Design, Implementation, and Performance of a Scalable Multi-Camera Interactive Video Capture System

Martin Frankel and Jon A. Webb

23 June 1995
CMU-CS-95-162

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

Abstract

We describe a system for interactive, real time, multi-camera video capture and display. The system uses largely general purpose, programmable hardware, and as a result is flexible and expandable. The computational framework and data storage is provided by the iWarp parallel computer, which we describe in light of the performance requirements of real time video. Video display is accomplished using a High Performance Parallel Interface network to write to a high resolution frame buffer. Video can be displayed as it is captured, with only a single frame latency. We provide interactivity with a VCR-like graphical interface running on a host workstation, which in turn controls the operation of the capture system. As a whole, this system allows a user to interactively monitor, capture, and replay video with the ease of use of a VCR, yet with flexibility and performance that is unavailable in all but the most expensive digital VCRs. We describe the implementation in detail, and discuss possible future enhancements.

This research was partially supported by the Advanced Research Projects Agency of the Department of Defense under contract number F19628-93-C-0171, ARPA order number A655, "High Performance Computing Graphics," monitored by Hanscom Air Force Base. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, the Department of Defense, or the U.S. government.

ACM Computing Reviews Keywords: B.4.2 Input/Output Devices, B.4.3 Interfaces, C.3 Special-purpose and application-based systems, C.4 Performance of systems, I.2.10 Vision and scene understanding, I.4.1 Digitization

1. Introduction

For the purposes of a fast, accurate multi-baseline stereo vision system currently under development [1], a video capture and display system was required to meet the following specifications:

- multiple cameras
- high frame rate (30 Hz)
- high resolution and image quality
- interactive, flexible user interface

In particular, in order to provide an interactive capture process, it is necessary to include video display capabilities. Ideally, the video would be displayed during capture, providing immediate feedback to the user.

Our system meets all of these goals, and provides considerable flexibility for future expansion. In the current implementation it supports four synchronized cameras sampling 512x480 8-bit grayscale images at 30 Hz. The foundation of this system is an iWarp parallel computer [2][3], which manages the overall data flow. Video input to the iWarp is performed by locally developed hardware. The video data is stored in the iWarp's local memory, and simultaneously sent via a High Performance Parallel Interface (HiPPI) network to a frame buffer, where all four images are displayed at full resolution in real time. The user directs this process with an X Windows application running on a workstation.

2. iWarp Architecture

The iWarp computer we are using has 64 cells in an 8x8 array. Each cell consists of a processor, its local memory, and support hardware. The processor itself consists of a *computation agent* and a *communication agent*; the computation agent is under program control, while the communication agent transparently supports the communication needs of the processor. Each cell is connected to each of its four nearest neighbors by a full-duplex, 40 MB/s physical bus; the boundaries of the array are connected together, resulting in a torus.

The iWarp executes a single *array program* at a time. An array program consists of a set of programs which execute on individual cells, a mapping of programs to cells, and a set of logical communication pathways to be established between processors. Before an array program is executed, the entire array is reset and a resident monitor, the iWarp Run Time System (iW/RTS), is loaded onto every cell.

Each cell has 512kB of fast static RAM, which is used for the iW/RTS, user program, and data. The cells in rows 0-3 also have 16MB each of slower dynamic RAM, which in our case is used for video data storage. With a combined total of 512MB of dynamic RAM, these cells can store roughly 17 seconds of four camera full frame rate video.

The iWarp provides hardware expandability via two mechanisms. First, special purpose *auxiliary* cells can be connected to the array. These cells contain standard iWarp processors and communicate with the rest of the array normally; however, they also perform a dedicated hardware function. One such cell, the Sun Interface Board (SIB), provides mechanisms whereby a host workstation can control the iWarp array and provides some I/O capability to the array program. Another pair of auxiliary cells, the HiPPI Interface Boards (HIBs), provide extremely high bandwidth, full duplex communications to other devices on a High Performance Parallel Interface (HiPPI) network.

While auxiliary cells offer tremendous flexibility, the design effort is considerable. Alternatively, each general purpose iWarp cell has an external memory bus to which a memory-mapped I/O device can be connected. The iWarp's 10 MB/s synchronous memory access rate enables high bandwidth I/O to be performed via this connector, as long as the control requirements are relatively simple. The video interface, described below, takes advantage of this external memory bus.

Figure 1 shows the configuration of the iWarp array used in our research.

2.1. Video Interface

We use four black and white, NTSC interlaced video cameras. The NTSC standard specifies 525 lines per frame, divided into two interlaced fields. Each field contains roughly 240 lines of useful information, with the remainder of the field devoted to the vertical retrace interval.

With 525 lines per frame and 30 frames per second, each scanline is roughly 63.5 microseconds (µs) long. Horizontal retrace and blanking take up some of that interval, leaving a useful video portion of about 51.2 µs. Using a 10 MHz sampling rate provides 512 pixels of horizontal resolution. The result is a 512x480 image for each frame, consisting of two fields which are captured 1/60th of a second apart. The overall data rate is slightly under 30 MB/s; however, data is captured in bursts, with a burst rate of exactly 40 MB/s, and with short pauses between lines and longer pauses between fields.

The actual video sampling is performed by specialized hardware [4] which performs simultaneous A/D conversion of four synchronized video signals. Each A/D converter produces an 8 bit intensity for each pixel. The four pixels are concatenated into a single 32-bit word. This word is read from the external memory bus of a general purpose iWarp cell, the *capture cell*.

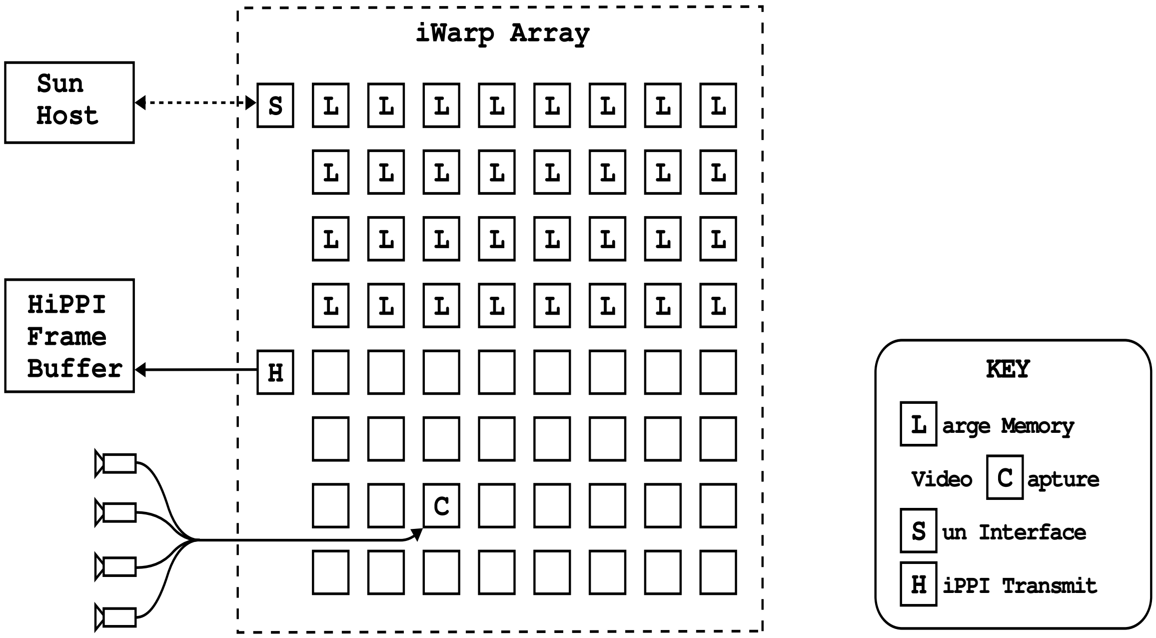


Figure 1. Hardware overview

The design of the capture hardware was considerably simplified by not including a buffer between the A/D converter and the iWarp interface. However, the lack of buffering makes the task of the capture software particularly difficult. Since we are using a 10 MHz sampling rate, a sample must be acquired on alternate cycles of the 20 MHz system clock. Since the memory read instruction takes two clock cycles, the capture cell must read a sample from the video capture hardware and transmit it to another cell in the array in every instruction. Any delay, even a single clock cycle, causes unacceptable error in the resulting image.

The fact that this is possible, and indeed practical, is due to several notable features of the iWarp architecture. A brief summary of these features follows.

3. iWarp Features

3.1. Systolic communication model

In the traditional message passing model, messages are accumulated in memory and transmitted as a unit to a destination cell. In our application, however, the overhead imposed by memory access and message processing would make real-time performance completely impossible. At the very least, storing each value in a memory buffer after it is sampled would require a second memory access in the inner loop, reducing the frame rate to 15 Hz. Even greater performance loss would be likely when the accumulated message is transmitted. However, with systolic communications, single words are transmitted and passed independently through the array; in other words, the message grain size can be as small as a single word. This obviates the need for buffering before transmission.

An iWarp program uses the systolic capability of the iWarp via special registers called *gates*, which serve as endpoints for communication. When the processor writes a value to a gate, it is automatically transmitted to a predefined destination; when a processor reads from a gate, the first available data from a predefined source is provided. The communications agent has short queues on each gate, transparent to the user, which provide limited buffering. If the receive queue is empty when the processor attempts to read from a gate, the instruction blocks until a word is received. A write to a gate may also block in a similar manner. Since gates are represented as registers, no special instructions are necessary to send and receive data, and indeed, operands or results of an instruction can be received or transmitted via gates in exactly the same manner as an ordinary register access.

3.2. Efficient data routing

The iWarp provides hardware-assisted data routing with *logical pathways* and *logical ports*. A logical pathway is a data transport mechanism with a predefined source and destination cell; each end of the pathway is assigned a logical port, which the program uses to send or receive data over the pathway. Data passes through the cells along the route with negligible delay and no performance impact upon the intermediate cells. Furthermore, several logical pathways can pass over a single physical bus, without added overhead, and without contention until the full 40MB/s bandwidth of the physical bus is reached. These mechanisms allow the user to largely ignore the physical topology of the array and the details of data routing, multiplexing, and transport.

3.3. Zero-overhead loops

The iWarp processor has dedicated support for automatic loop iteration and branching. Using a special loop count register and a dedicated bit field in the instruction set, a loop need not contain any instructions to decrement the loop counter or perform the branch; that is all performed by dedicated hardware, in most cases without any time penalty or overhead.

Most of the work performed by the video capture program is done in very small loops, usually only two or three instructions, which must be executed continuously. The additional overhead imposed by performing loops in software would make the software design task more difficult, if not impossible. For instance, although loops could be unrolled to reduce overhead, they must remain small enough to fit within the processor's instruction cache.

3.4. Real time performance

The iWarp has a very rudimentary operating system and no support for multitasking. The few interrupt-based services provided by the iW/RTS can easily be disabled. When this is done, the execution of a non-I/O-bound array program becomes largely deterministic. (Instruction caching limits the predictability to some extent.) Furthermore, each iWarp processor contains a hardware timer which decrements every 8 clock cycles. As a result, the behavior and timing of an array program can be made predictable, reliable, repeatable, and measurable down to a sub-microsecond level. This is necessary to our application, since an unpredictable delay on any cell which processes the unbuffered video data stream could cause loss of data at the capture cell.

4. Data Flow Part I: Capture and Storage

The inner loop which captures a single line of video data and sends it continuously to another cell could be as simple as the following[1]:

```
                loop 512                                ; executes once
        endloop load (video_ADC_address),gate0          ; executes 512 times
```

However, this causes two problems:

a. A four-byte word is being transmitted every other clock cycle within the video scanline, for a burst transfer rate of 40 MB/s. This is precisely the maximum theoretical bandwidth of the physical busses used to connect cells. However, a bug in the handshaking mechanism between cells results in unacceptable jitter when this full bandwidth is used entirely with data being read directly from memory (as opposed to a combination of data passed from other cells and data from memory).

b. Other cells must process the data stream, and if one sample arrives every two cycles, not much time is left for processing.

In order to reduce the peak transfer rate from 40 MB/s, we split the video data stream, by transmitting alternate samples over two pathways to two identical sets of storage cells. As a result, each video field is stored as two separate half-fields, one with the even-numbered pixels from each scanline, and the other with the odd-numbered pixels from each scanline. The inner loop which actually executes on the capture cell is as follows:

```
                loop 256                                ; executes once
                load (video_ADC_address),gate0          ; executes 256 times
        endloop load (video_ADC_address),gate1          ; executes 256 times
```

The storage cells are cells 0 through 31, each of which is has 16 MB of dynamic RAM. These cells make up the top half of the array, as it is shown in Figure 2. Of these 32 cells, the left 16 store the half-fields containing the even-numbered pixels, and the right 16 store the half-fields containing the odd-numbered pixels. The storage cells in each half are connected together in a serial, unidirectional fashion. The capture cell is connected to the first storage cell in each chain, and the last storage cell is connected to the display processing cells, which are discussed in Section 5. In all, these cells can store 2048 half-fields, which make up 512 frames, or slightly over 17 seconds of 30 Hz video.

When data is being captured, every storage cell passes the incoming data, uninterrupted, to the next cell in the chain. One of these cells also copies the data into its local memory; the control mechanism which determines which cell should store a given frame is discussed in Section 6.

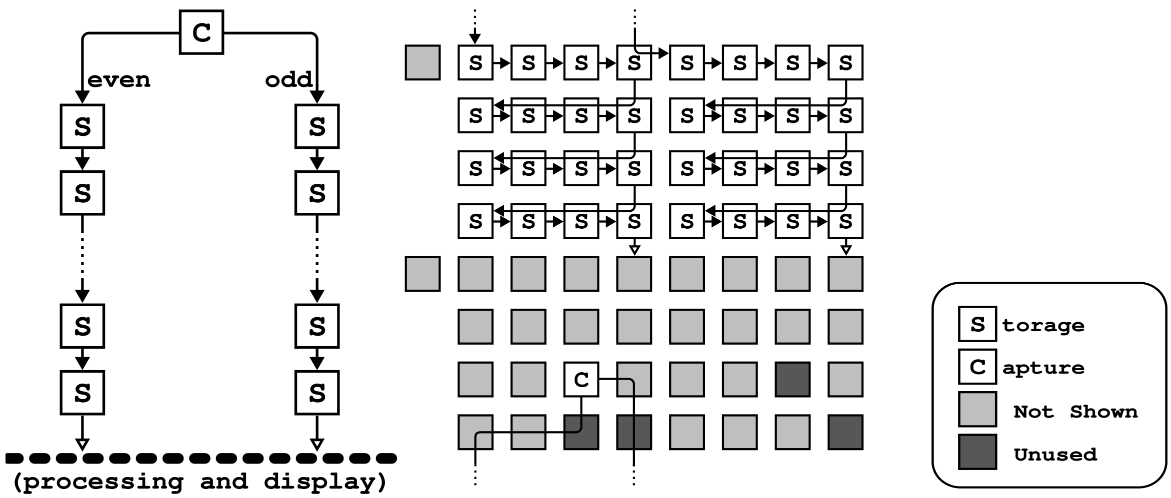
This design lends itself well to playback after capture as well as monitoring during capture. In order to display a particular frame, the storage cell containing that frame transmits the frame data, and every subsequent cell passes it on. The data reaches the display processing cells exactly as if it had just been captured.

4.1. Image Display

We considered three possible methods for displaying video during capture:

a. On the host workstation. It would be possible to send image data from the iWarp, via the SIB, to the host, and then display it on the host's monitor. However, the performance of this approach would be

1. The assembly language syntax used here is modified slightly for clarity.



a) Data Flow **b)** Layout **c)** Key

Figure 2. Capture and Storage

completely unacceptable. The SIB-to-host communications have low bandwidth and very high, unpredictable latency imposed by the Unix operating system. Only very small, “thumbnail” images could be transmitted, and only at a very low frame rate (a few Hertz); even then, uniform frame rates could not be guaranteed.

b. On television monitors. In addition to the video A/D converters described above, we have D/A converters which operate similarly, and provide an NTSC signal suitable for display on high quality television monitors. Unfortunately, the D/A converters are not as reliable as the A/D converters, and exhibit problems with synchronization and stability. Furthermore, using separate monitors for each camera is inelegant and scales badly.

c. On a dedicated framebuffer via a high speed network. This is the approach we chose to implement.

High Performance Parallel Interface (HiPPI) is a connection oriented, switched, very high bandwidth network protocol [5][6]. It provides guaranteed 100 MB/s connections between devices, using crossbar switches to create small networks [7]. A recent collaboration between Carnegie Mellon University and the Network Systems Corporation resulted in the development of HiPPI transmit and receive hardware for the iWarp computer [9][10]. Like most HiPPI interfaces, the iWarp HiPPI Interface Boards (HIBs) do not achieve the full theoretical data rate of HiPPI. The transmit HIB (X-HIB) achieves a reliable 42.5 MB/s transfer rate.

We also have a HiPPI frame buffer developed by Network Systems Corporation. The frame buffer receives data in a simple protocol (Frame Buffer Protocol) layered over the HiPPI framing protocol (“raw” HiPPI). It can be configured for any window size up to 1024x1024, and can display 8-bit grayscale, 8-bit indexed color, or 24-bit color images using a number of pixel formats. The frame rate is limited only by the HiPPI bandwidth and the size of the window being updated. Therefore, using the X-HIB and the HiPPI framebuffer, we can easily display four 512x480 images tiled in a 1024x960 window at 30 Hz.

It should be noted that, in order for an expanded implementation to display more than four images, we would need to subsample the images or allow different images to be multiplexed onto the display at different times, because of the fixed resolution of the monitor. Either of these possibilities could be implemented quite straightforwardly and would not greatly hamper the usability of the system.

HiPPI communications on the iWarp are handled by a software suite running on the HiPPI Interface Boards, called the HiPPI Streams Interface (HSI) [8]. The HSI serves to insulate the array program from network-specific or protocol-specific details. The mechanism of this abstraction is particularly powerful: the iWarp's logical pathways. When a connection is established, the user program tells the HSI over which pathways it will be transmitting or receiving data, and how that data should be packetized; from that point on, as far as the user program is concerned, network communications are no different from ordinary inter-cell communications. The HSI has built-in support for the protocol used by the framebuffer.

5. Data Flow Part II: Processing for Display

A significant amount of processing is required before the video data can be sent to the HSI and the framebuffer. The data arrives with three levels of interleaving, and must be sent to the frame buffer as a single 1024x960 window. The necessary processing tasks are as follows:

a. Pixel interleaving. Data arrives on two separate pathways, one with even pixels, the other with odd pixels.

b. Image demultiplexing. Each word of incoming data consists of four 8-bit pixel values, one from each camera. These must be separated so that each image can be transmitted independently.

c. Line interlacing. The even lines of an image arrive first, followed by the odd lines.

d. Image tiling. The resulting four images must be combined for transmission to frame buffer.

These tasks could conceivably be performed in many different ways. However, the requirement that they be implemented using the 31 available iWarp cells, each of which has only 512kB of memory, yet provide 30 Hz frame rate with as little latency as possible, limits the possibilities considerably. For example, during a video scanline, two words of data arrive at the display processing cells every four clock cycles; these must be rearranged and then sent at an average rate of 30 MB/s over a single pathway. Even if the task were split evenly across all available processors, without any blocking or overhead, only 21 clock cycles would be available to process each byte of data sent to the frame buffer.

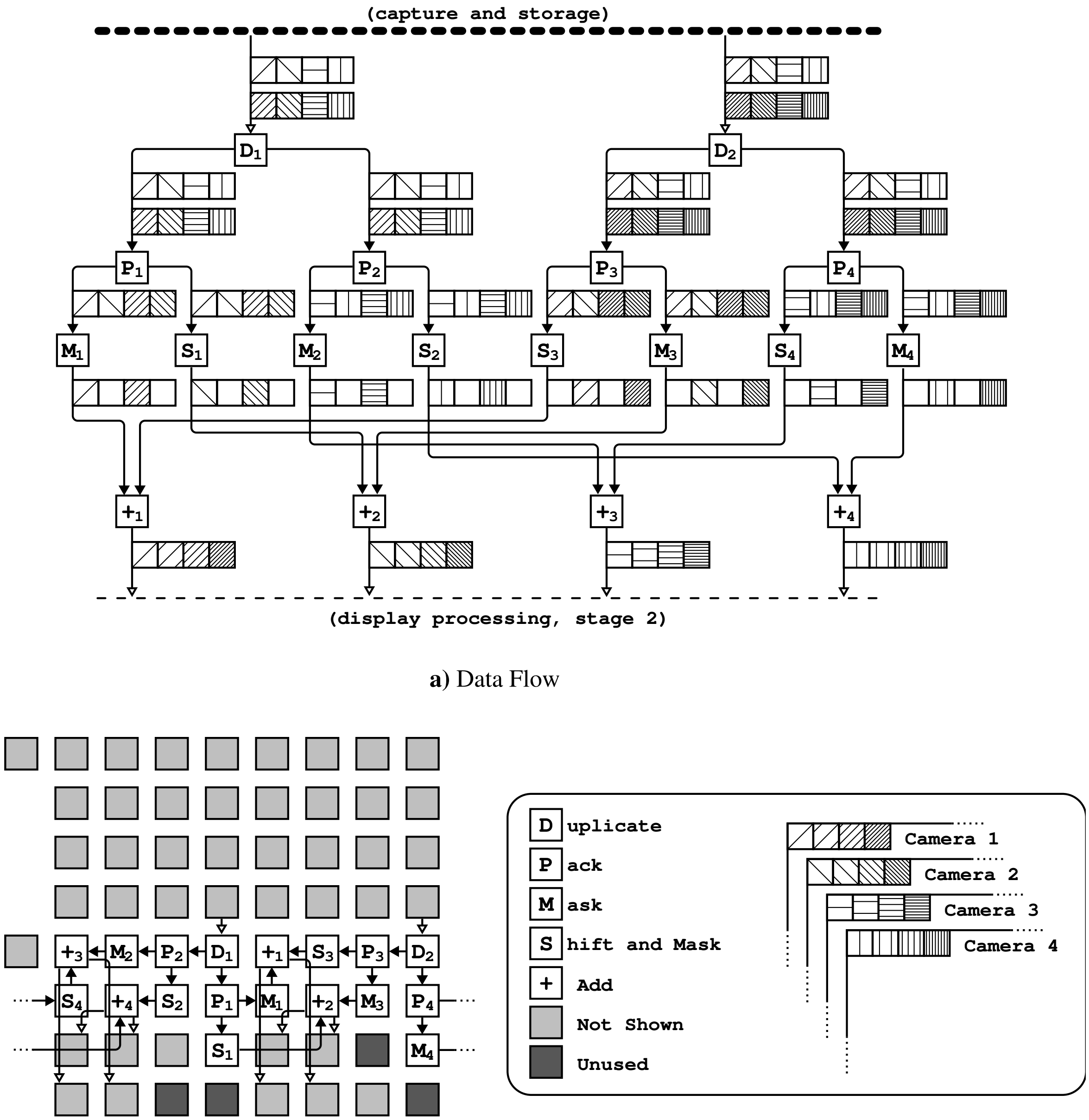
Our approach involves two stages, which use different operational paradigms. The first stage, which performs pixel de-interleaving and channel demultiplexing, operates on the data as a continuous pixel stream, ignoring scanline and frame boundaries. Consecutive cells perform simple operations on the data stream and then pass data on to other cells for further processing. The second stage, which performs line de-interlacing and image tiling, uses double-buffering to store complete frames and then send them, line by line, in a rearranged order.

Stage 1 is best understood by considering the first four pixels of a frame arriving from the storage cells. Due to pixel interleaving, the first and third pixels arrive on one pathway, while the second and fourth arrive on another pathway. Due to pixel multiplexing, the incoming images are combined, with a single word containing the pixel values for all four images. The output of stage 1 should be four words on four separate pathways, each one of which contain the first four pixels of an image. Subsequent blocks of four pixels are processed identically. The operations performed by each cell to achieve this goal are illustrated in Figure 3.

Stage 2 is somewhat simpler, consisting of only two basic types of cells: 8 *buffer* cells, and one *switcher* cell. The 8 buffer cells are broken into two sets of four, which are used to double-buffer the four images. While one set is buffering data from the capture cell, the other is transmitting data to the switcher cell; at the end of the frame, the two sets of buffer cells trade roles. This process is illustrated in Figure 4 by physical switches; in reality, the switching is performed by the combine cells (from stage 1) and the switcher cell, which bind alternating sets of logical ports on alternating frames.

Line interlacing is performed trivially by the buffer cells; they read the interlaced video data directly into a memory buffer, and then send individual scanlines from the buffer in such a way that the switcher cell receives non-interlaced data.

The switcher cell is responsible for tiling the images and sending them to the X-HIB. The upper half of the window consists of images 1 and 2 side-by-side, which is accomplished by passing individual scanlines of images 1 and 2 in alternation. The lower half of the window is filled with images 3 and 4 in an identical manner.

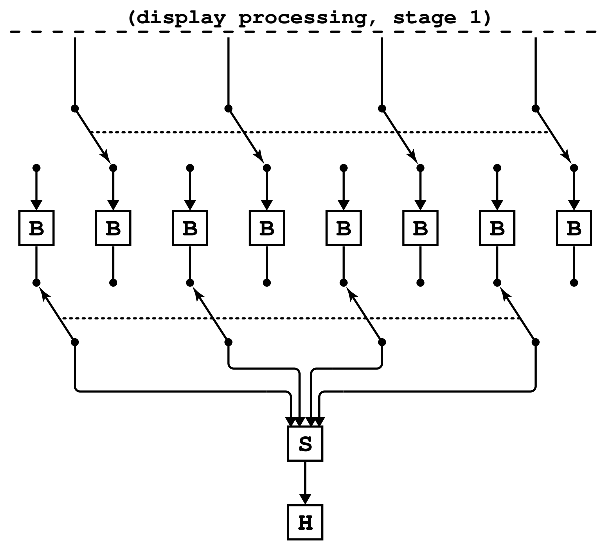


a) Data Flow

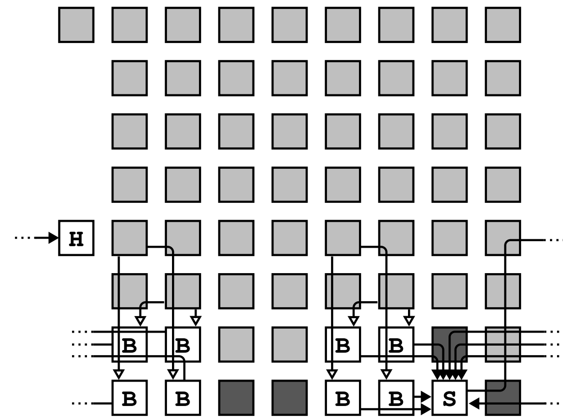
b) Layout

c) Key

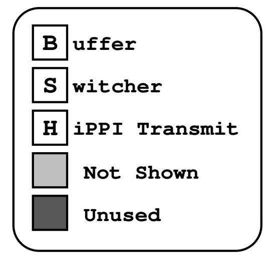
Figure 3. Display Processing – Stage 1



a) Data Flow



b) Layout



c) Key

Figure 4. Display Processing – Stage 2

6. Control and User Interface

The videocassette recorder (VCR) is the most basic and widely used device for video capture. It is interactive and flexible, yet familiar and easy to use. It provides most of the controls we need to capture and store video from multiple cameras. Hence, we sought to emulate the functionality and user interface of a VCR as much as possible.

This was achieved with a graphical user interface (GUI) running on the host workstation. A snapshot of the interface is show in Figure 5. This interface was programmed using Tcl, an interpreted programming language, and Tk, a toolkit under Tcl which allows rapid prototyping and development of X Windows applications. Using a small library of C routines which we developed, the Tcl program starts the video capture array program on the attached iWarp array and exchanges short messages with the array. All communication with the array is performed via the Sun Interface board, using a communications facility called *imsg* supported by the iW/RTS.

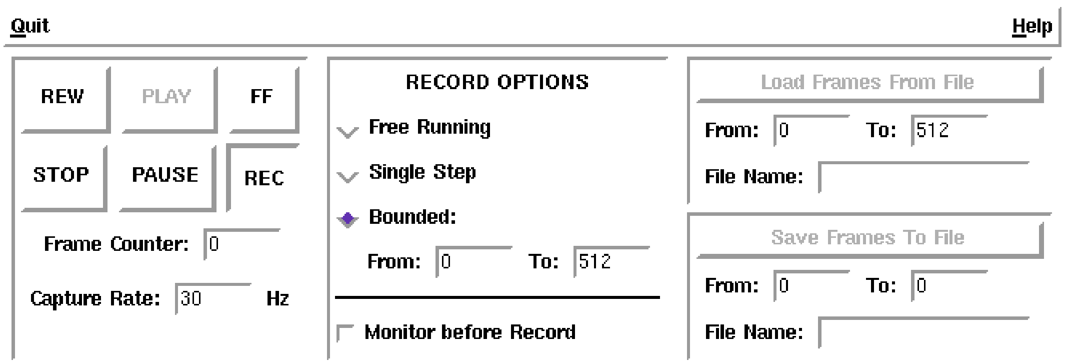


Figure 5. GUI for video capture program.

The GUI program sends messages to the SIB, and receives return messages, in what we call the *external protocol*. The external protocol concerns actions by the user, and is restricted to a relatively coarse level of control. The SIB then performs real-time, frame-by-frame control of the rest of the iWarp array, using the *internal protocol* to communicate with the other cells.

The external protocol is used to transmit user actions to the array, and to update the state of the GUI (to reflect, for instance, the number of frames available for recording). However, the imsg mechanism used by the external protocol has a number of drawbacks which limit its utility. imsg has a high, unpredictable message latency due in large part to the Unix operating system on the host, so it is impossible for the host to exercise real time control of the array. Also, imsg sends are inherently blocking in nature, so the SIB cannot send a message to the host and continue to control the array in real time. This means that certain desirable features cannot be implemented, such as a frame counter on the GUI which increments during recording or playback to reflect the actual current frame number.

The internal protocol consists of short control messages, which are sent over a dedicated pathway connecting every cell in the array in a closed loop. During the video retrace interval between frames, the SIB sends a message, e.g. to capture a frame into memory at a specific location. Each cell copies and passes the message, until it returns to the SIB. From the content of the message, each cell can determine what it needs to do for the next frame. The message takes about 47 µs to pass through the entire array, leaving time to spare before the end of the 1.25 ms vertical retrace interval. After the ensuing frame is complete, all of the cells switch back to the message pathway and wait for further instructions.

6.1. Performance

The resulting system works as described, and provides all the usual capabilities of a conventional VCR — play, record, fast forward, cue, and rewind — through a convenient graphical interface. It also supports many capabilities important in research in multi-baseline stereo research, such as variable capture rates and integration with a computer system (allowing frame capture to be triggered under program control, and providing accurate per-image timestamps, allowing the video data to be related to data from other devices).

The peak transfer rate within iWarp is 40 MB/s. The average transfer rate within iWarp and over HiPPI is 30 MB/s, a significant fraction of the peak 42.5 MB/s achievable with the current hardware interface. We have recently expanded the system to support four video capture boards, for an aggregate peak bandwidth within iWarp of 160 MB/s.

One might think that a "HiPPI VCR" such as this is merely an expensive demonstration of the same capabilities available cheaply from consumer electronics. But this is not true. Conventional VCRs do not support frame-synchronized capture of video data from multiple cameras, do not allow variable capture rates, do not record data reliably in digital form, and, most important in our application, are not tightly integrated into a computer system that supports transfer of the video imagery to disk for processing, or transmission to other high performance computing devices for further processing. Measured against a system that provides such capabilities, our system is quite cost-effective (iWarp's cost was roughly $500K in 1992).

7. Future

There are a number of paths for future expansion and improvement which can easily be added to our existing architecture. We are considering:

1. Additional camera support. Video interfaces can be attached to up to 8 cells, allowing simultaneous capture from 32 synchronized cameras. Naturally, there are trade-offs involved, since the storage and computational capabilities of the iWarp are fixed. Some features which may be sacrificed in order to support a larger number of cameras include frame rate, storage capacity, and display capability.

2. External storage over HiPPI. The Parallel Data Laboratory at Carnegie Mellon University is developing a large (100+ GB) storage server which should provide high bandwidth disk storage over HiPPI [11]. Much of the software developed for image display will be directly applicable to image storage over HiPPI, including the user interface and video processing. Furthermore, since additional processing could be performed in the iWarp cells currently devoted to storage, a greater number of cameras could be supported at high frame rates. However, the limited throughput of the HiPPI adapter and storage server may necessitate data compression, possibly with additional impact on performance.

3. External stereo vision calculation over HiPPI. In the past, stereo vision processing was performed on the iWarp. Since more powerful computers are now available, the iWarp could serve as a dedicated video capture system and feed data over HiPPI to another computer (e.g. Intel Paragon) which would do further processing. From the perspective of the iWarp, this task is very similar to data storage on an external server, requiring only minor changes in software between the two applications.

Hence, this system has a clear path of future growth. Ultimately, it could lead to a flexible, modular, expandable video capture and processing network based on HiPPI, using largely non-specialized facilities and hardware.

8. Bibliography

[1] Kang, S. B., J. A. Webb, et al. (1994). An Active Multi-baseline Stereo Vision System with Real-time Image Acquisition. ARPA Image Understanding Workshop, Monterey, CA, 1325-1334, Morgan Kaufmann.

[2] Borkar, S., R. Cohn, et al. (1988). iWarp: An Integrated Solution to High-Speed Parallel Computing. Proceedings of Supercomputing ‘88, Orlando, Florida, 330-339.

[3] Borkar, S., R. Cohn, et al. (1990). Supporting Systolic and Memory Communications in iWarp. 17th International Symposium on Computer Architecture, Seattle, WA, 154-163.

[4] Kang, S. B., T. Warfel, J. A. Webb (1994). A Scalable Video Rate Camera Interface. CMU-CS-94-192, Carnegie Mellon University.

[5] Tolmie, D. and J. Renwick (1993). “HIPPI: Simplicity Yields Success.” IEEE Network (January) 7 (1), 28-32.

[6] ANSI X3T9 (1991). High Performance Parallel Interface – Mechanical, Electrical, and Signalling Protocol Specification (HIPPI-PH). ANSI X3.183-1991.

[7] ANSI X3T9 (1993). High Performance Parallel Interface – Physical Switch Control (HIPPI-SC). ANSI X3.222-1993.

[8] Hemy, M. (1993). HIB Interface Manual, Revision 1. CMU-CS-93-186, Carnegie Mellon University.

[9] Steenkiste, P., B. Zill, et al. (1992). A Host Interface Architecture for High Speed Networks. Proceedings of the 4th IFIP Conference on High Performance Networks, Liege, Belgium, 1-16, IFIP.

[10] Steenkiste, P., M. Hemy, et al. (1994). Architecture and Evaluation of a High Speed Networking Subsystem for Distributed-Memory Subsystems. The 21st Annual International Symposium on Computer Architecture, Chicago, IL, IEEE, 70-81.

[11] Gibson, G., D. Stodolsky, et al. (1995). The Scotch Parallel Storage Systems. Proceedings of the IEEE CompCon Conference, San Francisco, CA, IEEE.