Scalable Video in a Multiprocessing Environment

by

Gregory James Hewlett

B.S., Electrical and Computer Engineering, Rice University 1990

B.A., Art/Art History, Rice University 1990

Submitted to the Media Arts and Science Section, School of Architecture and Planning, in Partial Fulfillment of the Requirements for the degree of

Master of Science

at the Massachusetts Institute of Technology

February 1993

Author _____

Gregory J. Hewlett
September 16,1992

Certified by _____

V. Michael Bove
Assistant Professor of Media Technology
MIT Media Laboratory
Thesis Supervisor

Accepted by _____

Stephen A. Benton
Chairman
Departmental Committee on Graduate Students

Scalable Video in a Multiprocessing Environment

by

Gregory James Hewlett

Submitted to the Media Arts and Science Section, School of Architecture and Planning, on September 16, 1992 in partial fulfillment of the requirements for the degree of Master of Science at the Massachusetts Institute of Technology

Abstract

This thesis furthers research in the area of Open Architecture Television, specifically the feature of scalable video. A parallel processing software operating system was designed for the Cheops Image Processing System with the goal of accommodating the needs of a scalable video implementation. Several approaches to dividing the task of scalable video for implementation in a parallel environment were analyzed. It was found that a spatial division of the image was desirable for our system. A scalable video system was then implemented within this new multiprocessing Cheops environment using the spatial division approach.

Thesis Supervisor: V. Michael Bove

Assistant Professor of Media Technology, MIT Media Laboratory

Reader _____

V. Michael Bove
Assistant Professor of Media Technology
MIT Media Laboratory
Thesis Supervisor

Reader _____

David L. Tennenhouse
Assistant Professor, Electrical Engineering and Computer Science
MIT Laboratory of Computer Science

Reader _____

Kenneth W. Haase, Jr.
Assistant Professor of Media Technology
MIT Media Laboratory

Reader _____

Stephen A. Benton
Professor of Media Technology
MIT Media Laboratory

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis furthers research in the area of Open Architecture Television, specifically the feature of scalable video. A parallel processing software operating system is developed for the Cheops hardware system with the goal of accommodating the needs of a scalable video implementation. The existing multitasking operating system is modified to allow for multiple processor modules while retaining transparence to those applications designed in the single processor, multitasking system. A scalable video algorithm is then implemented within this multiprocessing Cheops environment to demonstrate the usefulness of parallel processing to a video display system. The specific algorithm implemented in this thesis is single level subband decoding.

1.1 Scalable Video

The direction of television engineering has taken a sharp turn over the last several years. Video is no longer described simply by the number of lines in a display. Many television formats proposed to the FCC as the new television standard contain the feature known as *scalability*. A scalable video format is that which is not confined to a unique resolution, framerate, or bandwidth.

The Entertainment and Information Group at the MIT Media Lab has proposed the idea of Open Architecture Television(OAR) as the basic television system for the display of scalable video. In the Open Architecture approach to television research, television is described by the number of features or degrees of freedom in the system.[8]

The Entertainment Information Group has conducted many experiments in developing scalable video. (see, for example [17]). In keeping with the MIT Media Lab tradition of creating the tools with which we can conduct research, the Cheops Imaging System, a modular processor for scalable video decoding, has been developed.

Cheops was designed as a multiprocessing engine because scalable video lends itself to a parallel environment. In this thesis, I develop the parallel processing operating system to accommodate the special needs of a scalable video system. My approach was a direct adaptation of the previous, single module operating system to a multiple processing system. I then implement a scalable video algorithm, spatial subband decoding, within the system. The parallelization of this implementation is shown to improve efficiency as the number of modules used within the Cheops system increases.

1.2 Previous Work

In 1990, Watlington and Bove began developing the Cheops Imaging System in order to demonstrate of the goals of Open Architecture Television.[20] Cheops fulfills this by providing an environment which readily accommodates a video representation which is scalable, extensible, and interoperable. Scalability means that the signal is coded in such a way that the channel can vary in bandwidth or the receiver can vary in decoding capacity, in order that images with differing picture quality can be represented by the same signal. Extensibility means that the representation can expand as higher quality images become

available. Interoperability means that the video parameters of the receiver (such as resolution and frame rate) need not match those of the source.

Because of the extensive demands on such a system, we have found that general-purpose computers have not been able to provide an environment where Open Architecture Television techniques can be tested. Thus, Cheops, a compact, bus-oriented platform, was developed to meet these demands. One feature of Cheops which makes it adaptable to such an environment is its flexible expandability, consisting of a number processor modules, output modules, and/or input modules.

In his 1991 MIT Senior Thesis, Allan Blount completed work on an operating system for a single processor module driving a single output module.[7] This operating system, called Magic Seven, allowed for multitasking of up to eight processes through the use of process swapping. Magic Seven included a monitor, useful for low level debugging and performing multi-tasking process control operations.

The Magic Seven operating system was expanded to allow interprocess communication, including advanced communication with the host by Hewlett and Becker.[12] Application software has been developed based upon the client-server model[6].

A resource manager for Cheops was recently completed by Irene Shen.[16] This resource manager schedules and monitors application tasks for each stream resource. A stream resource is a special purpose device located on a Cheops processor module. These devices include a matrix transpose operation, various filter chips, a color space converter, and a motion compensation device. The resource manager was show to greatly reduce the complexity of using the stream processing system for video decoding.

My thesis will update the Magic Seven operating system to accommodate parallel scalable video algorithms residing on multiple processor modules. Furthermore, parallel

algorithms for scalable video decoding will be investigated and demonstrated within the Cheops System.

1.3 Thesis Overview

The following chapter will describe the concept of scalable video. One method of implementing scalable video, subband decoding, is discussed. A description of the Cheops Hardware and Software environment is presented in Chapter 3. The development of a parallel operating system is described in Chapter 4. Included in this chapter is a discussion of the system requirements needed in the implementation of scalable video decoding algorithms. A general description of the operating system developed is also presented. Finally, I describe how such an operating system is implemented. Chapter 5 contains a discussion of several approaches to dividing the algorithm of scalable video into parallel tasks. In chapter 6, I describe an actual scalable video application which I have implemented. A full description of the algorithm is followed by details of implementation followed by a performance evaluation of the system. The final chapter is a conclusion of my work, including suggestions for further optimization and possible further extensions to this thesis.

An appendix are also included. It contains a few notes for software developers who wish to use multiprocessing on Cheops.

Chapter 2

Scalable Video

This chapter discusses the concept of Scalable Video. A definition is given as well as a description of one coding scheme which is scalable: subband coding.

2.1 Definition

Scalability is perhaps the most important of the degrees of freedom allowed in an Open Architecture environment. Scalability simply means that "the picture resolution can vary smoothly with channel bandwidth, processing capability, or display line count...also that it is not specified in terms of a fundamental frame rate of 24, 50, or 60 Hz. Instead, it is represented as a set of spatiotemporal components from which displays of almost any frame and line rate can be constructed"[5] These variables are all adjusted by the receiver, not defined by the transmitter. For a complete discussion of scalable video, see [8].

In a scalable television system, video data can be visualized as a flowing river of information, from which each receiver can retrieve as much or as little information as it wants. (see Figure 1.). If a receiver has limited processing ability or its channel has limited bandwidth, it has the option to only retrieve the necessary data needed to construct a signal of smaller resolution or lower framerate than the original transmitted signal. So it is not

necessary to decode the entire transmitted signal to obtain an image at lower resolution than the source. In other words, each receiver need not drink the entire river, discarding the unneeded portions. Each receiver only drinks the amount it needs.

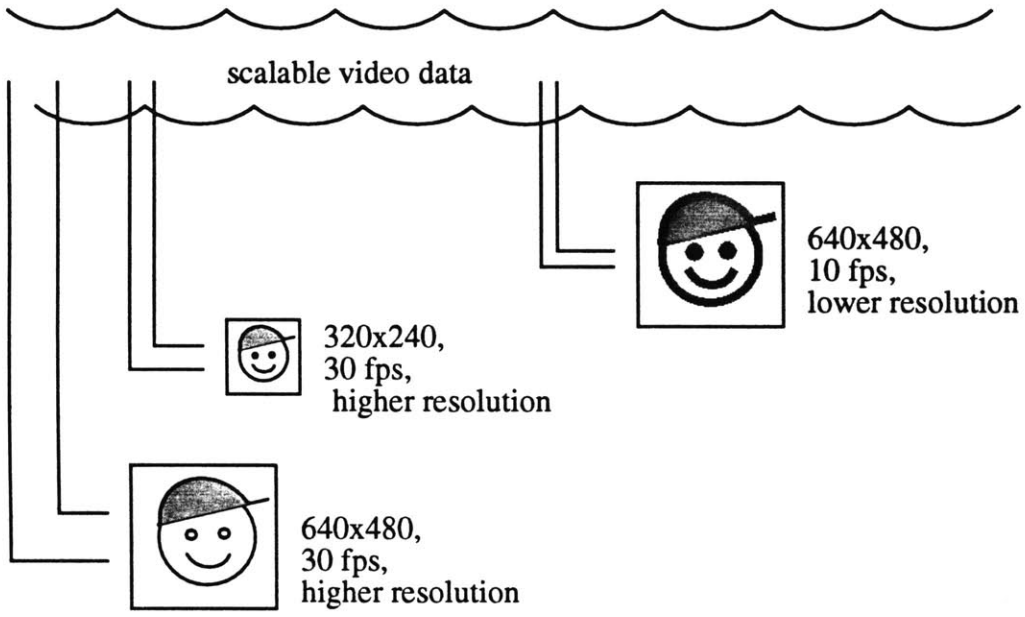

Figure 1. Scalable video as river of data

Advantages associated with scalability have been expressed [8][14] and focus on the utility of creating bitstreams that can be transmitted at high bandwidth for complex decoders to produce high quality images, or at low bandwidth for relatively simple decoders to produce low quality. This gives users the freedom to make their own quality/ cost choices and facilitates interoperability of applications. The number and variety of choices that a particular scalable system provides defines the degree to which the system is scalable.

2.2 One Implementation: Subband Coding

In order to implement a scalable video system it is desirable for the video signal representation to be hierarchical or multiresolutional. One coding technique which meets this

requirement is pyramid coding, developed by Burt and Adelson[9]. In their work, the image is represented by a series of bands of descending resolution each sampled at successively lower rates. Pyramid coding is hierarchical in that these successive bands may be sent progressively with the image improving as each band is received. Another advantage to this system in video display system development is the reconstruction computations are simple, local and may be performed in parallel.

Another similar coding scheme, and the one I have used in this thesis, is subband coding, developed by Woods and O'Neil[23]. In subband coding, the bands are not obtained interatively, but instead as parallel bands obtained through the use of bandpass filters. This scheme is advantageous in that the reconstruction of the bands is obtained through a simple filter-add operation, and the bands are the same size and parallel.

This coding scheme has been investigated in the Entertainment and Information Groups.[8][17] Since Cheops is primarily designed for research of this group, the algorithm which I parallelize in this thesis is the construction of an image (video) from bands obtained through subband coding.

Through the use of high-pass and low-pass filters, the video image is divided into four spatial-frequency subbands corresponding to low-pass in both directions, low hori-

zontal/high vertical, high horizontal/low vertical, and high in both directions. The process is carried out recursively on the low/low band.

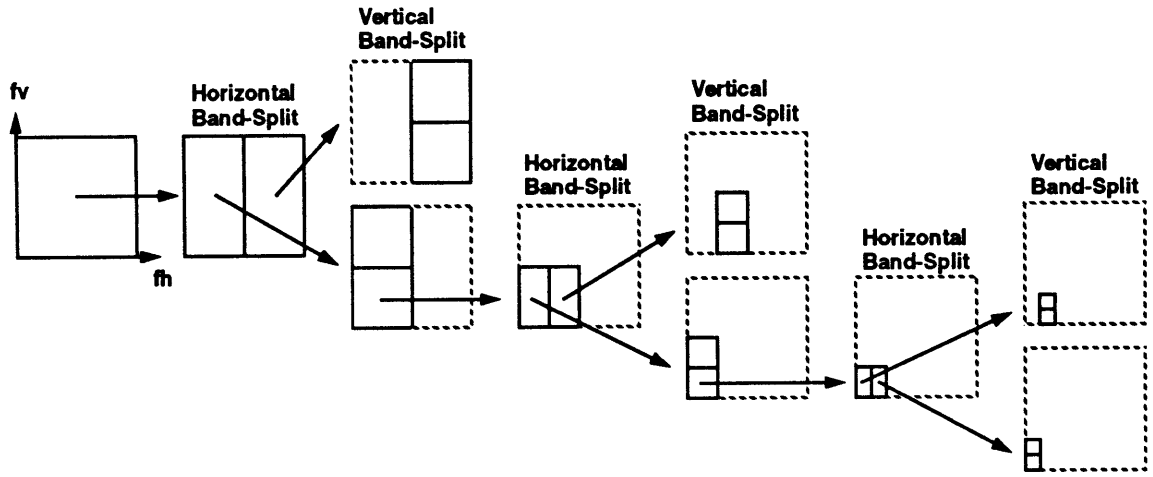

Figure 2. Two-dimensional subband decomposition using one-dimensional filters.

The resulting blocks of data are referred to as subbands. They are arranged in a lattice for hierarchical retrieval from most important to least (to the human eye). Thus, the subband derived through only low-pass filters in both directions is retrieved first. For means of clarity, one can view the subbands arranged hierarchically in the amount of space of the original frame of data. Each subband is labelled as in Figure 3..

I will not go into extensive detail here describing the exact methods of filtering because scalable video does not necessarily need to be represented by the use of subband coding. Other hierarchical means of representation are available (such as Laplacian Pyramids, see [2]) which are scalable in nature. Furthermore, the specific type of subband coding used in this research is not important. For reasons of convenience, I have chosen to use 2-D subband coding to effect scalable video, which I will describe later. Thus, the video

will be scalable in two dimensions (independent spatial resolution), but not in time (independent frames per second).

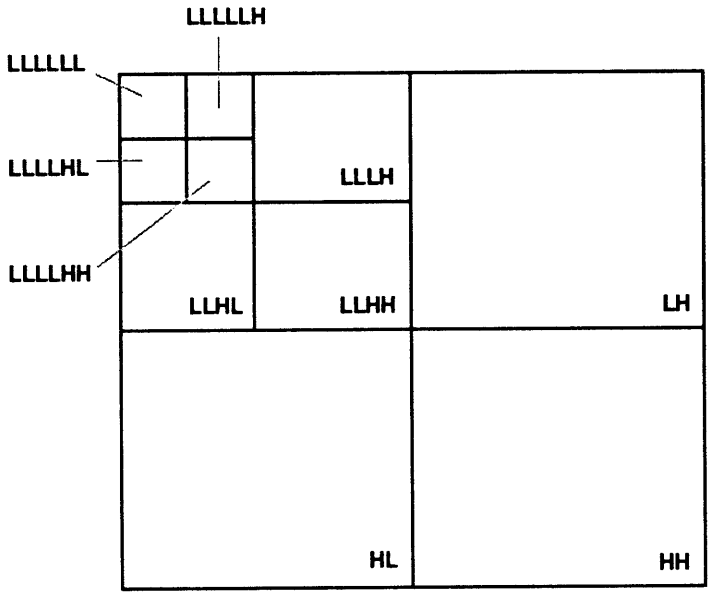

Figure 3. Three levels of 2-D subbands arranged hierarchically

Chapter 3

The Platform: Cheops Image Processing System

Current Cheops hardware and software work are described in this chapter. I have used the Cheops platform as the parallel processing engine to demonstrate the usefulness in adapting scalable video algorithms to a parallel processing environment.

3.1 Cheops Hardware

The essence of the hardware of Cheops is modularity and expandability. It is divided into video digitizing processing modules, processor modules, and video display modules. Thus it can be easily expanded to fill the demands of Open Architecture experiments. Currently two versions of video display modules are available: O1 and O2, each suited to a particular display system. Two versions of processor modules are available: P1 and P2. This work is centered around the use of the more powerful P2 unit.

The second output card mentioned above, O2, has recently been developed to drive large 2000 line screens. Although I have used the O1 output card, the code I write should work on O2 as well.

The modules communicate with one another via two separate data channels: the Nile Bus and the Global Bus. The Nile Bus is used for high speed two-dimensional data block transfers. The Global Bus is used for general purpose transfers. A local bus is also available within each processor module.

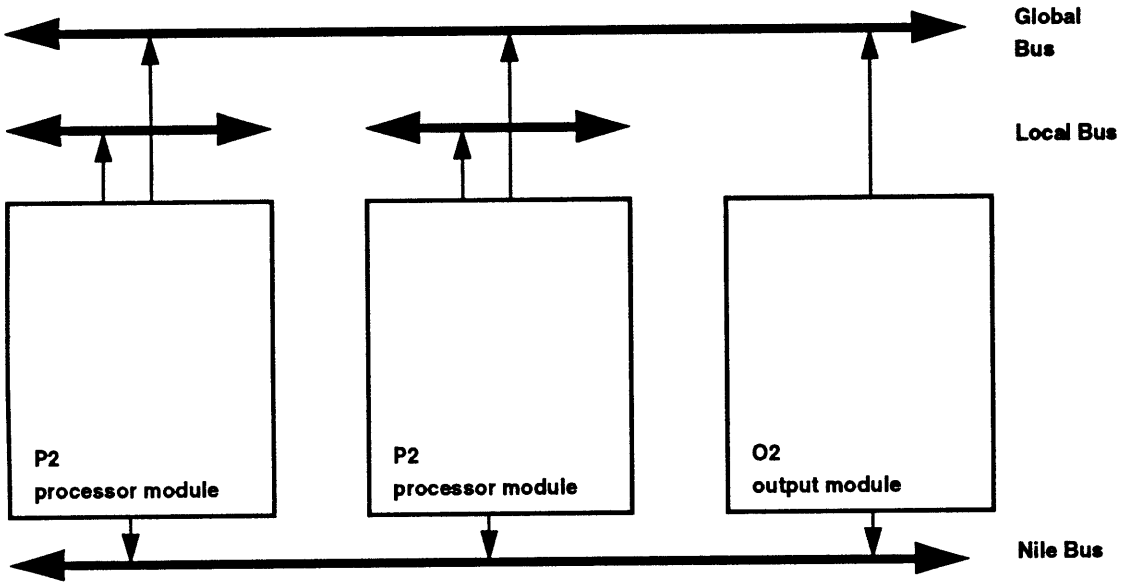

Figure 4. Cheops global, local, and Nile buses on a system with two processor modules and one output module.

Cheops communicates with a number of host computers through a SCSI port. Thus it looks like a disk drive to the host and can be communicated with using normal disk-drive operations like read and write. An I/O port is planned for high-speed transfers with the host for specific host architectures. I will deal primarily with the SCSI port as the channel by which video data will be transferred into Cheops. In addition, Cheops has an RS232 port for testing purposes, thus allowing for a "window" into Cheops for debugging purposes.

Although processor cards can vary in vast ways, I will deal primarily with P2. This board contains 32 megabytes of memory closely tied to several special purpose chips,

such as filter and transpose engines, and a general purpose chip, the Intel 80960CA. The P1 board has one filter chip.

Specifically, I will be using a Cheops system with two P2 boards. My project explores how to utilize the combined power of both boards in a software system for decoding scalable video signals. Ideally, this software will be written in such a way that it can be easily adapted to the system as more boards are added.

3.2 Cheops Software: The Magic Seven Operating System

The Cheops Magic Seven operating system supports process multitasking. The multiprocessing system itself is referred to as Ra (named after the Egyptian Sun God in the tradition of our Ancient Egyptian nomenclature of the Cheops system). Processes communicate with one another through an interprocess communication system based upon double circular queues, referred to as ports. Magic Seven was developed with only one processing module in mind, so no trans-board control or communication is supported.

3.2.1 Multitasking Implementation

Ra is the usual sort of time sharing system, one that gives each running process an occasional slice of the processor's time. Because many of the applications on Cheops are very time-critical (such as video decoding display), Ra gives the user's processes control of the swap times. With this feature comes the responsibility of each process to sleep periodically.

Ra allows up to a fixed number of processes to be resident on Cheops simultaneously. They are written to a number of fixed-length spaces in memory. When a process completes or is otherwise removed, its space is freed and another process may be downloaded into that space.

Ra gives the user control of process swapping at the procedure level with a `proc_sleep()` command. When a process has no urgent task to complete (such as between displaying frames of a movie, or while waiting for a command) it may call `proc_sleep()`.

Before the function of `proc_sleep()` is described the notion of a process's status must be introduced. A process may have one of five possible statuses, described in Table 1..

P_NULL	This slot in the process table is empty.
P_HALTED	Process exists, but is not being executed. Process swapper ignores this process.
P_ASLEEP	This process is not currently being executed, but the process swapper will return control to it when its turn comes around.
P_ACTIVE	Process is currently being executed.
P_NAILED	This process has been killed, but is still functioning because it has not executed a `proc_sleep()`. When it does so, it will not return.

Table 1. Magic Seven process states

The function `proc_sleep()` works in three stages. Its first code segment is an assembly language procedure that saves the process' state. The process' state consists of the contents of the global and local registers, including the stack, frame, and previous frame pointers, plus the arithmetic control, process control, and special function registers. `proc_sleep()` then sets the process' status to **P_ASLEEP**, and calls `ra()`, the process swapper, `ra()` finds another sleeping process (which may be the process just swapped out) and calls `restore()`, which replaces the process' environment and returns it control.

The function `ra()`, which is of the kernel, uses a round-robin queueing algorithm to determine which process goes next. All processes have the same priority level. A more

complicated process swapper could allocate processor time based on varied priority levels. While the design and implementation of a priority based swapper would be simple, it was deemed unnecessary because in applications requiring a large amount of the processing power, the user is well aware of all processes executing. Furthermore, it is even desirable in our case to have complete control of process swapping through the use of `proc_sleep()` because we would not like to hand over complete control to a swapper with no knowledge about our processes. Processes that require frequent processing can be scheduled on interrupts, which already have a priority scheme.

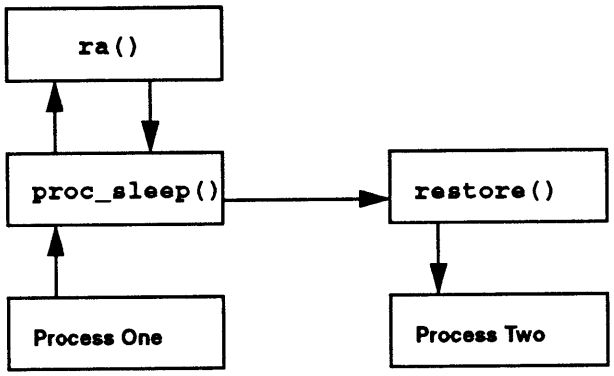

Figure 5. A process swap, initiated by a call to `proc_sleep()`. Up arrows indicate calls, down arrows, returns.

The multiprocessing system uses a pair of data structures to store information about all the processes resident on Cheops. They are the `proc_state` structure and the `usr_proc_table`. The `proc_state` contains the number of resident processes, the process ID of the currently running process, and is also a catch-all for miscellaneous system data and flags. The `usr_proc_table` contains a copy of each process' state.

3.2.2 Interprocess Communication Implementation

Interprocess communication is accomplished within the Magic Seven Operating System by the use of asynchronous message passing. Processes communicate with one another via

two-way communication channels referred to as ports. A variety of system calls are available to send and retrieve messages and conduct remote procedure calls from one process to another.

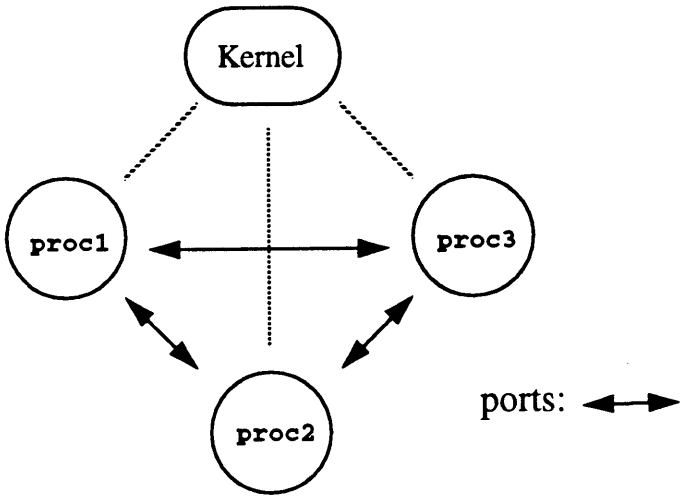

Figure 6. Typical process setup with three processes, each with a communication port with each other process.

The contents of the messages which are sent in ports can be constructed in any way the programmer chooses. However, conventions have been set up which allow the "packing" of variables and strings together into a message for sending and the parsing of the message once it has been received.

Figure 6. shows a typical situation with three processes interconnected by ports. In this setup, each of the three processes is able to communicate with any of the other processes. It may not always be necessary (and often is not) for there to be a port between every process and every other process. For this reason, ports are not automatically setup, but each must be initiated within a process itself.

A unique port is required for every communication channel desired between processes. A port consists of two circular queues, one for each direction of communication. Any process may establish a port connecting any two existing processes.

The host also has the capability to communicate with local processes on Cheops. The M7 IPC system recognizes host processes, which reside on the host computer. Thus, ports can be set up between the host and Cheops.

For example, when one uses the **chex** program to download and execute program *abc*, a port is set up between local process *abc* and host process *chex*. All communication between chex and the local process (i.e. printf, read, etc.) is done through the use of this port.

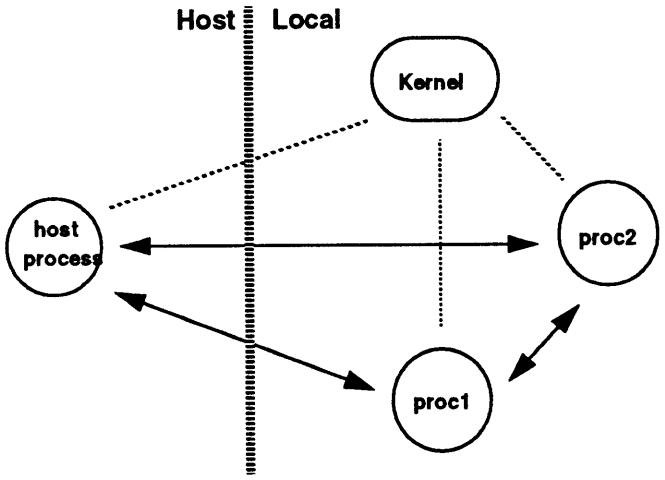

Figure 7. Magic Seven IPC Set-up with a host process, two local processes, and ports between each.

3.3 Resource Management

The Cheops resource manager was written by Irene Shen.[16] It only needs to be briefly described because it is used in my sample application, but details about it are not necessary in a discussion of scalable video in a parallel processing environment.

A stream resource is a special purpose device located on a Cheops processor module. These devices include a matrix transpose operation, various filter chips, a color space

converter, and a motion compensation device. The resource manager schedules and monitors application tasks for each stream resource.

The resource manager is available for use by any process. The process simply creates a doubly linked queue of all of the operations and buffers needed in its filtering operations. A call to the resource manager is made with a timestamp requesting the transfer to be made at the given time. A callback routine is called by the manager when the transfer is complete.

The actual transfer is very fast. We have found that in practice, it is instantaneous compared to all the overhead involved in reading in data, bitstream dequantizing it (i.e. dequantization), and queue construction.

Refer to [16] for a complete description of the resource manager as well as a detailed guide on how it is used.

Chapter 4

A Parallel Operating System for Cheops

In order for Cheops to support scalable video applications operating in a parallel manner, the Magic Seven Operating System must be able to accommodate multiple processes executing and communicating with one another on more than one processor module. I have described in this chapter the requirements for such a system, and the general description of the system I have developed.

4.1 Previous Work

Several established approaches to parallel operating systems have been researched. These have been developed for both loosely-coupled, network based systems and tightly-coupled, shared memory systems. The following is a brief look at some features of operating systems developed in parallel environments:

1 **Distributed Implementations of Conventional Operating Systems.** The LOCUS System, developed and used at the University of California at Los Angeles[15] and the Sun Network File System (NFS)[18] have demonstrated

the adaptation of previous, non-parallel systems to parallel environments. Both of these systems provide transparent access of remote files. All UNIX file access functions were necessarily adapted from existing systems to provide access to remote file servers. The new Cheops Magic Seven Operating System is developed with a similar goal. Many applications have been written previously on Cheops and must transparently be supported by the new operating system.

2 **Client-Server Model for Interprocess Communication and Message Passing Techniques.** The LOCUS and NFS systems use a message passing system which may be traced back to early systems developed at Carnegie Mellon such as HYDRA, the kernel of an operating system for C.mmp[24], and StarOS, a multiprocessor operating system for the support of task forces[13]. Cheops is similar to these systems in that it is utilizes network based client-server communication with remote procedure calling. Cheops utilizes a similar, scaled down version of the client-server model for interprocess communication. However, it is not limited to network access, but allows for shared memory access via the global bus. Thus, the new operating system on Cheops is not confined to remote procedure calls or message passing as the only means of communication.

3 **Multiple Threads of Control.** Another feature of several developed parallel operating systems is the support of multiple threads of control where code and data are shared by multiple threads of execution. Implementation of multiple threads can be seen in systems such as the Mach distributed system kernel developed at Carnegie Mellon University[1] and Topaz, the software system for the Firefly Multiprocessor Workstation developed at Digital Equipment

Corporation Systems Research Center.[19] Cheops hardware could support multiple threads because memory on remote boards able to be accessed by any processor. However, they are not supported in the new Cheops operating system.

4.2 Requirements

The new Cheops Magic Seven Multiprocessing Operating System has been developed as a direct adaptation of the existing single processor operating system. Because Cheops is somewhat a special purpose machine, built for high-speed real-time image decoding and coding, the operating system must have certain capabilities and may be scaled down from larger, more developed, general purpose systems such as those described in the previous section. Many considerations must be made in developing this multiprocessing operating system. I discuss here those requirements which pertain specifically to a system specifically designed to accommodate scalable video. Bearing scalable video in mind, the following requirements are set forth for the multiprocessing system developed to accommodate scalable video:

1 **The system must accommodate multiple processes executing on one processor module.** This is referred to as *multitasking*. Some sort of process swapping must occur between multiple processes each needing use of the same processor. We do not want simply one process running on each processor, but must allow for many processes to execute on each board.

2 **The system must accommodate processes executing on more than one processor module.** This is referred to as *multiprocessing*. These processes must be unique to allow for copies of software executing on different boards. Simple information about each process must be available to any process residing on

any board. For instance, a process may need to know what the current state of a certain server.

3 **An interprocess communication(IPC) system must be provided to accommodate numerous processes on multiple boards.** This IPC system must allow for full message passing capabilities between any process with any other process, regardless of the module in which either process resides. Ports must be allowed to connect processes on the same board, processes on different boards, and processes on any board to host processes.

4 **A semaphore system must be developed.** This is important for real-time video decoding where different frame buffers are being filtered and shuffled among modules primarily to allow transparence so that earlier video systems developed on Cheops may still perform properly. Of course, the semaphore acquires, releases, and polls must be atomic. Semaphores should be accessible across boards.

5 **The operating system should not depend upon how many processor modules are in the Cheops System.** Because the Magic Seven operating system will reside in ROM, it should perform properly whether it is the only module or not. A Cheops software developer does not want to have to load a new operating system every time the number of processor modules used in his application is changed. Furthermore, applications developed on one-module Cheops should run on a multi-module Cheops.

6 **The operating system executable code must be identical on each board.** This is for reasons of convenience. A processor module should be allowed to be the only module on one system, while have the capability to be transferred

to another Cheops System as board number two. One ROM (the same ROM) should be burned and copied for use on every processor module.

7 **The system must provide a means for loading and executing code from one board to another.** This is so that not every module must have a direct link with the host computer, which serves as a file server via SCSI. The system must provide the means necessary for this to be accomplished, but does not necessarily have to accomplish this directly.

8 **As much as possible, the system must be an extension of the current one-module system.** System calls in this system must be supported with the same interface as system calls from the original operating system. Ports and semaphores must have the same appearance to the user as in the old system. This allows upward compatibility for all existing applications.

9 **A means for debugging and updating the actual operating system itself must be provided.** This includes the capability of loading and executing a modified M7 in RAM.

10 **Interprocess communication should prefer the local bus over the global bus for reasons of speed as well as availability.** The global bus must be shared, while each module has its own local bus.

4.3 General Description of Implementation

With the above requirements at hand, the new Magic Seven Operating System has been completed. The the three major aspects affected in conversion to multiprocessing are multitasking, interprocess communication, and semaphores. The conversion process to accommodate parallel processing was similar in all three of these areas.

The basic approach takes advantage of the characteristic that the code on each board is exactly the same. (requirement 6) Because the Magic Seven code in each module's ROM is the same, the space reserved for global variables is in the same respective locations on each board.

Thus, to access a a global variable from another boards' memory, a global bus mask is applied. For example, memory within module one is accessible by the local bus with address 0x3abcdefg..., while also being accessible via the global bus as address 0x8abcdefg.

In Figure 8., we see how a process on board one can access a global variable on each board. Because it knows the location of its own global variables, a simple mask is applied to access the parallel variable on other boards. Thus, it accesses its own addressing space via the local bus, fulfilling requirement step 10, while accessing other boards via the global bus.

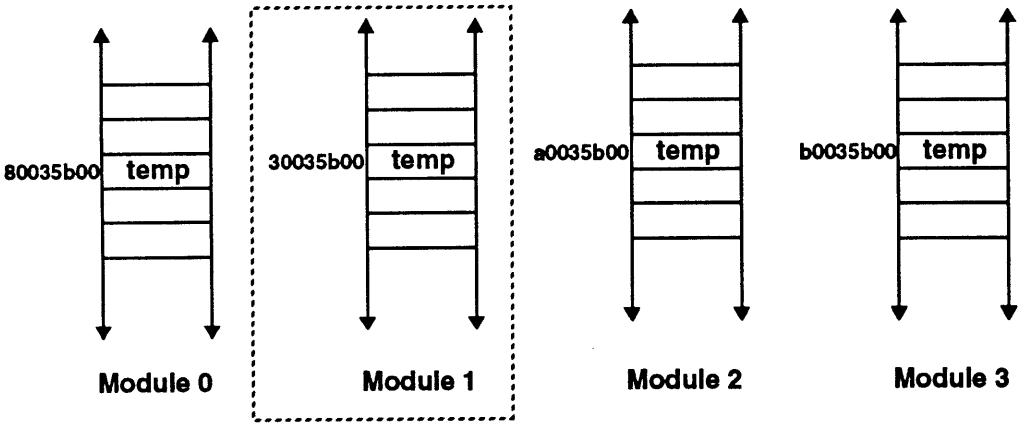

Figure 8. Parallel global variables as accessed by Module One

One important consideration is that of more than one module trying to write to the same location at the same time. Since each of the processor modules must be capable of initiating a bus transaction, a method is provided in hardware for bus masters to request

and arbitrate for possession of the global bus. The priority scheme gives lower board numbers priority. Thus, board zero is the primary board, always capable of controlling the global bus.

The preliminary design of the multiprocessing Magic Seven system included a monitor server, which accepts system call requests from the other modules. Thus if a process on board one wanted to know if a certain process was executing on board zero, it would send a `proc_get` request to board zero. The monitor server on board zero would then service the request and return a message revealing whether or not the process in question was executing on its board.

This approach presented several problems. One is that a process must always be polling for messages on each board. This is not efficient, especially when the application processes are in need of the maximum processing power available. A second shortcoming is precious port space must be reserved for these module-to-module messages. Only three bits are reserved in the message packing mechanism for the port number. Thus with only eight ports, we cannot afford to have a unique port connecting each module's kernel. One partial solution to this was to limit message passing to communication between each board and the mother board (module zero). This reduces the total number of ports, but ports from each kernel to the mother kernel (probably residing on module zero) module zero would still have to be allocated, thus taking away port resources from applications.

However, further evaluation revealed that every system called on a remote board could be handled by accessing that boards memory directly via the global bus as long as there is knowledge of the location of all the magic seven global variables on the other boards. This is done by taking advantage of the earlier discussed fact that each board's operating system resides in exactly parallel memory locations.

Instead of requesting to another module's operating system that a system call be made, we can simply bypass that operating system and execute the desired function upon that boards global operating system variables directly. This is the approach taken in approaching the modification of Magic Seven.

I describe this in more detail by discussing the implementations within three major areas of operation: multitasking, interprocess control, and semaphore implementation.

4.3.1 Multitasking

This section deals with modifications to the multitasking system including all process information system calls.

The method by which multitasking is accomplished on the single processor module was explained earlier in section 3.2.1. Briefly speaking, system calls within the multitasking system perform operations upon a table, which contains information about each process. The simplified table is shown in Figure 9..

`proc_table`

process state	return inst ptr	process label
P_ACTIVE	600004b0	M7 monitor ROM
P_SLEEP	301004b0	hello
P_HALTED	301004b0	font_server
P_NULL	0	NULL

Figure 9. Simplified `proc_table` in memory, showing process' state, return instruction pointer, and process name.

In the new Magic Seven system, a table of pointers, `trans_proc` (see Figure 10.), is created during boot time, which points to the `proc_table` on each of the other boards. This is done easily by masking the local `proc_table` address with the respective module global bus bits.

An example system call is `proc_get`. This call returns the process identification number (PID), given the name of the process. It might be used in the case where a process wants to know if a certain server is executing on the current module. Previously, `proc_get` was simply a search through the process names in `proc_table`. In the new multiprocessing system, `proc_get` first searches its own table. Then, using the `trans_proc` pointers, it searches the tables from other modules directly. Thus, no processing power is used from the other processors.

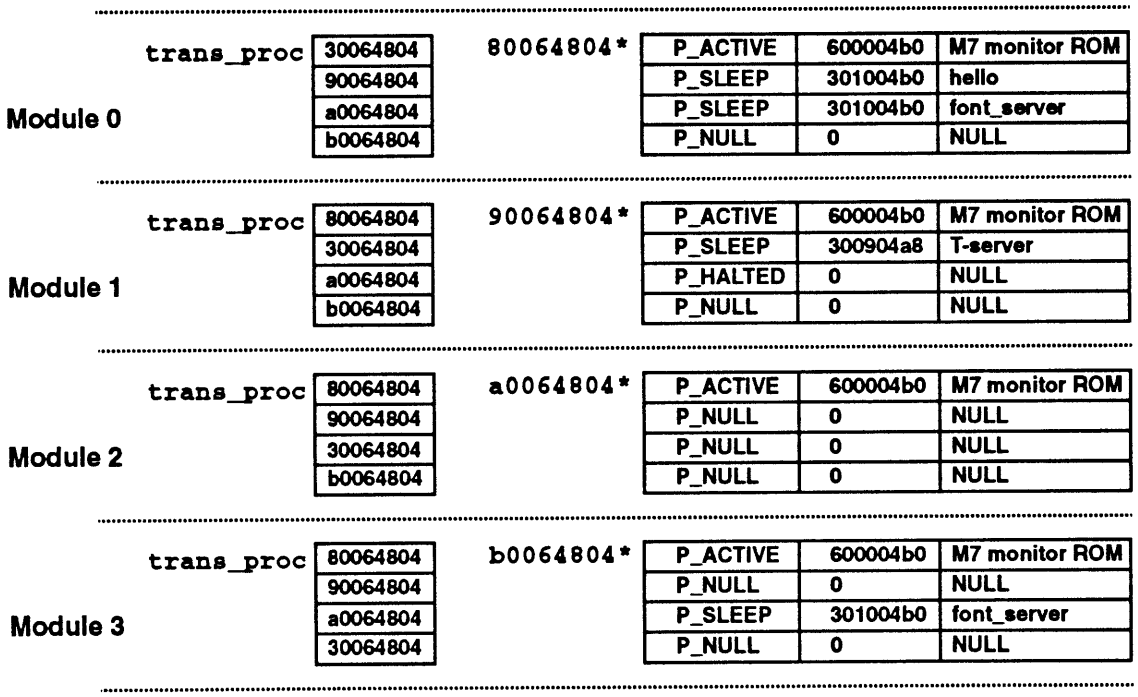

Figure 10. Each module's `trans_proc` table with simplified `proc_table`. Addresses marked by * are global bus addresses.

Care is taken to make sure a module's processor does not write to the table at a dangerous time. For instance, when setting up a process in `proc_start`, the state (P_ACTIVE) must be the last thing that gets set. Otherwise, a system call from another module may try to access that process prematurely. A rule I used to prevent similar problems as

this when modifying `proc_*` system calls was to only *read* from remote modules. Only the local operating system *writes* to its own memory, thus eliminating the problem dealing with other remote processes destroying the flow of a local process. All necessary `proc_*` system calls could be implemented in this way.

Another modification to the old operating system stems from the need of unique PID's. In order to maintain uniqueness, each module will have its own reserved PID's. Board zero is assigned ID's 0-15; board one, 16-31; board two, 32-47, and so on. System calls were all changed to support this new numbering system. Parameters to these system calls, as well as return values (where applicable) all use this new PID numbering system. With reserved PID numbers, one can immediately know on which board a process is running given its PID, and *vice versa.*

System calls are thus handled by directly accessing other modules' global variables with process information. The actual multitasking process swapping implementation was untouched during the modification to the new multiprocessing Magic Seven system.

4.3.2 Interprocess Communication

Interprocess communication on the single processor operating system was described in section 3.2.2.In our modification from a one module M7 to a multiprocessing M7, the modification of the interprocess communication (IPC) portion is similar to that of the multitasking part, described in the previous section. The knowledge that every module has a replica operating system executing is used to directly access other module memory.

The essence of the IPC system within M7 is two-way ports. Each port consists of two circular queues, and connects a source and a destination process. This structure is represented below.

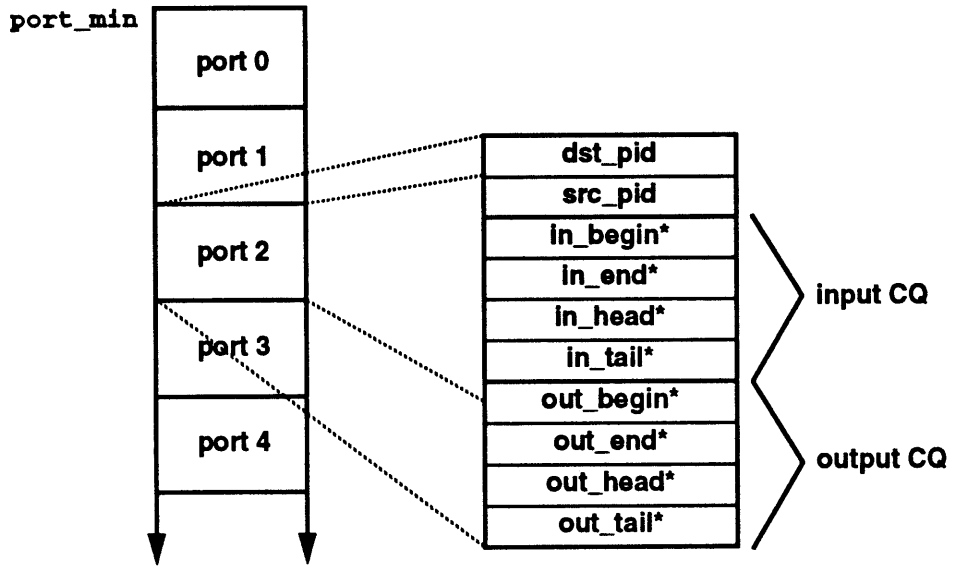

Figure 11. Port structure representation in memory on a single module.

To access another module's port structure, the same global bus mask method is used. The address `port_min` is masked to create a table of pointers, `trans_port`, to the port structure of each separate module. Thus, the `trans_port` table can be used to directly write to or read from any port on any board. For the diagram of this table, see Figure 12..

A convention is needed to determine the location of a port which connects processes on different modules. The convention selected was the port will reside in the module containing the process which creates the port.

While the multitasking part of Magic Seven can make the assumption that only the local module's operating system needs write to its own memory, the IPC system does not have the convenience. Each process on each board must have the ability to read from or *write to* any other process, regardless of the module which the port resides. Circular func-

tions only advance head or tail pointers after the given operation was performed. Thus, no data will be prematurely written or read. Furthermore, only one process writes to any given circular queue

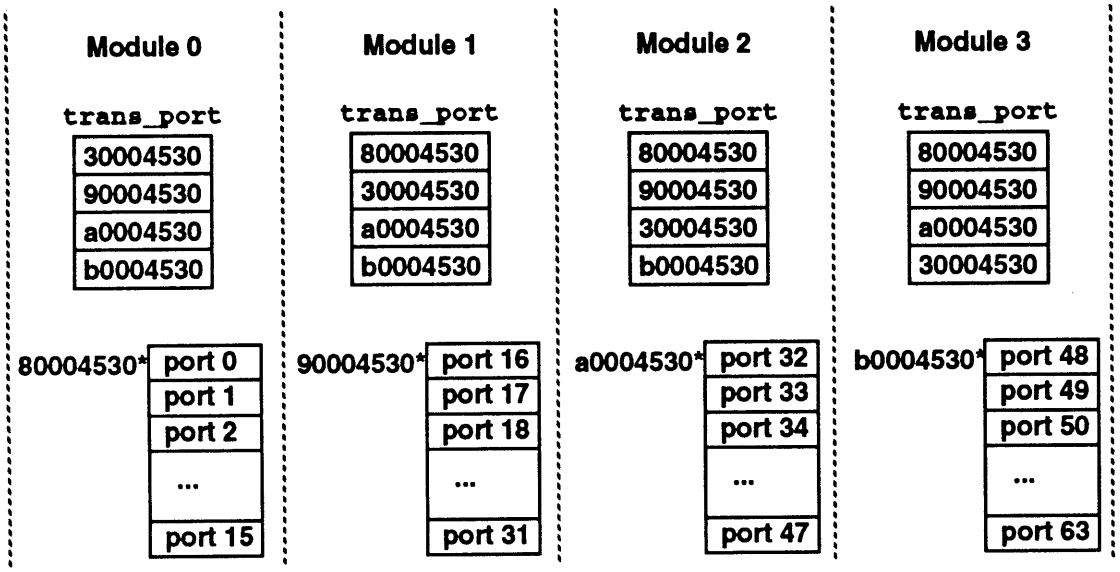

Figure 12. Multiprocessing port structure

One more problem that was addressed is that of the actual head, tail, begin and end pointers for each queue. They were changed to be global bus pointers, so remote processes could use them to access the port. If local bus pointers were used, the remote process would alter a port within his own memory rather than the desired port. Circular queue functions check if the given global bus pointer points to the local board. If so, it is converted to a local bus pointer to allow for faster queue access.

4.3.3 Semaphores

Semaphore implementation was quite trivial after working through the multitasking and IPC implementations. The same method is used of directly accessing memory of remote modules through the use of global bus address masking.

The structure `trans_sema` is set up in the same way as `trans_proc` and `trans_port`. All semaphores have a unique ID. The resulting structure is shown below.

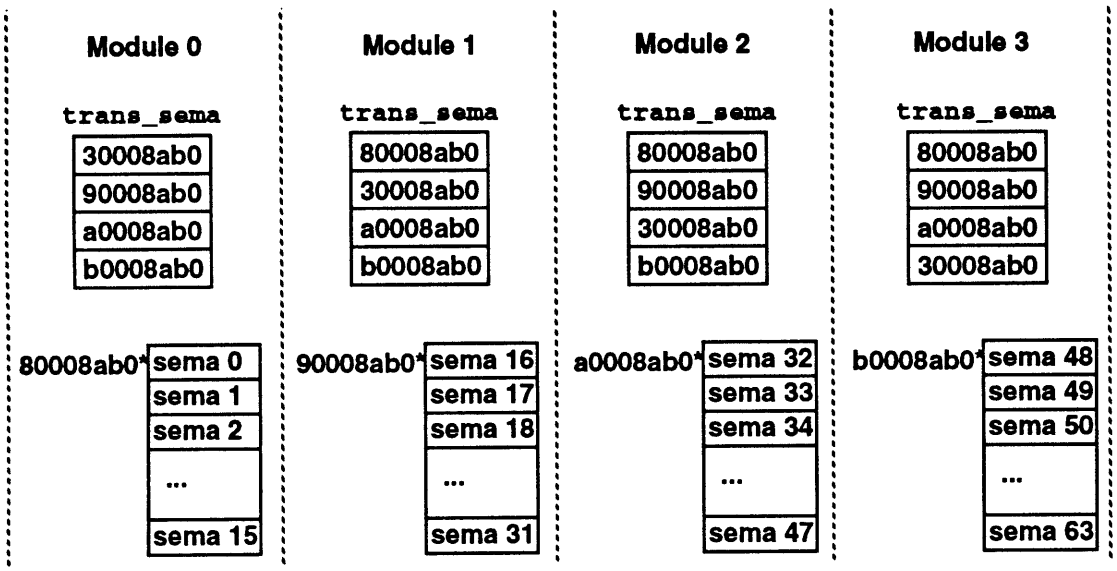

Figure 13. Multiprocessing sema structure

The only concern with semaphores being acquired, released, and polled across modules is whether these operations are still atomic. Investigation showed that global semaphore atomic read/writes are supported within the hardware for the global bus as well as the local bus.[21] This is accomplished through hardware bus locking.

By convention, the function `sema_allocate` allocates a semaphore of the module on which the calling function resides.

4.4 Overview of Requirements

Once more, I will describe each of the requirements, followed by a description of how the new system meets each requirement.

1 **The proposed system must accommodate multiple processes executing on one processor module.** The *multitasking* aspect of M7 was retained.The same sort of process swapping occurs between multiple processes on a given board as before.

2 **The proposed system must accommodate processes executing on more than one processor module.** The *multiprocessing* requirement of M7 is met. Processes may execute in parallel on each board. Necessary information (all that in `proc_table`) is available to any process on any board through the use of `trans_proc`.

3 **An interprocess communication(IPC) system must be provided to accommodate numerous processes on multiple boards.** Because ports are set up with the capability of connecting to processes on other boards this new IPC system allows for full message passing capabilities between any process with any other process, regardless of the module in which either process resides.

4 **A semaphore system must be developed to allow synchronization among processes.** Because all semaphores are accessible via the global bus from any board through the use of `trans_sema`, synchronization is possible among all processes on any board.

5 **The operating system should not depend upon how many processor modules are in the Cheops System.** The beauty of this new system is that the code on each board is exactly identical. Thus, the code for a single board will work on a board among many boards.

6 **The operating system executable code must be identical on each board.** Again, this system uses identical code on each board. We only need to burn one type of ROM to be used on any board.

7 **The system must provide a means for loading and executing code from one board to another.** This can be done by downloading the code (via global bus) and then setting up the process using system calls on the remote processes. The remote monitor will then recognize the new process and give it processor time.

8 **As much as possible, the system must be an extension of the current one-module system.** The new system has the same "look" as the old. All system calls behave the same, therefore old applications execute properly on this new system.

9 **A means for debugging and updating the actual operating system itself must be provided.** In the same way as before (as described in current Cheops Documentation), new versions may be loaded into RAM. Notice that it is very important that the same version be loaded on every board in the system. Otherwise, specific memory variables may not be in parallel locations.

10 **Interprocess communication should prefer the local bus over the global bus for reasons of speed as well as availability.** System calls always check at the beginning of the system function itself if the desired call may be completed on the current board. If so, the local bus is utilized.

Chapter 5

The Decomposition of Scalable Video Algorithms

The most important decision to be made in developing a scalable video system utilizing parallel processing (or for any parallel process algorithm) is how to divide the task at hand into different processes. This chapter contains a discussion of the issues involving dividing a Scalable Video system for implementation in a parallel processing environment. A set of characteristics are discussed as requirements for such a system.

Three approaches are considered as viable alternatives in the task division of a Scalable Video display system: Pipeline Division, Temporal Division, and Spatial Division. For each approach, a description is given, as well as an analysis of how the process fits the above characteristics.

5.1 Scalable Video Guidelines

In our situation, we would like a division scheme with the following characteristics:

1 **Cheops Improvement.** Primarily, parallel processing should allow Cheops to do something it cannot already do. This thesis should yield means by which work can be accomplished which was not previously possible.

2 **Module Process Similarity.** Each module should have similar (preferably the same) processes running on it. This is due to the fact that our processor modules are, in fact, the same. An elegant solution to task division would have parallel "looking" processes running on parallel hardware. This is not completely necessary, but desired to allow for easy adaptation of existing software to a parallel environment.

3 **Algorithm Adaptability.** The division scheme should adapt to other scalable video implementations other than subband decoding. (such as motion compensation)

4 **Hardware Extensibility.** The given algorithm should be easily extended to more or less modules than as designed.

5 **Data Flow.** Data passing between modules should be kept at a minimum. The bus for intra-module transfers is much slower than that of local transfers. High speed busses may be used, but a minimum of data transfer is desirable.

6 **General Simplicity.** A highly complex implementation is disadvantageous because it not only does not easily transfer to other applications readily, but because in a research environment, time should be spent on developing new methodologies and algorithms, not debugging complex software.

5.2 Pipeline Division Approach

In the Pipeline approach, each module performs a unique task in line. A diagram of how this may look is shown below.

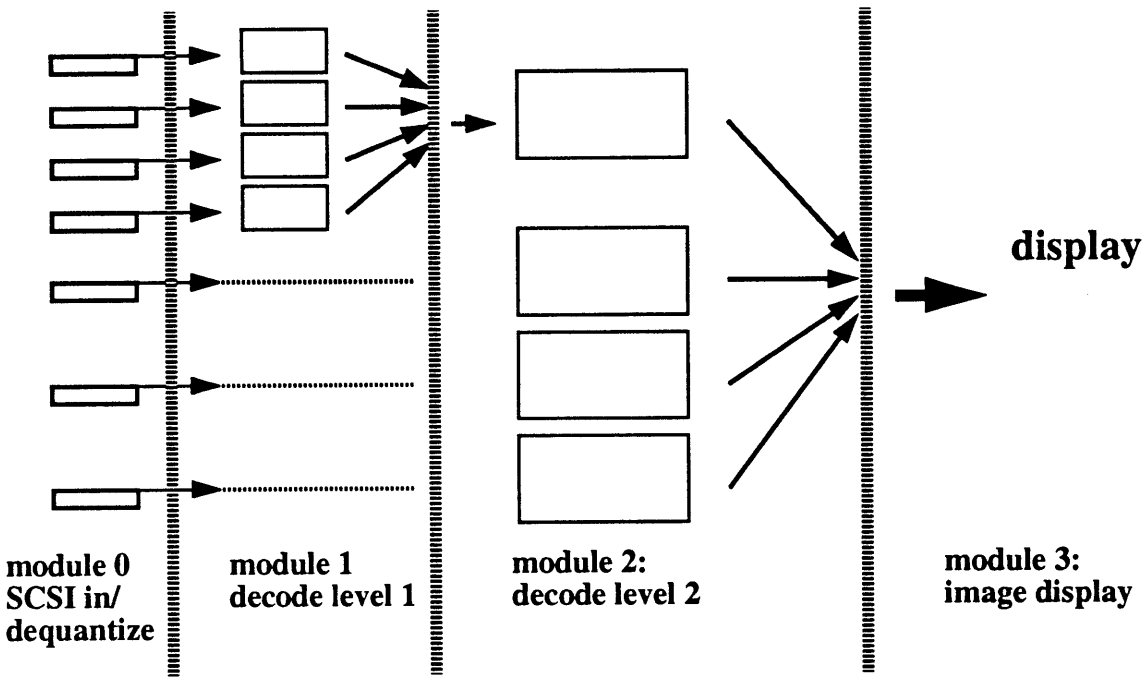

Figure 14. Pipeline division of two level subband decoding

The pipeline approach meets the *Cheops Improvement* requirement, but only partially. Cheops is only improved in efficiency. Bottleneck problem areas common on one board, such as SCSI data in and image display rate, still exist because each area is still handled by only one module.

The *Module Process Similarity* requirement is obviously not met due to the complete differences in processes running on each board. This leads to extremely limited *Hardware Extensibility*. If the number of hardware modules is increased or decreased, major changes must be made to the implementation to adapt to the hardware change. This is not necessarily a disadvantage because it allows for module specialization, which can lead to very effi-

cient processing. The *Algorithm Adaptability* requirement is not met because the task division is so closely tied to the algorithm itself. Another disadvantage is that large amounts of data flows between boards, thus the pipeline approach does not meet the *Data Flow* requirement. Finally, General Simplicity is not met because the software to implement this division is quite complex. Comparatively Much time must be reserved in testing new algorithms when implementing this division approach.

Task Division requirements met:	Cheops Improvement
unmet:	Module Process Similarity
	Algorithm Adaptability
	Hardware Extensibility
	Data Flow
	General Simplicity

Table 2. Overview of Pipeline division analysis

5.3 Temporal Division Approach

In the Temporal Division approach, the video is divided temporally, such that each module individually and completely processes a given number of frames.

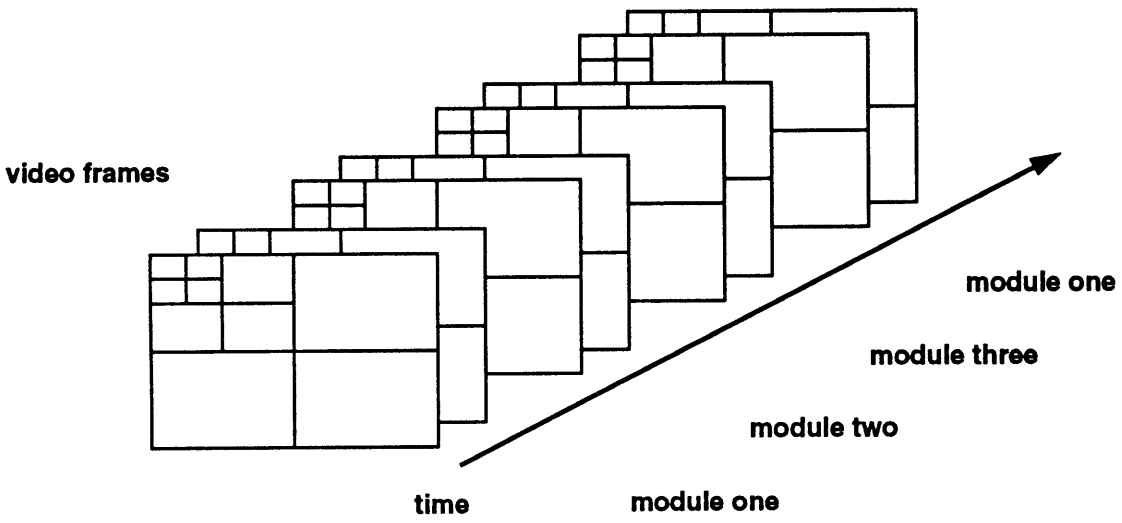

Figure 15. Temporal division of three level subband decoding with each module decoding two frames

The temporal approach better meets the requirements set forth in this chapter. Each module is performing identical task, thus *Module Process Similarity* is fully met. *Hardware Extensibility* is also met because the division scheme can easily be adapted to any number of hardware modules. Only synchronization and control signals are sent from module to module via the global bus, so we have good *Data Flow.* The *General Simplicity* requirement is met, with new algorithms relatively simple to design using this method.

Cheops Improvement is only partial met because only efficiency is increased in this parallel system over a one module system. The bottleneck associated with SCSI may be somewhat overcome in this method, but any display bottlenecks, which may limit the final

picture size, still exist. Finally, *Algorithm Adaptability* is only partially met. This division breaks down significantly when the scalable video algorithm calls for processing one frame given information about the previous frame. Such is the case with motion compensation decoding systems.

Task Division requirements met:	Module Process Similarity
	Hardware Extensibility
	Data Flow
	General Simplicity
somewhat met:	Cheops Improvement
	Algorithm Adaptability
unmet:	none

Table 3. Overview of Temporal division analysis

5.4 Spatial Division Approach

The final approach considered is to divide the image spatially. This method is shown below.

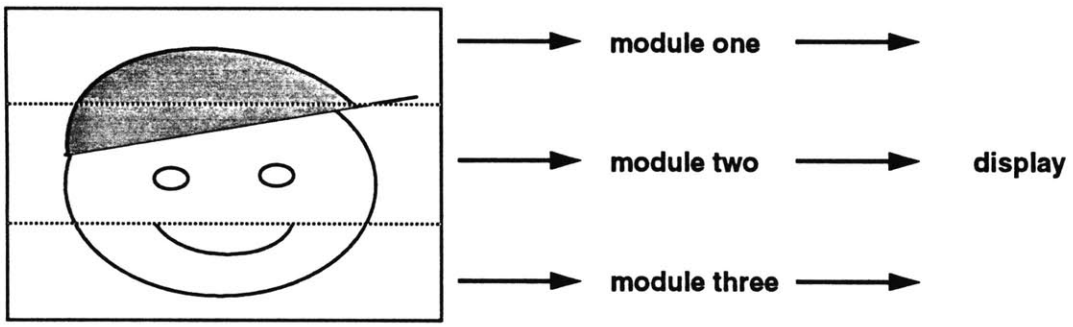

Figure 16. Spatial division of subband decoding with each module decoding an image segment

This method of division has many advantages over the previously discussed methods. Each module is performing identical tasks, thus we meet the *Module Processes Similarity* requirement. *Cheops Improvement* is met because all bottlenecks are reduced. Each

module does essentially the same as if handling the entire operation alone, but every level of processing work is reduced. The division scheme is *Hardware Extensible* because any number of boards can be readily combined to contribute to the video decoding system. This method also meets the *Application Adaptability* requirement because practically all video display algorithms work at some level within the spatial domain. Finally, the *General Simplicity* requirement is met, with new algorithms relatively simple to design using this method.

Some increase in Data Flow is required. This is due to problems at the edges. Either some amount of edge data must be transferred among modules, or each module must read in an overlap into other modules' regions. My particular application uses the latter approach. This overlap edge problem is related to the size of the filter, range of motion compensation vectors, or other factors which utilize spatial correlation information in the coding/decoding.

Note that the *Hardware Extensibility* requirement, while met, is not trivial. The coding process may need to know how many modules will be used to decode the video. Alternatively, a relatively intelligent means must be used to parse the bitstream to access data from each spatial image region.

Task Division requirements met:	Module Process Similarity
	Cheops Improvement
	Algorithm Adaptability
	Hardware Extensibility
	General Simplicity
somewhat met:	Data Flow
unmet:	none

Table 4. Overview of Temporal division analysis

5.5 Overview of Division Approaches

Here is an overview of the requirements as met by each of the three presented task division approaches.

Requirement	Pipeline	Temporal	Spatial
Cheops Improvement	met	somewhat met	met
Module Process Similarity	unmet	met	met
Algorithm Adaptability	unmet	somewhat met	met
Hardware Extensibility	unmet	met	met
Data Flow	unmet	met	somewhat met
General Simplicity	unmet	met	met

Table 5. Overview of scalable video task division requirements and approaches

An investigation into the advantages and disadvantages of the methods of division presented in this chapter leads to my decision to use the spatial division method. The only disadvantage to this is the Data Flow problem, or simply stated the "edges". This may be solved at little loss by reading in the overlap the size of the filter. This is only 9 samples in my subband decoding example, so spatial division is the obvious choice.

Chapter 6

Sample Application: Subband Decoding on Cheops

To further illustrate the spatial division process, I will discuss a particular scalable video implementation on Cheops. This specific implementation is that of subband decoding. The issues dealt with in this chapter should be useful to those interested in a) expanding this application, b) developing other parallel scalable video systems, and c) writing any parallel process on Cheops.

6.1 Description of Algorithm

A detailed description of common subband decoding as commonly used by the Entertainment and Information Group is discussed in [8][17], and thus is not repeated here.

6.1.1 Subband Representation: *code2dll*

A UNIX script has been created (called *code2dll,* for 2-D, one-level subband decoding), which converts a datfile (the standard image representation in our lab) into the BAND format shown below.

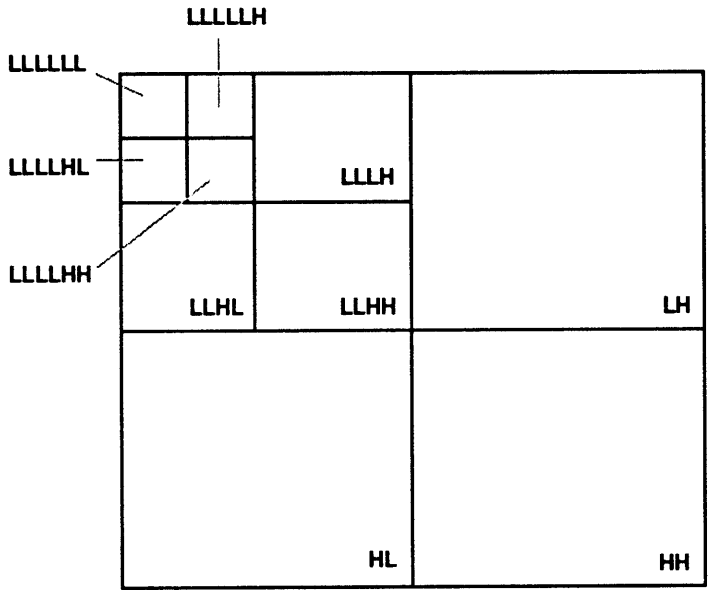

Figure 17. Three levels of 2-D subbands arranged into BAND format

The BAND format show here only represents one frame of a movie. This representation is the file available from which the movie player may retrieve data. It is chosen as the source for parallel scalable decoding because of the ease at which it divides spatially, as well as in frequency. I use only one-level subband decoding in this implementation to reduce complexity so that focus may be upon the parallel nature of this system, not the subband coding itself.

6.1.2 Subband Retrieval: *decode2dll*

A program, *decode2dll* has been written which is a Cheops executable file to do the actual scalable video simulation. The key to this decoder is making sure the code on each module

receives the correct data to decode. I have provided a function called **get_subband_dims()** which does this. It takes as parameters the band_string (i.e. "LH","LLHL",...) and an empty BAND structure. The function identifies the datfile submatrix where the proper data is located and stores the information in the BAND structure. The following figure shows how single level subband encoded data is divided into different modules.

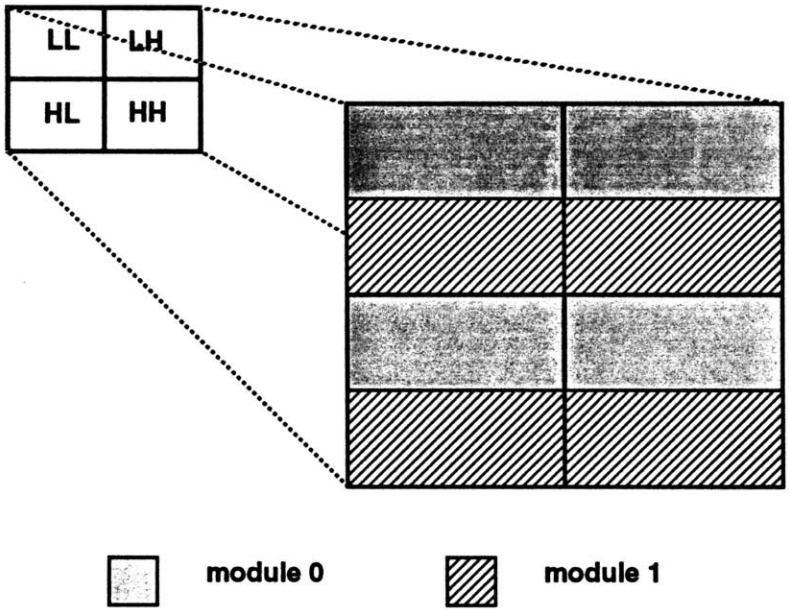

Figure 18. Division of single level subband data for processing with two modules

This function actually identifies the submatrix slightly larger than shown. For example, with two processor modules (as the above figure), the areas designated for each processor actually overlap in the center of each band. This is so that the filters do not cause the noticeable lines in the reconstruction of the final image. In this implementation, with filters of 9 taps, required an overlap of 12 pixels (theoretically 9, but in my implementation, all transpose operations had to be multiples of 12).

The flow of decoding the four subbands in single level subband decoding is shown in below.

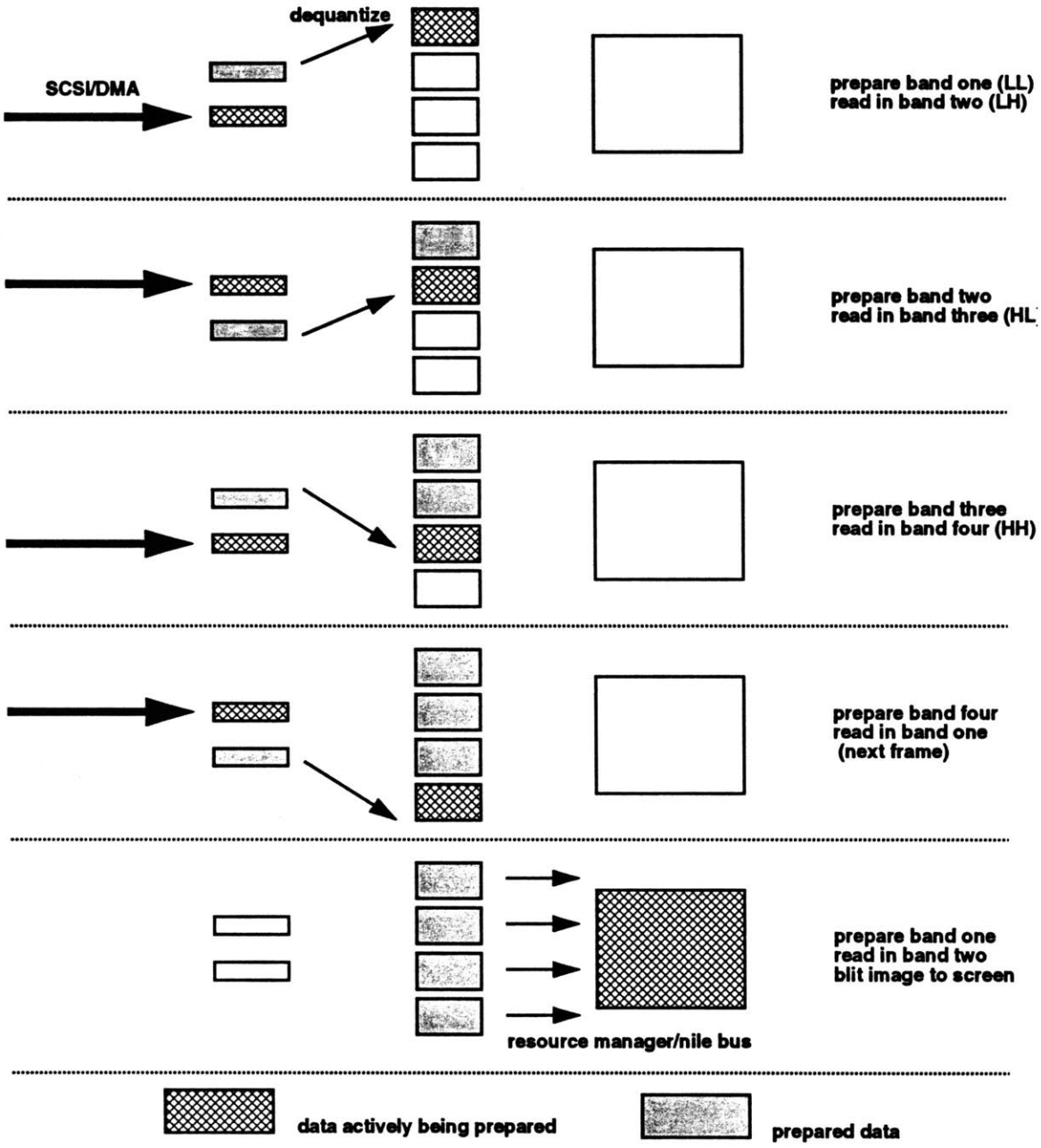

Figure 19. Flow of one processor of subband retrieval system (the *decode2dl11* movie player)

A few things must be noted about this diagram. Double buffering transfers into the two SCSI buffers is handled by the SCSI DMA. Thus, a some degree of pipelining occurs

within each individual module. The dequantizing is done in my implementation by the Intel 80960CA processor. Future plans are to have special bitstream decoding chips to handle dequantizing or similar decoding techniques.

Data is read from the network via SCSI with the function, `d_future_read()`. This function activates the SCSI DMA so that transfer will begin, but control is immediately returned to the subband retrieval application. When all four bands are loaded, a `request_transfer()` is called, which invokes the resource manager with the request to do the proper filtering for subband decoding.

Working together, a two module system looks like the following figure:

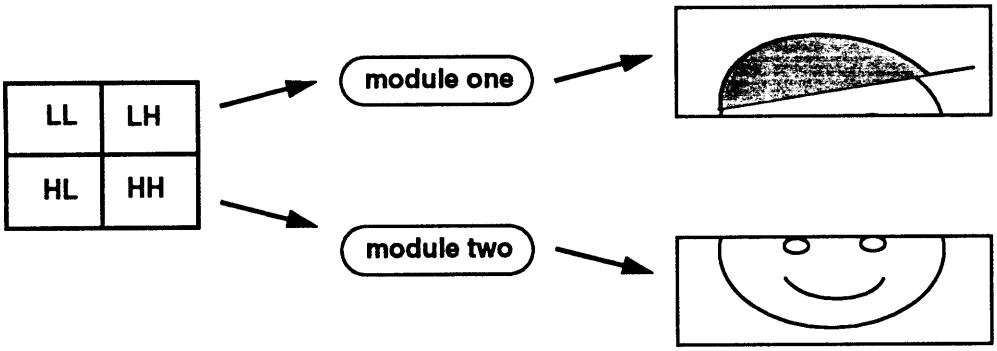

Figure 20. Combination of two processor modules to reconstruct a one level subband image

Proper output must be considered. All that is needed is to properly position the final result of each module on the screen. This is trivial since the size of the segment is known to the application.

Synchronization is handled easily through the use of Nile Bus transfer timestamps. Every transfer is assigned a timestamp. Initialization of the program includes an agreement of the time of the first frame to be displayed. According to the desired frames/second of the movie, transfers are requested at the respective times. Unfortunately, synchroniza-

tion is not available until the Nile transfers are working properly. As of the writing of this these, it was not quite ready, so the various segments of the image were not synchronized.

6.1.3 Performance Evaluation

In analyzing the performance of this implementation of single level subband decoding, I will refer to the following figure:

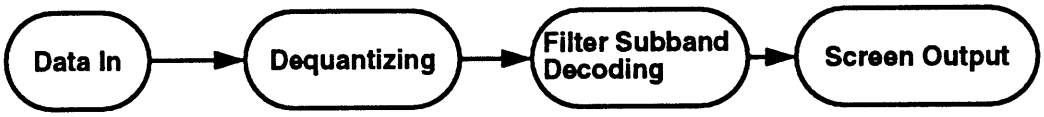

Figure 21. Flow of data in subband decoding

The Data In stage is the bottleneck in my implementation. However, SCSI is by no means the only (or fastest) means by which video data will enter a video system. Cheops is even projected to have a high speed bus connection to replace SCSI in the future.

The dequantizing stage is currently handled by the Intel 80960CA. This stage will be replaced in future versions of Cheops with hardware bitstream decoding chips. Thus, this is not a stage which presents a bottleneck problem.

The key stage is the subband decoding using filters. Investigations in implementations of scalable video will hinge upon this reconstruction phase of the display system. This is the stage which I have successfully cut the execution time substantially (see Table 6.) when utilizing two processor modules.

The Nile Bus library, which handles high speed transfers to the screen from processor modules, did not work properly when I wrote this implementation, so a global transfer was written to simulate the transfer. This is a much slower method of transfer. In figuring the timing numbers in Table 6., I did not display to the screen because the global bus caused a bottleneck which will disappear when the Nile Bus works properly.

The table refers to two tests executed, one with the decoding stage and one without. From these two tests, I approximate the time Cheops takes to decode the bands. Four modules were not available, so the times are computed with one module doing the work it would do if there were four modules available to run the tests.

number of modules: test:	one	two	three	four
a. SCSI-deq.-decode	4.73	3.89	3.63	3.49
b. SCSI-deq.	4.41	3.67	3.47	3.40
c. subband decoding (computed a-b)	0.32	0.22	0.16	0.09
d. percent of time used by one module (c)	100%	69%	50%	28%

Times are seconds per frame of video (one 800x640 monochromatic image)

Table 6. Timing results of single level subband decoding as scalable video implementation on Cheops

The key line of this table is line c. As the bottlenecks which will disappear in the future are removed from the calculations, we see a significant improvement of efficiency as more modules are utilized. I had expected a direct improvement, with 50% for two boards, 33% for three, and 25% for four. The expected result occurred with four boards, but not with two and three. The cause was not directly apparent to me at the time of writing, but the problem may be related to quantum effects. Certain pieces of the implementation may be more efficient in dealing with certain multiples of samples occurring in the four module scenario. More investigation is required to determine the cause of this result.

Chapter 7

Conclusions and Recommendations

The overall goal of this thesis was to further research in the area of Open Architecture Television, specifically the feature of scalable video. A parallel processing software operating system was developed for Cheops with the goal of accommodating the needs of a scalable video implementation. The operating system was created to easily develop parallel image decoding algorithms. This platform, the newly updated Magic Seven Operating System, was developed with the scalable video researcher in mind. Several approaches to dividing the task of scalable video for implementation in a parallel environment were analyzed. It was found that a spatial division of the image was desirable for our system. A scalable video system was then implemented within this new multiprocessing Cheops environment using the spatial division approach.

Overall, the new Cheops Magic Seven Operating System, while a simple, direct extension of the earlier, single processor version, provides a unique and sufficient platform for the experimentation of scalable video experimentation within a parallel environment.

The Cheops system provides room for up to four processor modules. The multiprocessing operating system was developed with this in mind. Many features of the system, as well as of the scalable video implementation are geared towards a small number of processors (on the order of 16 or less). I do not recommend adaptations of this operating system or scalable video decomposition for platforms with more than 16 processors.

I make several recommendations for further research related to the work described in this thesis:

1 Only many experiments within this system will fully confirm the advantages of the spatial division approach.

2 Considerable work could be done in the area of automation of the task dividing. Given an algorithm for displaying a movie, what would go into an automatic division according to the number of modules in the system? My example application requires the user to manually execute the same process on each module with different parameters telling the module which module number it is and how many modules are present.

3 I found the Magic Seven Operating System to not have a very apt method of synchronization. Other than timestamping with the use of the resource manager, is there any good way to synchronize processes? This may be helpful when audio is brought into play.

Finally, it is my hope that I have presented here a document which will serve as a introductory overview for Cheops users and programmers.

Bibliography

[1] Acetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., and Young, M., "Mach: a new kernel foundation for UNIX development", *Proceedings of USENIX 1986 Summer Conference*, pp. 93-112.

[2] Adelson, Edward H., Simoncelli, Eero, and Hingorani, Rajesh, "Orthogonal pyramid transforms for image coding". *Proceedings SPIE*, 1987.

[3] Allworth, S.T. and R.N. Zobel, Introduction to Real-Time Software Design, Springer-Verlag New York, NY, 1987.

[4] Anderson, A.J., Multiple Processing: A System Overview, Prentice Hall, New York, NY, 1989

[5] Bender, W., Bove Jr., V.M., Lippman, A.B., Liu, L. and Watlington, J.A., "HDTV in the 1990's: Open Architecture and Computational Video", in *HDTV World Review, the journal for high definition and advanced television technology*, Volume 1, Number 3, 1990.

[6] Birrell, A. and B. Nelson, "Implementing Remote Procedure Calls", ACM Transactions on Computer Systems, vol. 2, no. 1, February 1984, pp. 39-59.

[7] Blount, A., *Display Manager for a Video Processing System*, SB Thesis, MIT, Cambridge, MA, 1991.

[8] Bove Jr., V.M., and A. B. Lippman, "Scalable Open Architecture Television," in *A Television Continuum- 1967 to 2017*, pp. 210-218, *SMPTE Journal, 101*, White Plains, NY, 1991.

[9] Burt, P.J., and Adelson, E.H., "The Laplacian Pyramid as a Compact Image Code," *IEEE Trans. on Communications*, COM-31, April 1983, pp. 532-540.

[10] Chambers, F.B., Duce, D.A., and Jones, G.P. (editors), *Distributed Computing*, London: Academic Press, 1984.

[11] Coulouris, G.F. and Dollimore, J., 1989. Distributed Systems, Addison-Wesley Publishing Company, Workingham England, 1989.

[12] Hewlett, G. and Becker S., *The Cheops Software Reference Guide*, MIT Media Lab Internal Memo, Cambridge, MA, April 1992.

[13] Jones, Anita K., Chansler Jr., Robert J., Durham, Ivor, Schwans, Karsten, and Vegdahl, Steven R., StarOS, a Multiprocessor Operating System for the Support of Task Forces, *Communications of the ACM*, 1979, pp. 117-127.

[14] Lippman, A.P., "Perfectly Scalable Video" Technical Report, MIT Media Lab, May 1990

[15] Popek, G., and Walker, B. (editors), *The LOCUS Distributed System Architecture*, MIT Press, Cambridge, Mass., 1985.

[16] Shen, Irene J., *Real-Time Resource Management for Cheops: A Configurable, Multi-Tasking Image Processing System*, SM Thesis, MIT, Cambridge, MA, 1992.

[17] Stampleman, J.B., *Scalable Video Compression*, SM Thesis, MIT, Cambridge, MA, 1992.

[18] Sun Microsystems, Sun Network File System (NFS) Reference Manual, Sun Microsystems, Mountain View, Calif., 1987.

[19] Thacker, Charles P., Stewart, Lawerence C., and Satterthwaite, Jr., "Firefly: A Multiprocessor Workstation", *IEEE Trans. on Computers*, Vol. 37, No. 8, August 1988.

[20] Watlington, J. and V.M. Bove, Jr., "Cheops: A Modular Processor for Scalable Video Coding," *SPIE Vol. 1605*, Bellingham, WA, 1991, pp. 886-893.

[21] Watlington, J., *The Cheops Hardware Reference Guide*, MIT Media Laboratory Internal Memo, Cambridge, MA, 1990.

[22] Woods, J.W. (editor), Subband Image Coding, Kluwer Academic Publishers, Norwell, MA, 1990.

[23] Woods, J.W., and O'Neil, Sean D., "Subband Coding of Images," *IEEE Trans. on Acoustics, Speech and Signal Proc.*, ASSP-34, Oct. 1986, pp. 1278-1288.

[24] Wulf, W., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C., and Pollack, F., "HYDRA: The Kernel of a Multiprocessor Operating System", *Communications of the ACM*, Vol. 17, No. 6, June 1974, pp. 337-345.

Acknowledgments

Many thanks go to all those in the Entertainment and Information Group for their contributions to the work related to my thesis.

...to Allan Blount for a well-written (almost), and working initial version of the Magic Seven Operating System

...to Irene Shen for her hard labor on the resource manager

...to John Watlington for keeping the machine running (most of the time) for me

...to Shawn Becker for meeting and far exceeding his title of Cheops Software Czar

...to Dr. V.M. Bove, my advisor, for his good humor, patience, guidance, and leadership of the Cheops team

...to my readers, Dr. Steve Benton, Professor of Media Technology, MIT Media Lab; Dr. David Tennenhouse, Assistant Professor, MIT laboratory of Computer of Science; and Dr. Kenneth W. Haase, Jr., Assistant Professor of Media Technology, MIT Media Lab

...to Mom, Dad,.Susan & Julie for support and trying to understand "now what exactly do you do"?

...to Christine. I love you.

Appendix

Notes for Cheops Multiprocessing Software Designers

I describe here a few important notes for multiprocessor Cheops software developers. These are additional to the information in the Cheops Users Guide. The following sections deal with two boards, but may be intuitively extended for up to four boards.

7.1 Setting up the Hardware

Two switches on the P2 processor module must be set properly. I suggest the following configuration:

Board one	
SCSI ID switch:	4
Module ID switch:	0

Board two	
SCSI ID switch:	5
Module ID switch:	1

The double SCSI cable (labelled SCSI Beast) must be plugged into both boards. Furthermore, the host computer must be booted with Cheops in this state.

7.2 Operating System Differences

All system call behave in the same manner as with the single processor Magic Seven Operating System. All that changes is the identification numbers of processes, ports, and

semaphores. For example, board zero has processes 0-15, while board one's processes have ID's 16-31.

7.3 Executing Different Processes on Different Boards

The **chex** utility still is used to download and execute code on Cheops. Type 'scsi4' in the window which you want to run **chex** for downloading and executing on board zero. (with switches set as prescribed earlier in this appendix). Type 'scsi5 in the window you want to run **chex** for downloading and executing on board one.

7.4 Downloading New Versions of the Operating System

I recommend burning new ROM's for new versions, but while debugging, a utility has been written to download and begin Magic Seven from RAM on two boards. Simply type 'm7-srcb'.