few seconds by downloading the hardware personalization database.

As mentioned earlier, Splash-2 is a general-purpose machine not specifically designed for image processing. Nonetheless, it is a suitable testbed for implementing a wide range of computer vision tasks, including those that require temporal processing. One Splash-2 processing board contains slightly more than 69 megabits of memory—enough for 32 frames of image data. (This number is based on 17 256K × 16 SRAM devices plus 12,800 bits of storage (maximum) in each of the 17 Xilinx 4010 chips.[1]) Not all this storage may be conveniently available to applications.

APPLICATION DEVELOPMENT ON SPLASH-2

While the programming environment for Splash-2 is one of the most advanced and automated in its class, numerous difficulties must be addressed before this type of machine can become accepted into mainstream computing. Here we provide a brief summary of the rapid prototype-development process from the formulation of task behavior to the generation of a physical database read for execution. We then assess some challenges that need to be addressed.

Basic design flow

Figure 3 illustrates the basic design flow for developing a typical Splash-2 application. (This simplified figure does not depict all possible iteration paths in the design process.) The first step in the process is the definition of the problem. As in all hardware and software system design, a sound problem definition will facilitate the design process.

Step two is the behavioral modeling of the problem. Typically, a VTSplash programmer models the problem by using the C programming language or a behavioral VHDL model. Not only is the model constructed to comply with the problem definition, but sample images are run through the model (when possible) for later comparison with the results of the synthesized implementation.

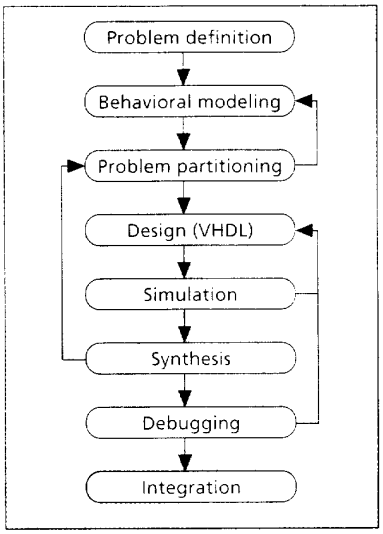

Figure 3. The application design process.

The next step, which is often difficult, consists of manually partitioning the model into a form suitable for final implementation on Splash-2. The model is first mapped onto processor boards and then partitioned more finely into individual PEs. The three main factors that drive a partition are time, area, and communication complexity.

The time and area factors are familiar problems discussed in the high-level synthesis and silicon compiler literature.[2] Time refers to how much computation is desired per clock cycle. Area refers to how much of the reconfigurable resources should be allocated to a given computation, to the total available reconfigurable resources within each processor board, and to each of the 17 PEs on each board. Even though Splash-2 contains ample hardware support to aid signal propagation between PEs, not all communications are equal in cost and in bandwidth (communication complexity). Splash-2 imposes limitations on available communication resources. Some of these are

- a total of 108 signals split equally between the left neighbor, right neighbor, and crossbar network;
- a 16-bit data path between a PE and its 0.5-megabyte RAM;
- several 1-bit signals for global communications and broadcasts; and
- a 36-bit data path between processing boards, along with several 1-bit global signals.

(These numbers are simplified somewhat for the sake of

Table 1. A representative list of image-processing categories and example tasks.

Class	Example image task	Description
Transformation	Convolution	Linear filtering operation
	Median filtering	Nonlinear filter used to eliminate "salt and pepper" noise
	Morphological filtering	Nonlinear operations that alter region shapes in an image Gray-scale erosion and dilation operations implemented
Combination	Laplacian pyramid generation	Produces an image hierarchy of decreasing image size and spatial resolution. The image for each pyramid level formed by taking the difference of two blurred versions of the original image.
Measurement	Histogram generation	Statistical operation for computing intensity distribution of pixels in an image
Conversion	Fast Fourier transform	Converts an image from the spatial domain to the frequency domain
	Hough transform	A voting scheme that detects the presence of lines (or parametric curves) from a set of points in an image
	Region detection and labeling	Finds connected regions in an image and assigns a unique label to each

this discussion. The actual communication structure is more intricate. Refer to Arnold, Buell, and Davis[1] for more detail.) Although these numbers may appear to be quite generous, the limits of these data paths will eventually disgruntle some designers. Not all applications easily map to these limitations, and tough design trade-offs must be considered. As it stands, few quantitative up-front measures are available to gauge partitioned alternatives. A designer must often wait until after the synthesis step before knowing whether a given problem partition is feasible.

Detailed structural design

After the problem is partitioned, the designer produces and verifies a detailed structural design. Many alternatives are available to designers for converting the structural representation into a hardware configuration database, including FPGA design tools like XBLOX.[3] However, the best-supported design environment for Splash-2 contains the Synopsys VHDL simulation and synthesis tools. The simulations for many of the image-processing tasks we discuss consumed several days of CPU time per run on a SparcStation-10—in many cases, for just a small fraction of an image. Because of this, only so much simulation can occur within a reasonable amount of time. Therefore, the stimulation input for a simulation run must be considered judiciously.

Debugging

Because simulations of the applications under development are based on VHDL models created prior to placement and routing of the FPGAs, they are barren of signal propagation annotation. Actual propagation delays in the Xilinx FPGAs are sensitive to the outcome of the placement and routing process, and these delays can have a disturbing effect on application behavior. To counter these problems, and to cope with the limited functional coverage that can be achieved with the simulation tools, a powerful debugging tool is available in the Splash-2 environment. The T2 interactive debugger[1] provides the power of conventional high-level language debuggers by allowing such features as monitoring internal state variables and tracing. Debugging a hardware/software codesign adds conceptual difficulties not found in traditional debugging environments. After the image operations are performing satisfactorily, they must be integrated within the body of an application. A rich C library to facilitate communication between host programs and attached processors is accessible within the Splash-2 environment.

Reducing development time

Although Splash-2 represents the state-of-the-art in custom computing processors—both in hardware capabilities and software support—it requires a substantial time investment to develop an application. To make this class of machinery more widely accepted and cost-effective, methods must be developed to reduce application-development time. Several promising endeavors focus on this issue.[4-6] Their main emphasis is depicted by the portions of the gray shaded region previously shown in Figure 3.

IMAGE-PROCESSING TASKS

Image operations have been classified into five generic classes.[7] An operation in the *combination* class takes two images and produces a new image of the same type. This is accomplished by combining each pair of elements from the input images into a new element. The *transformation* class accepts an image from a given class and produces a new image in the same class. The *measurement* class reduces an image of a given type into a scalar or vector. The *conversion* class refers to those operations that take an image of a given type and convert it into a new class. (The *generation* class, which produces a new image from scratch, concerns image synthesis rather than image processing and so is not considered here.)

To evaluate the agility of the VTSplash system, we modeled examples from each of the first four categories (with

varying degrees of difficulty). Table 1 summarizes the categories and gives examples of each evaluated on VTSplash. (We refer the interested reader to Jähne's text on the subject.[8])

Figure 4. Example filtering operations: (a) original image; (b) smoothed image created by applying a low-pass filter to the original image; and (c) edge image created by applying a simple high-pass filter. All images are processed as 512 × 512 pixels in size. The output images were obtained by using 8 × 8 templates on VTSplash.

Transformations: Linear and nonlinear filtering

Two-dimensional filtering techniques are very common in image processing. The most common methods process small neighborhoods in an input image to generate new pixels in an output image. Neighborhood-based filtering is characterized by the repeated application of identical operations and often serves as a preprocessing step followed by higher-level image analysis.

Neighborhood operations typically use a 2D template, usually rectangular, which is applied at every pixel in the input image. (The template is often called an operator or filter.) In the linear case, the hardware applies a template by centering it at a given pixel of the input image, multiplying each template pixel by the associated underlying image pixel, and summing the resulting products. The sum becomes the pixel value (for this template position) in the output image. Each new template position generates a single new output pixel value. (Special rules may be needed for pixels near the image borders.) Algebraically, this is represented as

$$I_{out}(r,c) = \sum_i \sum_j I_{in}(r+i,\ c+j) \times h(i,j)$$

where I_{in} is the input image, I_{out} is the output image, h is the filter, and r and c refer to the row and column location in the images. The summation is typically performed over a small window, often 3×3 or 7×7, as determined by h. Figure 4 shows examples.

Template operations can also be nonlinear. For example, designers can implement a median filter by using a template. For every position of the template, the hardware system chooses the median value from the image pixels covered by the template and uses it as the new pixel value for the output image. In this case, the template simply serves as a window and has no cell values. Another form of nonlinear image processing is based on mathematical morphology.[9] This algebra uses multiplication, addition (subtraction), and maximum (minimum) operations to produce output pixels. The filtering operations, known as erosion and dilation, can be used to perform such tasks as low- or high-pass filtering and feature detection. This approach provides less blurring than linear filtering.

Image-combination operations

After an image has been appropriately low-pass filtered, it can be subsampled without fear of violating the Nyquist criterion. If an image is recursively filtered and subsampled, the resulting set of images can be considered a single unit called a Gaussian pyramid. An image-processing system can use this data structure to reduce computational requirements by employing the lower-resolution portion of the pyramid to guide processing at higher-resolution levels. For some tasks (such as surveillance and road following), this approach can greatly reduce the overall amount of processing. (Burt and Adelson[10] provide a popular technique for generating these pyramids.)

In addition to low-pass pyramids, a system can generate band-pass (or Laplacian) pyramids, in which each level of the pyramid contains information from a single frequency band. VTSplash can process either type of pyramid by dynamically reconfiguring data paths through the crossbar. Gaussian and Laplacian pyramids are produced at 30 per second and 15 per second, respectively.

Measurement computations

Unlike the other processing classes, measurement operations typically do not produce a new output image. Instead, the goal is to extract descriptive statistics of the input image. For example, the mean and standard deviation of pixel values in the image are often of interest. These and similar statistics can be computed by using simple multiply-accumulate processing, where one such operation is required for each input pixel.

Real-time histogram generation, another useful operation, often constitutes an initial step for other applications, such as region detection and region labeling. In generating a histogram, the processor must maintain and update a 1D array that records the number of occurrences of particular pixel values. Histograms are often further analyzed and used to adjust parameters for image enhancement.

Image-conversion operations

The 2D discrete Fourier transform (DFT) is a useful operation in signal-processing applications but is often avoided because of its large computational requirements. Although linear, it differs from the neighborhood operations described above, since every transformed output pixel depends on every pixel of the input image. The problem can be simplified somewhat, since the 2D Fourier

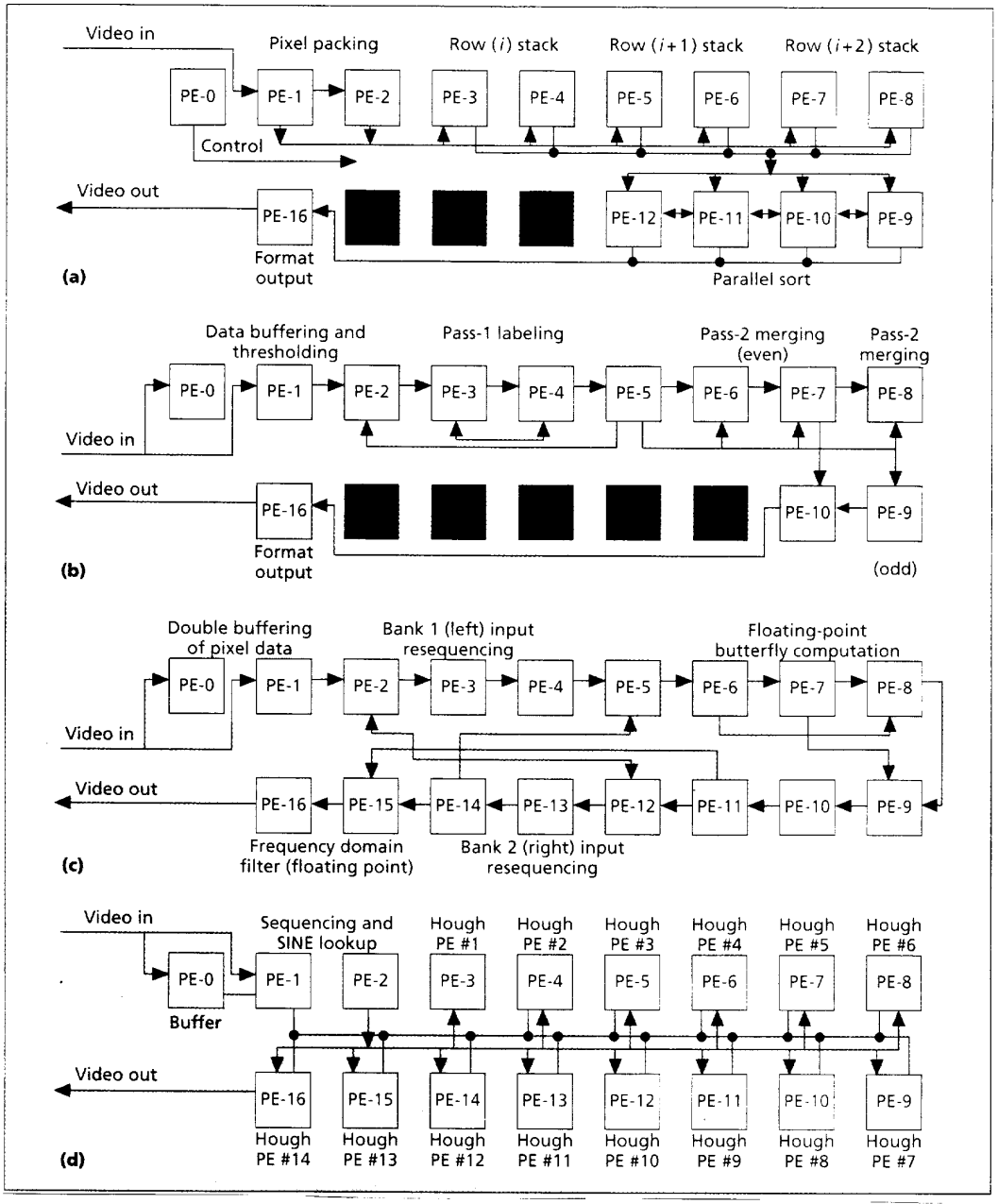

Figure 5. Examples of the communications structure and partitioning for four examples that use only one Splash-2 processor array: (a) median filter, (b) region detection and labeling, (c) FFT (forward transform), and (d) Hough transform. Solid squares at PE sites denote unused PEs.

transform can be decomposed into multiple 1D fast Fourier transforms (FFTs). For example, a 512×512 DFT can be implemented as 512 1D FFT computations (one for each row) followed by 512 additional 1D FFTs (one per column). We have implemented this on VTSplash using floating-point arithmetic.

The Hough transform, another 2D transformation, can be used after an edge-detection task to determine if a set of points lie on a straight line.[11] (The generalized Hough transform can search for other shapes in an image.) Each boundary point in the original image specifies a curve in the transform space. The coordinates of high-intensity points in the transform domain correspond to the position and orientation of best-fit lines in the original image.

Region detection and labeling is a common operation for automated visual-inspection tasks. The purpose is to identify connected regions in an image and assign a unique label to each region. This is not trivial when objects in the

image are nonconvex (such as gears, blood cells, or alphanumeric characters). When receiving image data in raster-scan order, designers may not know if two or more regions, disjoint at the top of the image, connect later at the bottom of the image. Typical solutions to region labeling often require a second pass over the image to merge connected regions. Once this is done, unique labels can be assigned to each region.

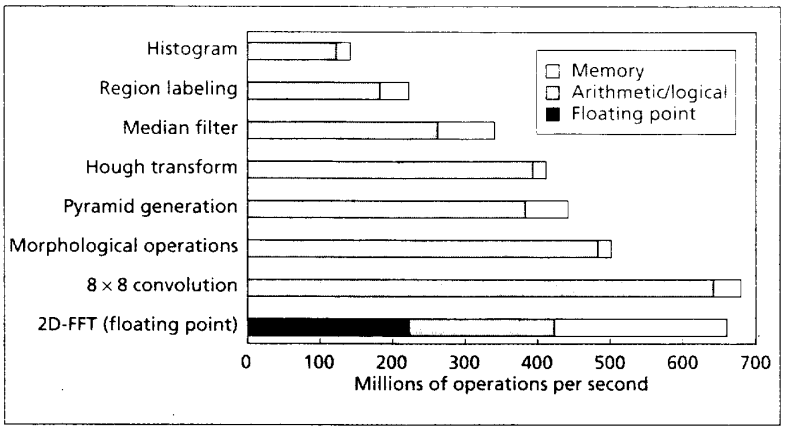

Figure 6. Approximate performance of image-processing tasks.

PERFORMANCE RESULTS

Computational properties, communications architectures, and required resources vary significantly from one application to the next. All the examples described here operate at the pixel clock rate of 10 MHz with 512 × 512 images. Many of the applications presented here have been implemented using a pipeline architecture. The pipeline accepts digitized image data in raster order, often directly from a camera, and produces output data at the same rate, with some latency. Many of these applications can be chained together to form higher level image-processing functions.

Prototype architectures

Figure 5 shows simplified block diagrams that illustrate the partitioning and communication architecture for representative tasks. For example, Figure 5a shows the architecture for the 3 × 3 median filter. This pipelined implementation produces output pixels at the same rate that input pixels are received, with a latency of less than the time required to receive three image rows. This requires the simultaneous access of nine neighboring pixels (produced by the gray shaded blocks labeled *Row stack*), which are presented to a parallel sorter in the gray shaded block labeled *Parallel sort*. The median value of the sorted list is then presented to the *Format output* block, which assembles the data for subsequent display on the monitor.

Performance evaluation

Conventional performance-benchmarking techniques are at best awkward when applied to custom computing machinery. Figure 6 graphically illustrates the computational performance of each of these tasks executing on the VTSplash platform. In this figure, the application name appears vertically to the left of the graph. The performance bar associated with each task consists of two or three components. The first component (arithmetic/logical) is an appraisal of the number of general-purpose operations performed on average per second. (These operations are likely to be found in the repertory of common RISC processors, such as Multiply, Xor, or Compare.) This number, when divided by the pixel clock frequency of 10 MHz, indicates the average number of the easily discernible arithmetic and logic function units (word parallel) active in each task.

The second component of the performance bar estimates the number of storage references (memory accesses) performed by the task per second. The third component represents the number of floating-point operations per second. All tasks, except for the 2D FFT application, use fixed-point operations. The pixel calculations for the 2D FFT task use custom-designed floating-point arithmetic. When combined, these three components provide a basis for quantifying the computational load of each task, as well as a rough estimate of the number of operations performed each second.

With VTSplash, the operating speed for an application is under the designer's control and depends upon critical path delays in the implementation. The Splash-2 processor features a programmable system clock that can be varied under software control from zero to 40 MHz. We developed the tasks to satisfy the minimum criterion of operating at the pixel data rate of 10 MHz. The designs were verified at this rate only, although some of these tasks may operate well beyond this clock frequency.

Processing rates

In addition to quantifying the number of operations per second, it is useful to consider how fast computations are performed relative to the 30-Hz frame rate of the input image. Some tasks (histogramming, median filtering, region labeling, Gaussian pyramid generation, and gray-scale morphological operations) are completed during one frame time. Others (8 × 8 convolution and Laplacian pyramid generation) require two image frames. The floating-point FFT implementation can completely process two 512 × 512 images per second. The time necessary to complete the Hough transform depends on the complexity of the image; the implementation shown in Figure 5d distributes equal portions of an input image to separate PEs that process in parallel.

Comparisons

Another method of benchmarking is to compare VTSplash operation with that of contemporary machines. We chose a general-purpose workstation (the Sun SparcStation-10). VTSplash applications run between 10

to 100 times faster than the same application written in C and executed on the SparcStation. Numerous commercial machines have been designed specifically for image processing. The Datacube MaxVideo 200[12] consists of several functional units carefully tuned to perform common image-processing tasks. For applications that are suited for its specific architecture, the MaxVideo 200 outperforms the VTSplash system. For example, the MaxVideo 200 can perform 8×8 convolution four times faster than our current VTSplash design. The motivation of the custom computing approach, therefore, is not to provide the fastest possible performance for a given task. As illustrated by VTSplash, the strength of this approach is a system that can be rapidly reconfigured to provide high performance for a wide range of tasks. The performance of application-specific systems diminishes quickly for tasks not directly supported in hardware.

RECONFIGURABLE COMPUTING PLATFORMS, such as Splash-2, can readily adapt to meet the communication and computational requirements of a wide variety of applications. By adding I/O hardware, we have demonstrated that general-purpose custom computing machines are well suited for many meaningful image-processing tasks. Such platforms are excellent testbeds for prototyping high-performance algorithms. The custom computing platform can also serve as

- a medium for hardware/software codesign, and
- a VHDL accelerator.

Our work on VTSplash continues in the area of high-level-language compilation for custom computing machines. We are investigating architectural enhancements for broadening the scope of tasks suitable for these machines and for streamlining automated partitioning and scheduling. Application development continues for image processing as well as for other problem domains including communications. ■

Acknowledgments

We are indebted to the graduate students who contributed to the VTSplash program. These include application developers Luna Chen, Robert Elliott, James Peterson, Ramana Rachakonda, Nabeel Shirazi, Adit Tarmaster, and Al Walters, along with the VTSplash hardware/software team of Brad Fross and Jeff Nevits. In addition, we appreciate the support and guidance of Jeffrey Arnold and Duncan Buell from the Supercomputing Research Center. This work was supported by a grant from the Institute for Defense Analyses.

References

1. J.M. Arnold, D.A. Buell, and E.G. Davis, "Splash 2," in *Proc. Fourth Ann'l ACM Symp. Parallel Algorithms and Architectures*, ACM, New York, 1992, pp. 316-322.
2. D. Gajski, *Silicon Compilation*, Addison-Wesley, Reading, Mass., 1988.
3. *The Programmable Gate Array Data Book*, Xilinx Inc., San Jose, Calif., 1994.
4. M. Gokhale and R. Minnich, "FPGA Computing in a Data-Parallel C," in *Proc. IEEE Workshop on FPGAs for Custom Computing*, IEEE CS Press, Los Alamitos, Calif., Order No. 93TH0535-5, 1993, pp. 94-101.
5. P. Athanas and H. Silverman, "Processor Reconfiguration through Instruction-Set Metamorphosis: Architecture and Compiler," *Computer*, Vol. 26, No. 3, Mar. 1993, pp. 11-18.
6. L. Agarwal and M. Wazlowski, "An Asynchronous Approach to Efficient Execution of Programs on Adaptive Architectures Utilizing FPGAs," *Proc. IEEE Workshop on FPGAs for Custom Computing*, IEEE CS Press, Los Alamitos, Calif., Order No. 5490-02U, 1994, pp. 111-119.
7. R.C. Vogt, *Automatic Morphological Set Recognition Algorithms*, Springer-Verlag, New York, 1989.
8. B. Jähne, *Digital Image Processing*, Springer-Verlag, New York, 1991.
9. A.L. Abbott, R.M. Haralick, and X. Zhuang, "Pipeline Architectures for Morphologic Image Analysis," *Machine Vision and Applications*, Vol. 1, No. 1, 1988, pp. 23-40.
10. P.J. Burt and E.H. Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Trans. Comm.*, Vol. COM-31, No. 4, Apr. 1983, pp. 532-540.
11. L. Abbott et al., "Finding Lines and Building Pyramids with Splash-2," *Proc. IEEE Workshop on FPGAs for Custom Computing*, IEEE CS Press, Los Alamitos, Calif., Order No. 5490-02U, 1994, pp. 155-163.
12. *The MaxVideo 200 Reference Manual*, Datacube Inc., Danvers, Mass., 1994.

Peter M. Athanas *is an assistant professor in the Bradley Department of Electrical Engineering at Virginia Polytechnic Institute and State University in Blacksburg, Virginia. He was also a senior design engineer in the Advanced Technologies Group for United Technologies Hamilton Standard in Windsor Locks, Connecticut. His research interests include custom computing machinery, logic synthesis, VLSI technology, and parallel processing. Athanas received a BS degree in electrical engineering from the University of Toledo, an MS degree in electrical engineering from Rensselaer Polytechnic Institute, and an ScM degree in applied mathematics and a PhD in electrical engineering from Brown University. He is a member of the IEEE Computer Society.*

A. Lynn Abbott *is an assistant professor in the Bradley Department of Electrical Engineering at Virginia Polytechnic Institute and State University. From 1980 to 1985, he was a member of the technical staff at AT&T Bell Laboratories in Holmdel, NJ. His research interests include computer vision, high-performance architectures for image processing, and artificial intelligence. Abbott received a BS from Rutgers University in 1980, an MS from Stanford University in 1981, and a PhD from the University of Illinois at Urbana-Champaign in 1990 (all in electrical engineering). He is a member of the IEEE Computer Society.*

Readers can contact the authors at the Bradley Department of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0111.