IEEE Transactions on Circuits and Systems for Video Technology, 5, April 1995, pp. 140-149.

Cheops: A Reconfigurable Data-Flow System for Video Processing

V. Michael Bove, Jr. and John A. Watlington
Media Laboratory, Massachusetts Institute of Technology
Room E15-322, 20 Ames Street, Cambridge MA 02139 USA
Phone: (617) 253-0334, Fax: (617) 258-6264
{*vmb, wad*}*@media.mit.edu*

ABSTRACT

The Cheops Imaging System is a compact, modular platform for acquisition, processing, and display of digital video sequences and model-based representations of moving scenes, and is intended as both a laboratory tool and a prototype architecture for future programmable video decoders. Rather than using a large number of general-purpose processors and dividing up image processing tasks spatially, Cheops abstracts out a set of basic, computationally intensive stream operations that may be performed in parallel and embodies them in specialized hardware. We review the Cheops architecture, describe the software system that has been developed to perform resource management, and present the results of some performance tests.

INTRODUCTION: RESEARCH CONTEXT AND BACKGROUND

Digital coding of video signals has become a topic of much interest in recent years, with applications ranging from videotelephony and "movies-on-demand" to terrestrial broadcast of high-definition television (HDTV) through existing spectrum allocations. In addition to the dimensions of data compression and robustness to noise, the Information and Entertainment Section of the MIT Media Laboratory has been exploring what we feel will ultimately be the most dramatic impact of digital video: the ability to decouple the form in which images originate from the form in which they are ultimately viewed.[1] This research has proceeded in two directions. First, we have extended "traditional" video coding techniques to allow decoding at resolutions and frame rates other than that of the originating camera ("open architecture" television) and to support sending various subsets of the bitstream over different channels ("scalability"). Second, we have for several years been investigating moving away from pixel- and frame-based image representations and toward descriptions of moving scenes that embody information about the structure of the scene; this promises both greater coding efficiency and the ability (whether in post-production or in interactive viewing) to manipulate the data in a physically or semantically meaningful way.

While an increasing number of VLSI integrated circuits dedicated to the encoding and decoding of specific video compression algorithms are becoming available, the potential need to decode a digital video bitstream in different ways and the likelihood that a number of different digital video representations may need to coexist in a single platform suggest that much encoding and decoding will not be done on dedicated, single-purpose circuits. Rather a digital video bitstream might carry enough information about its format that differing sorts of flexible decoders will be able to produce images from it; this approach will also permit a continuing evolution of video representations to accommodate higher-resolution images, higher-bandwidth channels, and advances in compression methods as they develop. In examining our current and projected research video processing needs, and as a first examination of hardware and software architectures for flexible digital video processing, we have developed a set of system requirements:

- The system should be able to do digital signal processing in conjunction with graphics at real-time or near-real-time rates. Where algorithmic complexity is too high for real-time computation, the system should be able to run asynchronously, storing the results for later display at synchronized real-time rates.
- The system should be easily programmable, without undue attention to hardware resource management or process synchronization.
- The system should support time-sharing among multiple processes.
- The system should be easily upgradable as processing requirements change or as technology improves.
- The system should be compact and relatively inexpensive, so that multiple copies can be built.

Examination of algorithms used in both waveform-based and analysis-synthesis image coding shows a limited number of extremely computationally intensive and often quite regular operations (matrix algebra, correlations, and convolutions, for example) along with less-demanding tasks that might easily be handled by typical microprocessors. A process might be described as a flow graph in which these basic tasks are connected together in a way which indicates the data dependencies.[2] Such a model is often referred to as a *data-flow* description of the process. The basics of data-flow computing were developed and described by Dennis in the 1970's.[3] While the initial investigations of hardware and software systems that follow the data-flow paradigm were aimed at general-purpose computing, more recently several groups have looked at the use of data-flow for digital signal processing applications.[4] An early example is the DDSP system.[5] Researchers have addressed the speed, multidimensionality, and synchronization requirements of digital video processing by building hardwired pipelines such as the Cytocomputer,[6] though such an architecture has very limited reconfigurability. The more flexible TIP [7] was based on differing sorts of processors connected around a ring bus; here the performance-limiting factor may be the bus bandwidth rather than the processors themselves. FLIP [8] instead used identical processors which could be microcode-configured into specific functionalities and connected together in a variety of ways through sixteen selectable buses. The Princeton Engine [9] is a large linear array of identical processors tightly coupled to a communication network as well as to analog-to-digital and digital-to-analog converters. The Datacube MaxVideo 20 and 200 [10][11] use a full crosspoint switch to connect an ALU, a convolution and statistics processor, memory, and analog-to-digital and digital-to-analog converters. The latter five systems are notable for the designers' explicit use of the data-flow model for programming. MX-1, a parallel image and signal processing system using identical processors communicating through a crosspoint, is described in [12]; here a combination of parallel LISP and C was used for programming, with no apparent application of data-flow techniques.

In recent years a number of single-chip video processors have emerged. While software tools may enable programmers to use a data-flow abstraction, at the hardware level many of these chips – like Matsushita's ISMP [13] and Texas Instruments' MVP [14] – are based on a standard multiple-instruction multiple-data (MIMD) parallel architecture. PADDI [15] and Philips' VSP [16] enable the construction of various synchronous processing pipelines by connecting their sets of arithmetic logic units through reconfigurable communication networks.

Cheops, the system we describe in this paper, perhaps has its closest direct predecessor in Fujitsu's IDATEN system.[17] IDATEN used a modified Benes Permutation Network to connect together several different types of processors in reconfigurable pipelines, where each processor performed a high-level operation such as finite-impulse-response filtering. Cheops is similarly a data-flow computer, in that algorithms are described as graphs involving operations by specialized stream processors and specifying the data dependencies among the operations.[1] A von Neumann general-purpose processor controlling everything, though, greatly enhances the flexibility and programmability of the system. Cheops differs from IDATEN in several areas, particularly in that resources are scheduled dynamically so that the same software will run on differently-equipped Cheops systems and so that multiple application programs can run concurrently. Also, Cheops allows buffers to be included in the data-flow graph, permitting slower-than-real-time processing when demand exceeds available resources.[2] These buffers also

[1] We use the term *stream processor* because the basic units of data on which Cheops operates are arrays of data accessed in a sequential fashion. Some authors make a distinction between such a stream-oriented data-flow model and one in which an individual datum is considered elemental. See for instance [18].

[2] A corollary property is the ability to perform portions of an algorithm at faster-than-real-time rates within the limits imposed by the buffer space available.

allow the general-purpose controlling CPU to gain access to intermediate results in the data-flow graph, enabling conditional data redirection or performance of portions of the algorithm for which specialized hardware is not available.

CHEOPS SYSTEM ARCHITECTURE

Cheops was intended to be a compact, reasonably inexpensive adjunct to a workstation. Along the way it has become a nearly complete workstation in its own right, with the limitation that it doesn't maintain its own filesystem but (transparently to the programmer) depends on the filesystem of its host computer. Since Cheops is a laboratory tool as well as a prototype reconfigurable video processor, it must be modular to meet future processing and input/output requirements. Cheops takes advantage of parallelism at several levels:

- Individual specialized processing units (the stream, or data-flow processors) typically comprise multiple parallel computing elements.
- Multiple stream processors operate simultaneously on one processor module.
- Multiple modules may reside in the backplane.

At the module level, the Cheops system architecture (Figure 1) resembles the "open architecture television receiver" proposed by Schreiber, *et al.*,[19] though that device had processing modules connecting between distinct input and output buses. Cheops allows for three different module types: processor, input/memory and output. These modules are interconnected by three linear buses. Two of these buses are capable of sustained high bandwidth (>100Mbyte/sec) transfer of pixel rasters (the Nile Buses) and the third is a 32Mbyte/sec bus for transfer of smaller units of data and system control (the Global Bus). Up to four of each type of module may be present in the system at one time, allowing a wide range in both system processing power and I/O capabilities.

The stream processing portion of Cheops supports two basic data types: 16-bit and 24-bit. On the Nile Buses, image data (pixels) being transferred to or from an I/O device are represented as three independent 8-bit component channels, packed into a 24-bit word. Each component channel in a pixel may be either unsigned or signed two's complement. Internal to a processing module, component channels are stored independently, using a 16-bit, fixed-point, signed two's complement representation referred to hereafter as a sample. Although the overall dynamic range required by several image processing algorithms (in particular subbands and transforms) exceeds that provided by a single 16-bit fixed-point representation, the range required by an individual block of data (such as a particular subband) is typically predictable. Individual processing devices may maintain much greater internal precision and either automatically or under external control renormalize results to fit properly within 16-bit samples. When necessary, Cheops' general-purpose microprocessor can handle other data types such as 32-bit integers and floating-point numbers.

CHEOPS PROCESSOR ARCHITECTURE

The philosophy behind the design of the Cheops processor module is to abstract out a basic set of computationally intensive operations required for real-time performance of a variety of desired applications. These operations are then embodied in specialized hardware provided with a very high throughput memory interface, and controlled by a general-purpose processor. Although image data tends to be large, it is accessed in a regular fashion, and the operations needed tend to work within a limited data window (often <1Ksample). Rather than storing all the data locally, the specialized processors operate upon one or more high speed streams of data. Stream processors are not required to input and output data simultaneously; some act only as destinations or as sources, while processors that require very large pipeline delays may do both but in separate transfer phases. The general-purpose processor can read and write control registers and local memory on the stream processors, as well as read results of calculations on the streams – the latter particularly the case on destination-only processors like block motion estimators. This access is via an 8-bit-wide control register datapath which supports transfer rates of 8Mbytes/sec.

The processor module comprises eight memory units communicating through a full crosspoint switch with up to eight stream processing units. As it was possible to build a full crosspoint switch capable of 40MHz operation with just four chips (LSI Logic L64270 devices), we chose to implement our interconnection network in this fashion rather than to use a more compact arrangement of switching elements. Each memory unit is made up of dual ported dynamic memory (VRAM) and a two-dimensional direct memory access (DMA) controller for transferring a stream of data through the crosspoint at up to 40Msample/sec. Specialized stream processors attached to the switch perform common mathematical tasks such as convolution, correlation, matrix algebra, block transforms, spatial remapping, or non-linear functions. These processing units are on removable sub-modules, allowing reconfigurability and easy upgrade. A general-purpose central processing unit (CPU) – an Intel 80960CA or CF, clocked at 32MHz – is provided for sequencing and controlling the flow of data among the different functional units, implementing the portions of algorithms for which a specialized processor is not available, and performing higher-level tasks such as resource management and user interface. Figure 2 shows a simplified view of the topology of a single "P2" processor module.

Because the switch is a full crosspoint, it is possible to connect a memory bank to another memory bank, to cascade processors,[3] or to send one stream to several destinations simultaneously.

The DMA controllers on each VRAM bank, called "flood controllers," are capable of relatively agile handling of one- and two-dimensional (and to a lesser extent, three-dimensional) arrays. When acting as a source, a flood controller can replicate or zero-pad independently in two dimensions, and when acting as a destination it can decimate. Hence fractional sampling-rate conversion can be performed in a single transfer through a filter stream processor. The functionality of a flood controller, especially in conjunction with the transpose stream processors discussed below, is thus similar to the multidimensional stream functions of the data-flow language Lucid;[20] see also [21]. The Nile Bus interface on non-processor modules is effectively a flood controller also, and Nile transfers are handled in a similar fashion to transfers within a processor module.

The processor module interfaces to the Nile Buses via the crosspoint switch. The pixel data being transferred is stored internal to the processor module in three separate memory units, one component channel per unit. As the pixel data is transferred to/from the Nile Bus, it passes

[3]The ability to cascade processors is limited by the maximum pipeline delay supported by the flood controller in the ultimate destination. See below.

through a color-space converter (a 3×3 matrix multiplier and lookup tables). This processor decouples the pixel color representation used for display/input from that used for processing, and may also be used to process data within a P2 module.

The actual data transfers through the crosspoint switch are synchronized by handshaking lines called "OK channels." Every flood controller and stream processor connects to every OK channel. In setting up a transfer, a process selects an unused OK channel and sets the involved source and destination flood controllers and stream processor to use this channel. The destinations are also informed of the pipeline delay associated with the datapath and stream processors. The OK channels are the logical AND of the OK outputs of all of the participants in a transfer. When all participants are ready to transfer data, the sources begin transferring. After an amount of time equal to the pipeline delay of the transfer, the destinations begin writing result data into memory. The current processor design has three OK channels, and thus at most three transfers may be taking place at one time on a processor module, though any number of memory banks and stream processors may be involved.[4]

The processor module communicates with the host computer using a SCSI (Small Computer Systems Interface) bus. Each processor module appears as a fixed disk device, therefore in most cases no special host device driver is needed; Cheops systems have proven to be compatible with existing UNIX disk device drivers on DECStations, Sun SPARCStations, IBM RS/6000 systems, and Silicon Graphics workstations. Two RS-232 serial ports are available, one for debugging and diagnostics, and the other for user interface devices such as serial knob boxes, mice, or touch-sensitive screens. If only a low-bandwidth host interface is desired, one of the ports may substitute for the SCSI connection.

STREAM PROCESSORS

All stream processors constructed to date conform to the generic model diagrammed in Figure 3. The electrical connections are fairly simple, and the signals involved have been discussed above. The register interface and control state machine are usually implemented as individual PALs, but if a large field programmable gate array (FPGA) is used as the main processor it may have enough extra terms available to implement these functional blocks as well. Because one removable submodule can hold two stream processors, it is possible for a single stream processor to have two inputs or two outputs, though in so doing it disables the other unit on the submodule when enabled. Stream processors may maintain as much internal state as necessary, though if the delay between input and output is very long and the processor can itself hold an entire transfer's worth of data, the processor operates in a two-phase fashion, where first it acts as only a destination, then as a source. Details on specific processors follow.

- *Transpose processor:* Although VRAM provides dense memory with a high throughput, rapid access is allowed only with an address stride of one, along adjacent samples in a linear address space. Operations using other strides (such as vertical or temporal processing) may be performed by first re-ordering the data stream using this stream processor. Each processor module has two of these units permanently installed, which also handle conversion to and from the square block format commonly used by devices such as block

[4]There is a separate OK channel associated with the processor's connection to the Nile Buses, but in practice there are generally too few memory banks available on a processor module for three other transfers to take place at the same time as a Nile Bus transfer to or from that module.

transform and motion estimation chips. Each is implemented as an FPGA and 128Kbytes of SRAM; buffers larger than 256×256 must thus be tiled, an operation which the resource management software handles automatically and transparently.

- *Filter processor:* The first filter module built contains four INMOS A110 FIR filter chips, and performs 21-tap filtering organized as either 1×21 or 3×7. As these chips are limited to 20MHz operation, it is necessary to sample-replicate the data at the source flood controller and to decimate by two at the destination. A cascade adder input is brought out so that a the output of another filter unit can be summed in to provide the software abstraction called a "dual filter."

  A faster and more flexible processor made of LSI Logic L64261 devices, variable-length shift registers, and a programmable operation sequencer, replaces the earlier filter processor. The new module adds full-speed operation, larger filter kernels (up to 64 taps), stream adding and multiplying, and the ability to multiply 1×4 vectors by 4×4 matrices for graphics rendering.

- *DCT processor:* This unit contains an INMOS A121 device and performs 12-bit, 8×8 discrete cosine transform (DCT) and inverse DCT. Given a second input, it can also do a post-add of an error signal and a pre-subtract of a prediction signal, to support predictive coders.

- *Motion estimator:* This unit is based on an LSI Logic L64720, and performs full-search motion estimation of an 8×8 block in a 16×16 window or, 16×16 in a 32×32 window. The entire sum-of-rectified-differences buffer is accessible over the control register bus if desired.

- *Color space converter:* This has been discussed earlier in connection with the Nile Buses, but it may also perform on-board processing of three data channels. In order to operate at full speed, it is implemented as a pair of Brooktree BT281 chips.

- *Remap/composite processor:* This stream processor – composed of Altera 7000-series EPLD's and 4Mbytes static RAM – deocdes both forward and backward motion compensation, performs image warping, and incorporates a 16-bit z-buffer for graphics compositing and hidden surface removal. Images of up to 2048×1024 are supported and vectors may be either relative offsets or absolute addresses.

- *Superposition processor:* This is a very specialized unit which can perform superposition of weighted, arbitrary 16-bit basis functions of up to 1024 samples in length. Its main application is in computational holography and non-orthogonal transforms. On one pass, it can place up to four basis functions on each output sample; multiple passes or multiple processors in cascade (set up automatically by the resource mangement software) permit any number of functions to be superimposed. It is based on one LSI Logic L64260 multiply-accumulate chip and 2Mbytes of SRAM.

- *Sequenced lookup table:* Besides acting as a 16-bit lookup table, this processor was also designed to decode vector quantization with block sizes up to 4×4. Its functionality has been absorbed into the state machine processor described below.

- *State machine:* This highly reconfigurable device is intended to provide bitstream operations, variable-length encoding and decoding, floating-point arithmetic, and other functions not available in other processors. It very closely follows the generic model in Figure 3, with the processor portion comprising both a PowerPC 603 and an Altera 8000-series in-circuit-reprogrammable gate array. The memory portion contains 4Mbytes of SRAM. An application program (or more precisely the resource manager) configures this device by loading a specific logic function into the EPLD, or by loading software for the CPU, which operates either on a continuous stream or in two-phase fashion.

Currently supported stream processors are diagrammed in Figure 4; note that shown are *logical* processors (as they would appear to a programmer) rather than physical devices – the "dual filter," for example, is actually created by enabling an additional datapath internal to a submodule containing two FIR filter processors.

INPUT AND OUTPUT

We have built two different output modules, both providing more than one framestore of direct mapped (three channels of 8 bits) image memory. The first is software-configurable to produce a broad variety of video formats, from 640×480, 30Hz interlaced (RS-170A), up to 1280×1024, 66Hz progressively scanned. Both digital and analog outputs are provided, and the video may be synchronized to an external source. The second output module generates analog video for a 2048×2048, 60Hz progressively scanned display.

The memory/input module provides a Cheops system with a large memory buffer (64Mbytes to 8Gbytes) on one card, accessible at full bandwidth over the Global Bus and both Nile Buses. It also allows a single remote device, located on a daughter card, up to 400Mbytes/sec access into or out of the memory. We have built a high-speed interface (HiPPI) daughter card, and are developing another card which accepts various analog and digital video formats. The memory in both the output and memory/input modules is triple-ported to allow simultaneous full speed data transfer to the same block of memory over the Nile Bus, the Global bus and the I/O port of the module.

OPERATING SYSTEM AND COMMUNICATIONS

A Cheops system acts as a peripheral to one of a variety of host computers; the host provides a filesystem/network interface and a platform for software development and cross-compilation. Software is written in a high level language (C), compiled, then executed on the general-purpose processor. An operating system (Magic7) resident on each processor module provides overt process time-sharing, memory management, and message passing. Unlike most multitasking operating systems, Magic7 permits processes to specify the points at which they may be swapped out (by an explicit call to `proc_sleep()` or by performing an input/output function call), accommodating time critical tasks. A message passing facility may be used to commmunicate among multiple processes executing on a single processor, or on other processor modules. Processes executing on the host computer may also use this facility to communicate with processes executing on the Cheops system. Although Cheops is solely a target device on the SCSI bus, a resident client/server protocol allows Cheops programs downloaded and executed from the host to access files on the host filesystem, and to print and input on the host's console devices, using the same function calls as any ordinary host program under UNIX.

Since the flood controllers model memory as two-dimensional, we have extended the conventional `malloc()` function: allocation of two-dimensional arrays within memory is permitted, as is specifying a specific memory bank (for instance, different color components of an image should be stored in different banks in order to allow maximum parallelism).

RESOURCE MANAGEMENT

Lee has made a distinction among several types of schedulers for data-flow processors;[4] it is possible to use Cheops' stream processors in what he calls a fully static fashion, compiling code that sets up the crosspoint switch in a specific way and assigns tasks to specific hardware devices. Indeed, several simple programs (mostly for diagnostic purposes) were written to operate in this manner. However, because the stream processors are on removable submodules and thus the configuration may change from time to time or from system to system, this solution doesn't seem desirable. Further, static scheduling becomes nearly unworkable in a multitasking environment like Magic7, where several processes may be competing for the same resources. In order to alleviate these problems, we have developed a resource management daemon named NORMAN.[22] It manages the system resources, such as the currently installed stream processors and the Nile Buses, maintaining scoreboards and queues for each. A user program – a process which handles file I/O, interaction, and so forth – will pass a linked-list data structure describing the data-flow portion of the algorithm to the daemon. An individual stream transfer making up a sub-part of the data-flow graph occurs when all prerequisite data, an appropriate stream processor, and a destination are available. Transfers may optionally be made dependent upon a real-time clock (counting video frames on a selected input or output module) to permit process synchronization with video sources or displays. The programmer may associate a callback routine with any individual transfer, to inform the invoking process of the status of the data-flow processing.

NORMAN can be said to be a fully dynamic scheduler, as it assigns tasks to specific processors only at run time. A still finer distinction needs to be made, though: does the daemon assign physical devices to elements in the entire data-flow graph as soon as received, or does it assign devices as they become needed and available? The initial implementation of NORMAN did the former, a strategy which entailed less dynamic decisionmaking, but ultimately proved less efficient as well as not amenable to a multitasking environment. The current version of NORMAN does all scheduling on the fly. We will compare performance of these two approaches in a later section of this paper.

Creating the linked list describing the data-flow graph is the only unconventional task facing the programmer, and a number of successively simpler methods have been made available. Consider the simple case of a taking the 8×8 DCT of the difference of corresponding blocks in two arrays of values (Figure 5), as might be done in a predictive coder. The streams first pass through a transpose operation to rearrange the blocks into successive samples in each stream, then another transfer passes these through the DCT processor, which is set to "presubtract" mode.[5] The lowest-level, most esoteric, and most bug-prone option is to manage the pointers in the transfer elements directly. Below is the type declaration for these elements; for obvious reasons we won't include the actual C code for this first approach.

[5] Actually, we are allowed to connect the DCT processor right to the transpose engines, but for generality we will maintain the intermediate buffers.

```
typedef struct QueueElem {
        struct QueueElem *next;     /* next queue element */
        struct QueueElem *prev;     /* previous queue element */
        int num_deps;               /* # of backward dependencies */
        DependList *back;           /* list of backward dependencies */
        int num_dests;              /* # of forward dependencies */
        DependList *forw;           /* list of forward dependencies */
        PipeDesc *pipe;             /* subtransfer descriptor */
        int miniq_io;               /* flags the QueueElem as an input, output,
                                       both or neither of the miniqueue */
        long timestamp;             /* timestamp transfer should occur */
        int start_phase;            /* nile transfer start phase */
        void (*proc)();             /* pointer to completion routine */
        long user_data;             /* field to hold user data */
        int copied;                 /* RMAN management variables */
        struct QueueElem *copy;
} QueueElem;
```

Simple C functions to create, configure and connect the desired elements provide a more palatable interface:

```
/* representative functions prototyped in cheops_rman.h: */

extern QueueElem *RmanDCT(short *src, short *vector,
                          unsigned xdim, unsigned ydim,
                          unsigned mode, short *dst, void (*done)());

extern QueueElem *RmanTranspose(short *srcstart, unsigned xsize, unsigned ysize,
                                short *dststart, void (*done)());

/* in the user's program: */

QueueElem *my_dct, *my_transpose_1, *my_transpose_2;

my_transpose_1 = RmanTranspose(SRCBUF1, XSIZE, YSIZE, TMPBUF1, transpose_callback);
my_transpose_2 = RmanTranspose(SRCBUF2, XSIZE, YSIZE, TMPBUF2, transpose_callback);
my_dct = RmanDCT(TMPBUF1, TMPBUF2, XSIZE, YSIZE, DCT_PRESUB, DSTBUF, dct_callback);
RmanTransposeSetMode(my_transpose_1, TENG_DCT_8_MODE);
RmanTransposeSetMode(my_transpose_2, TENG_DCT_8_MODE);
RmanConnect(my_transpose_1, my_transpose_2, NULL);  /* connect parallel mini-queue */
RmanDepend(my_transpose_1, my_dct, NULL);   /* set dependencies */
RmanConnect(my_transpose_1, my_dct, NULL); /* connect DCT to the transposes */
```

```
/* Now get a pipeline ID from NORMAN, and then execute (return value
 * checking deleted for brevity). Reference the whole thing by the
 * head of the queue: my_transpose_1.
 */
pipeid = RmanPipeIdAllocate();
returnval = RmanExecute(pipeid, my_transpose_1);
```

The highest-level access currently available is a two-step process. The graph is described in a locally-developed language called OOPS,[23] which a pre-compiler expands into the C structures for inclusion into a skeleton C program.

```
SRCBUF1 = buffer[XSIZE, YSIZE]()
SRCBUF2 = buffer[XSIZE, YSIZE]()
TMPBUF1 = transpose[TENG_DCT_8_MODE](SRCBUF1)
TMPBUF2 = transpose[TENG_DCT_8_MODE](SRCBUF2)
DSTBUF = dct[DCT_PRESUB](TMPBUF1, TMPBUF2)
```

A graphical interface would most easily be added at the OOPS level.

HARDWARE DESCRIPTION

The typical Cheops system (Figure 6) is enclosed in a box 45cm by 60cm by 22cm, containing one P2 processor module and one or more O1 output modules and M1 memory/input modules; it consumes approximately 400W of power. Modules for Cheops have a standard size of 36cm by 40cm, and may be up to 4cm thick. Three 96-pin connectors connect the modules to the system backplane. This backplane carries the Global and Nile Buses, and provides power to the modules. The backplane is terminated at one or both ends with active termination, and may be up to 50cm in length. System clocks are provided by the backplane.

The vast majority of the integrated circuits used throughout Cheops are CMOS or BiCMOS. One notable exception is the Bus Transceiver Logic (BTL) used for all backplane signals, due to its reduced crosstalk, increased noise immunity and lower driver impedance. Much of the control logic is implemented using field programmable gate arrays (Altera MAX5000 and MAX7000 series), and so far no custom or semi-custom integrated circuits have proven necessary.

The Cheops P2 Processor Module consists of one main printed circuit board and five daughter printed circuit cards. Two (16cm by 14cm) memory cards each contain two memory banks and their control logic, replicating the four memory banks located on the motherboard. Stream processors (Figure 7) are located on separate 10cm by 14cm cards, two to a card.

APPLICATIONS AND PERFORMANCE EVALUATION

We have written a number of user applications for Cheops, including real-time interactive signal processing utilities for image sequences, image compression and decompression utilities, and renderers for moving computer-graphics particle databases. Reference [24] describes a script-driven, flexible decoding pipeline for producing images from 2-D and 3-D model-based coders, while [25] discusses the use of the superposition processor in computation of holograms.

We have measured the system's performance on a variety of simple tasks.[6] In order to evaluate the performance of NORMAN, we will consider a signal processing task: convolving a 21-tap FIR filter kernel both horizontally and vertically with an already-in-RAM RGB image sequence, and displaying the results. The data-flow graph contains three parallel sets of transpose-filter-transpose-filter, all connecting to one display operation across the Nile Bus – though in the actual benchmark presented here, the images were read from disk already transposed, so the first transpose operation may be dispensed with.

We compared run-time scheduling of the entire queue of transfers with on-the-fly scheduling of each transfer (discussed in a previous section), for three sizes of images, and for varying numbers of available filter stream processors. The results are summarized in Table 1. The theoretical maximum frame rates given assume no scheduling or setup overhead, and are an estimate of the performance of the stream processors and Nile Bus if simply hard-wired together. On-the-fly scheduling, because it involves more time-consuming computation before starting each transfer, can in some cases incur a small performance degradation in the maximum rate, though it is clearly superior in the two-processor case. We note that as the transfer sizes increase, the processing rate becomes significantly closer to the theoretical maximum, as the scheduling and setup overhead is amortized over larger transfers. The major performance limiting factor in the assigned task seems to be the presence of only two transpose units on the processor module (additional plug-in transpose processors can be installed if needed).

When considering the performance figures observed in the first table, the reader must note that the filter processors used for the foregoing test (which were limited to running at half the effective transfer speed of the flood controllers) have been superseded by newer submodules that operate at twice the former clock rate. The performance of these new filter processors, scheduled on-the-fly, is given in Table 2. The fact that overall performance does not quite double emphasizes the need for additional transpose processors, and illustrates the effects of the overhead in the resource management daemon.

As one of the reasons for on-the-fly dynamic scheduling is to support multitasking, it should be instructive to test the performance while running more than one independent process using the faster filter processors (literally the same executable code, invoked repeatly with different command-line arguments). Results of this experiment are given in Table 3.

A single DCT stream processor computes an 8×8 DCT/IDCT pair on 256×256 images (as in an MPEG encoder) at 81.0 frames/second. A single motion estimator performs 16×16 motion estimation on the same images at 6.6 frames/second. The corresponding theoretical maximum speeds are 115.2 and 55.8; the inefficiency in the latter case comes because the current motion estimator cannot accept input in a continuous stream, and enormous overhead is incurred in starting and stopping the stream every 256 samples. A typical Cheops system, then, cannot perform motion-compensated DCT video encoding in real time, but is very useful in accelerating the process. For computer graphics applications, a single multiply-accumulate processor renders 6.7 Msamples/second (theoretical maximum 8.0) and a single remap/composite processor remaps and z-buffers 3.7 Msamples/second (theoretical 3.9). Without z-buffering the remap processor runs at double the quoted speed.

CONCLUSIONS AND FUTURE WORK

[6] All figures given in this section are for a dataflow clock of 32MHz.

As of this writing, eight Cheops systems are in operation, and the fundamental hardware and software principles have proven sound. Dynamically-scheduled data-flow systems (or more accurately, hybrid systems) appear feasible solutions for compact, relatively inexpensive video processing applications. Nevertheless, as this work is being conducted in a laboratory environment, the underlying system undergoes constant improvement. We are attempting to increase the efficiency of the resource manager, and plan to allow cooperative operation among NORMAN daemons running on different processor modules. We feel that data-flow architecture is a feasible model for future inexpensive, programmable digital video decoding hardware, and have begun design work on a single DSP/graphics processor chip based on the lessons we have learned from the Cheops system.

ACKNOWLEDGMENTS

The authors wish to thank our co-workers who have contributed their time, efforts, and ideas to this system, in particular Shawn Becker, Brett Granger, Gregory Hewlett, Araz Inguilizian, and Irene Shen. The research described in this paper has been supported by the members of the Television of Tomorrow and Movies of the Future research consortia.

REFERENCES

[1] V. M. Bove, Jr., "Hardware and Software Implications of Representing Scenes as Data," *Proc. ICASSP-93*, pp. I-121–I-124, 1993.

[2] Arvind and D. E. Culler, "Managing Resources in a Parallel Machine," in *Fifth Generation Computer Architectures*, J. V. Woods, editor, Elsevier Science Publishers, New York, 1986, pp. 103-121.

[3] J. B. Dennis, "Dataflow Supercomputers," *Computer, 13,* no. 11, pp. 48-56, November 1980.

[4] E. A. Lee, "Static Scheduling of Data-flow Programs for DSP," in *Advanced Topics in Data-Flow Computing*, J.-L. Gaudiot and L. Bic, editors, Prentice-Hall, Englewood Cliffs NJ, 1991, pp. 501-526.

[5] E. Hogenauer, *et al.,* "DDSP – A Data Flow Computer for Signal Processing," *Proc. 1982 Intl. Conf. on Parallel Processing,* pp. 126-133, 1982.

[6] R. M. Lougheed and D. L. McCubbrey, "The Cytocomputer: A Practical Pipelined Image Processor," *Proc. 7th Ann. Symp. on Computer Arch.,* pp. 271-277, 1980.

[7] T. Temma, *et al.,* "Dataflow Procesor for Image Processing," *Proc. Intl. Symp. on Mini and Microcomputers, 5,* no. 3, pp. 52-56, 1980.

[8] P. Gemmar, *et al.,* "FLIP: A Multiprocessor System for Image Processing," in *Languages and Architectures for Image Processing,* M. J. B. Duff and S. Levaldi, editors, Academic Press, London, 1981, pp. 245-256.

[9] D. Chin, *et al.,* "The Princeton Engine: A Real-Time Video System Simulator," *IEEE Trans. Consumer Electronics, 34,* no. 2, pp. 285-297, 1988.

[10] Datacube, Inc., "Max Video 20," document DS599-1.0, Danvers MA, 1991.

[11] Datacube, Inc., "Max Video 200 Technical Specification," document DS600-1.0, Danvers MA, 1992.

[12] A. M. Aull, *et al.,* "Real-Time Radar Image Understanding: A Machine-Intelligence Approach," *Lincoln Lab. J., 5,* no. 2, pp. 195-222, 1992.

[13] M. Maruyama, *et al.,* "A 200 MIPS Image Signal Multiprocessor on a Single Chip," *1990 Digest of Technical Papers of the International Solid State Circuits Conference*, pp. 122-123, 1990.

[14] Texas Instruments, Inc., "TMS320C80 Technical Brief: Multimedia Video Processor (MVP)," Houston TX, 1994.

[15] D. C. Chen and J. M. Rabaey, "A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths," *IEEE Journal of Solid-State Circuits, 27*, no. 12, pp. 1895-1904, December 1992.

[16] A. H. M. van Roermund, *et al.,* "A General-Purpose Programmable Video Signal Processor," *IEEE Trans. Consumer Electronics, 35*, no. 3, pp. 249-258, August 1989.

[17] S. Sasaki, *et al.,* "IDATEN: Reconfigurable Video-Rate Image Processing System," *Fujitsu Sci. Tech. J., 23,* no. 4, pp. 391-400, December 1987.

[18] D. Skillicorn, "Stream Languages and Data-Flow," in *Advanced Topics in Data-Flow Computing*, J.-L. Gaudiot and L. Bic, editors, Prentice-Hall, Englewood Cliffs NJ, 1991, pp. 439-454.

[19] W. F. Schreiber *et al.,* "Channel-Compatible 6-MHz HDTV Distribution Systems," *SMPTE J., 98*, no. 1, pp. 5-13, January 1989.

[20] E. A. Ashcroft, "Proving Assertions about Parallel Programs," *Journal of Computer and Systems Science, 10*, no. 1, pp. 110-135, January 1975.

[21] E. A. Lee, "Representing and Exploiting Data Parallelism using Multidimensional Dataflow Diagrams," *Proc. ICASSP-93*, pp. I-453–I-456, 1993.

[22] I. J. Shen, "Resource Manager for a Video Processing System," SM thesis, MIT, 1992.

[23] K. L. Evanco, "A Compiler for an Object Oriented Parallel Stream Processing Language," SB thesis, MIT, 1993.

[24] V. M. Bove, Jr., *et al.,* "Real-Time Decoding and Display of Structured Video," *Proc. IEEE ICMCS '94*, pp. 456-462, 1994.

[25] M. Lucente, *et al.,* "A Hardware Architecture for Rapid Generation of Electro-Holographic Fringe Patterns," *Proc. SPIE Practical Holography IX*, (in press) 1995.

FIGURES AND TABLES

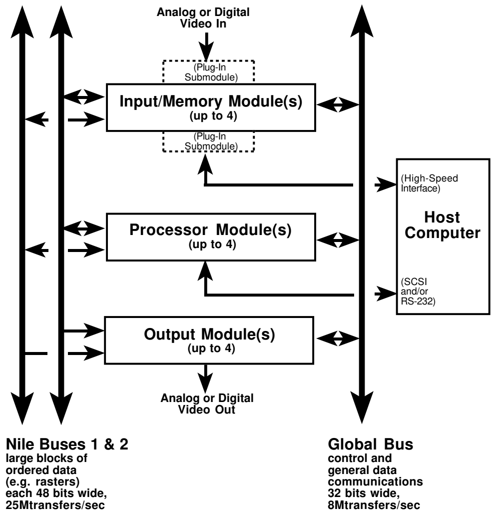

Figure 1: The Cheops system consists of three different types of modules connected by three buses.

	run-time			*on-the-fly*		
	256×256	512×512	768×768	256×256	512×512	768×768
1 processor	20.0	6.9	3.5	18.8	6.9	3.5
2 processors	24.1	8.0	4.0	29.5	9.4	4.7
3 processors	30.0	9.6	4.7	30.0	9.4	4.7
4 processors	30.0	9.6	4.7	30.0	9.4	4.7
theoretical max	66.1	17.9	8.1	66.1	17.9	8.1
best % of theoretical max	45.4%	53.6%	58.0%	45.4%	52.6%	58.0%

Table 1: Performance of the Cheops system (displayed frames per second) on two-dimensional separable filtering of color image sequences with two different resource scheduling strategies and various numbers of filter processors available. Older (half-speed) filter processors were used in this experiment. Theoretical maximum rates assume only two transpose processors and no scheduling or setup overhead.

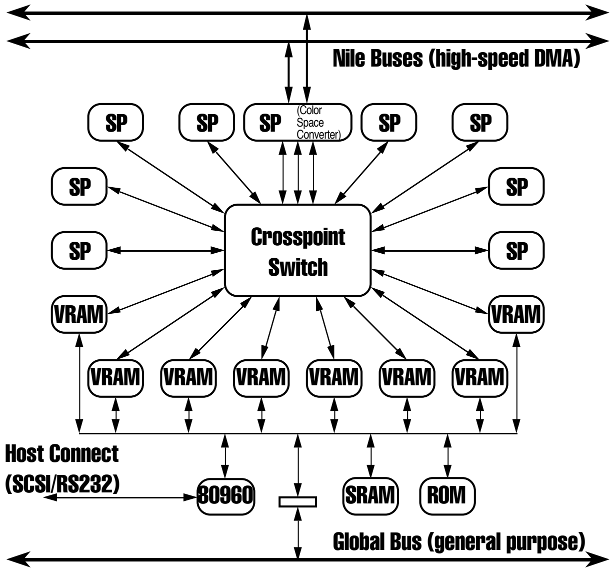

Figure 2: Highly simplified diagram of the Cheops processor. Blocks labeled **SP** represent stream processors. Not shown is the control register datapath by which the 80960 processor configures and controls the stream processors. Note that the "bidirectional" datapaths to the crosspoint are actually separate 16-bit input and output paths; note also that a stream processor may in some cases accept data from two input streams or produce two output streams.

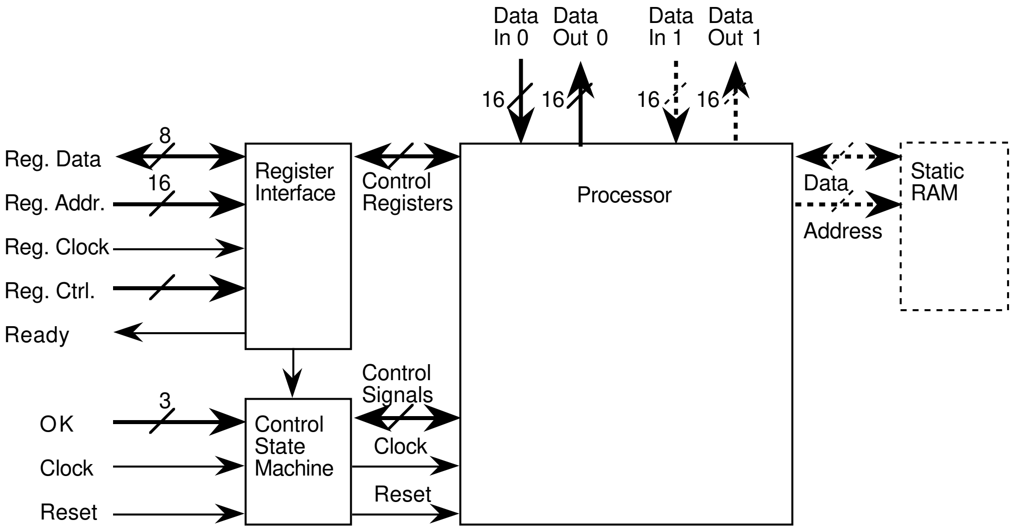

Figure 3: Block diagram of a typical stream processor. The second stream datapath exists only on two-input and/or two-output processors. The static RAM is needed only on two-phase devices or those that require local storage of information such as lookup tables. Depending on the intended function of the unit, the block labeled "processor" may be one or more programmable gate arrays, DSP processing devices, or ASICs.

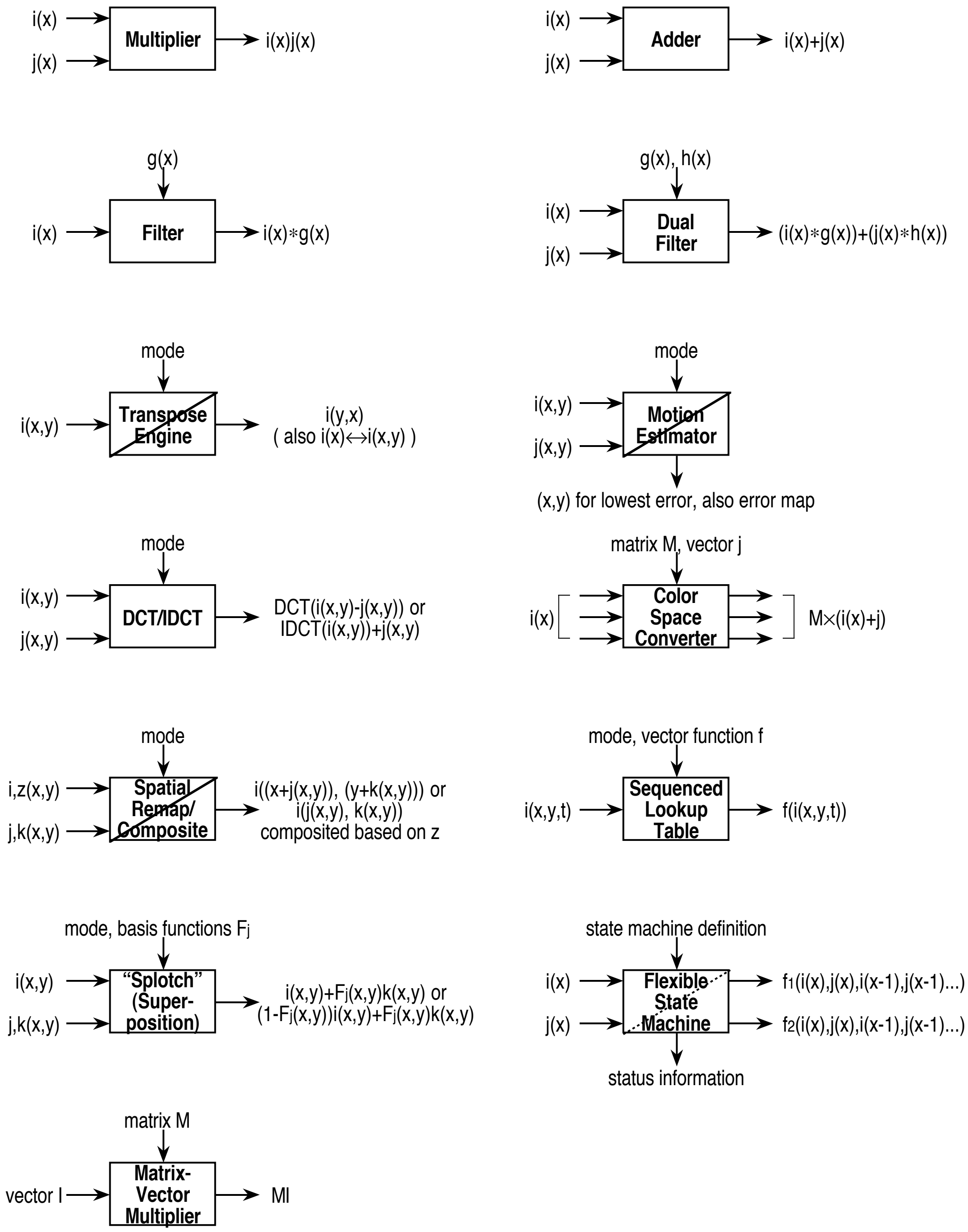

Figure 4: Currently supported stream processing operations. Horizontal arrows represent the stream datapath (through the crosspoint), while vertical arrows are the lower-speed control register interface. Processors marked with a diagonal slash are used in a two-phase fashion.

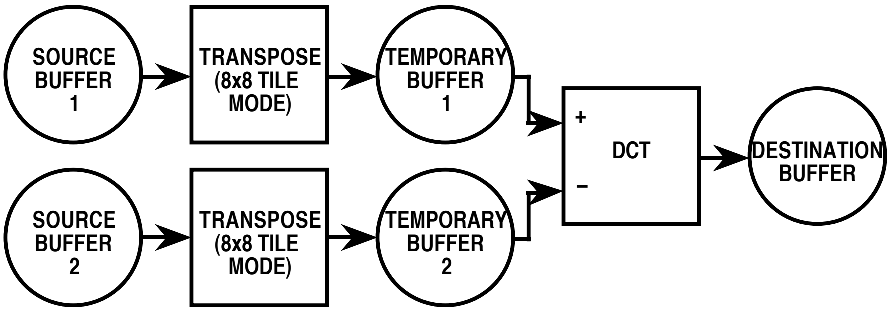

Figure 5: A simple task – taking the discrete cosine transform of the differences between two two-dimensional blocks of data – represented as a data-flow diagram.

Figure 6: Nine-slot Cheops system in a floor-standing cabinet, with memory (left) and processor (right) modules removed. Smaller systems occupy horizontal card cages and can be placed underneath a video monitor.

	256×256	512×512	768×768
1 processor	34.3	12.0	6.1
2 processors	43.8	14.4	7.8
3 processors	53.3	15.1	8.1
4 processors	53.3	15.1	8.2
theoretical max	84.6	21.8	9.8
best % of theoretical max	63.0%	69.2%	83.7%

Table 2: On-the-fly portion of preceding performance experiment repeated with faster filter processors.

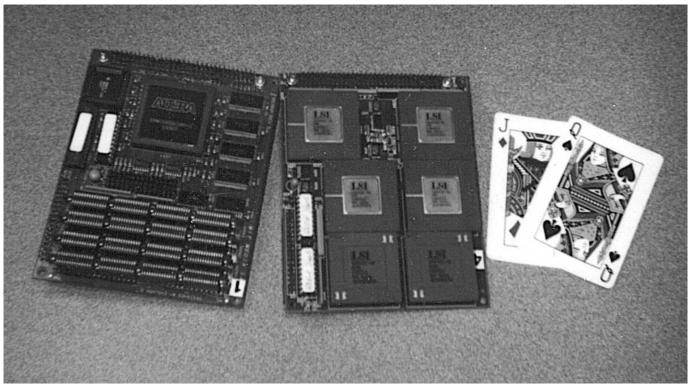

Figure 7: Stream-processor submodules, each containing two units (from left): remap/composite and DCT, dual multiply-accumulate/filter, playing cards for size comparison.

	one 256×256	two 256×256	three 256×256	four 256×256
rate (fps)	53.3	26.9	17.7	13.3
theoretical max	84.6	42.3	28.2	21.2
% of theoretical max	63.0%	63.6%	62.8%	62.7%

Table 3: Performance of scheduled-on-the-fly Cheops system (displayed frames per second) for one, two, three, and four simultaneous executions of preceding 256×256 filtering task. Four filter processors and two transpose processors are available.