A Parallel Architecture for Video Processing

D Turgay ALTILAR, Yakup PAKER, A Vahit SAHINER

University of London
Queen Mary & Westfield College
Department of Computer Science
Mile End Road E1 4NS London-UK

{turgay, paker, alivahit}@dcs.qmw.ac.uk

Abstract. Video data consists of a sequence of frames that is produced at a constant rate and many applications in real-time require the processing of these frames executing compute intensive algorithms. To handle many of such applications in real-time, we developed a new architecture based on parallel processing. The parallel architecture for video processing has been developed at Queen Mary and Westfield College as a part of an European Union RACE II project called MONALISA. A multi-processing kernel and a high level software environment called SAPS (self adapting parallel server) model has been developed for this architecture. This environment makes it possible to introduce a number of load balancing and data decomposition schemes that can be realised automatically in real-time by the kernel without any explicit inputs from the user. The developed architecture aims at applications such as image analysis algorithms, camera tracking, mixing captured foreground images and synthetically generated background images using depth values in real-time for Virtual Studios. In this paper we focus on the software architecture, frame buffer management and frame buffer access protocols. The system architecture and hardware is explained first. The standard frame buffer access protocol, SFBA, and the dedicated frame buffer access protocol, DFBA either of which address different needs of video processing are introduced. System performance evaluation, benchmark results and an analysis of DFBA protocol are given in detail.

1 Introduction

Technologies used in the broadcasting industry are changing rapidly as digital processing is entering all aspects of TV programme making and distribution. With the advent of digital TV and interactive multimedia over broadband networks, the need for high performance computing for broadcasting is stronger than ever. Processing numerically a video sequence requires considerable computing. One of the ways to cope with the demands of video sequence processing in real-time, we believe, is combining parallel processing with frame buffer technology.

Computer graphics techniques like rendering and radiosity and most of the video processing techniques like mixing, chroma-keying are compute intensive processes. These algorithms have been parallelised for various types of computers [Whit94]. Most of the parallelised algorithms suffer from the unbalanced loading because of highly irregular and unpredictable data processing power needed over a given size of data, i.e. data dependency [Sing94]. Although video sequences are normally synchronous streams with a standard number of frames per second, for some applications this is not true. For example if an algorithm is data dependent then the frame processing time is not constant. An example of such an application is MPEG compression/decompression, which has four different types of coded frames, I,P,B, and D to be coded/encoded [Stei94]. A proper load balancing algorithm is needed to cope with both data dependent and data independent cases.

Data decomposition schemes for a stream based data are similar to image processing ones. But video streams have some additional properties that should be taken into consideration. Moreover, the frame buffer management enables us to access a given rectangular part of a frame.

The properties of a video stream considering data decomposition are:

- Sequence of images refreshed with a fixed period
- A single image formed by three frames R, G, and B or two chrominance Y, U, and one luminance V

Bearing the properties of video streams in mind, the following decomposition schemes are possible for parallel processing:

- Consecutive frames processed by distinct processors
- Decomposing a single frame (or frame section) over a number of processors and recombining the frame sections into a single frame

Video sequence processing in addition to processing individual frames of a single sequence, like chroma-keying, could also require processing more than one sequence at a time such as mixing two frames by using Z depth values.

Our architectural approach is based on a scaleable shared address space multi-processors using Single Program Multiple Data (SPMD) parallelism and is aimed at real-time processing of broadcast quality video sequences. Automatic data distribution, load balancing, utilisation of the processors are the main concerns to be addressed.

At QMW, we developed a parallel video accelerator, ML-PVA, for the MonaLisa[1] project supported by the European Union, for real-time video processing. The project brought computer technology and image processing concepts into studios towards creating a Virtual Studio [Blon96]. It comprises a specially designed hardware and software architecture. The hardware architecture is aimed at the use of a processor pool attached to a frame buffer via a high speed bus. A multi-processor kernel and a novel server mechanism have been developed and implemented. SAPS (Self Adaptive Parallel Servers) is the model used for the system software which addresses the issue of accelerating pre-selected and computationally heavy procedures in a target application. This is done by building a server box, i.e. a pool of processors, which is transparent to its users and has the potential of accelerating a library of procedures on request [Sahi95].

ML-PVA hardware and software architecture has been developed for addressing the following issues:

- Acceleration of time consuming image analysis algorithms like real-time camera tracking
- Acceleration of illumination of preprocessing
- Acceleration of computer graphics algorithms
- Capturing of camera output, bluebox output and graphics workstation output
- Multi-frame delay of captured foreground images to compensate pipeline delay from camera tracking and rendering
- Real time mixing of captured foreground images and synthetically generated background images based on blue box key signal and depth values

Using this architecture a camera tracking algorithm has been realised and run successfully satisfying real-time constraints [Rout95].

The paper is organised in seven sections. Section 2 gives the description of the systems hardware. The systems software and the application programs are briefly explained in Section 3. Section 4 presents the SAPS model. Section 5 describes frame buffer accessing protocols. Evaluation of processing schemes and benchmark results are presented in Section 6. The paper ends with conclusions in Section 7.

2 The Hardware Architecture

The hardware structure can be defined under two substructures which are a graphics station front end and the Parallel Video Accelerator (ML-PVA). As seen in the Figure 1, a camera and a monitor are connected to the frame buffer respectively via an ADC and a DAC. The graphics workstation, SGI, is connected to the host of pool manager via the Ethernet. Software interfaces have been developed such as Administration Tool, Command Tool, and Studio to manage the system.

---

[1] MonaLisa (MOdelling NAturaL Images for Synthesis and Animation) is an EU supported RACE II project whose main goal is to develop a virtual reality platform by means of both hardware and software, for TV studio production and post production by mixing environment for using synthetically generated images and real images together.

A UNIX based workstation, SGI with OSF/Motif and X11 utilities, run the user interface modules, and the application programs . A Motorola 68030 based computer board, running a real time operating system, OS9, is responsible for controlling and managing the overall system which consists of a DSP based processor pool running a kernel (WKernel), and a frame buffer system with a high speed bus interface. These platforms are attached to each other via different mediums such as Ethernet, VME bus, high speed bus, and I/O channels (Fig. 1).

FIFO-buffered I/O processors (IOPs) are used to connect I/O devices and clusters to the frame buffer. IOPs are connected to 601/656 interfaces to cope with digital video signals. Another interface has been developed for DSP-IOP connections. All data transfers are controlled by the address generator consisting four address processors, each controlling one data transfer.

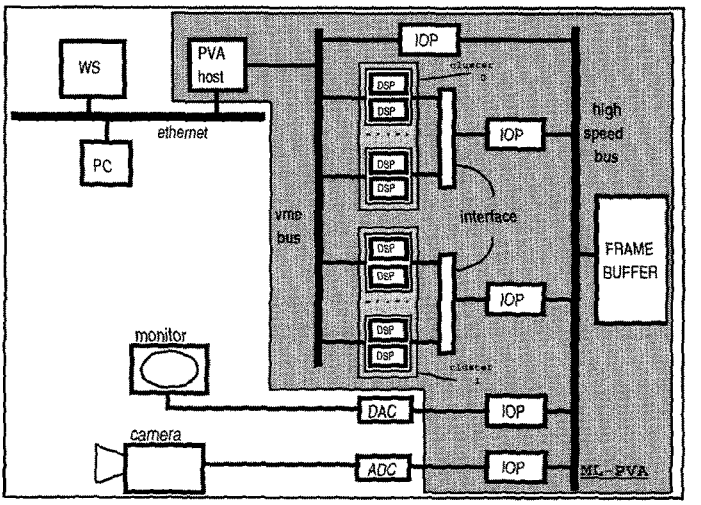

Figure 1. ML-PVA and its environment

ML-PVA comprises a 68030 main processor, a frame buffer of 32 Mbytes with 5 IOPs and an address generator, and 8 DSPs (DSP96002). The two main parts of ML-PVA are the processor pool and the frame buffer of which hardware characteristics are given in the following two subsections.

2.1 The Processor Pool

The processor pool is based on DBV dual processor boards, manufactured by LSI, containing two Motorola 96002 DSPs. There exists four boards in the current system. Two boards make a cluster attached to a single I/O processor. Boards are plugged to VME bus through the shared global A-BUS.

DBV boards operate at a speed of 40 MHz. Each DSP has a local SRAM of 256K words, one dbex and one hyperbus interface for external devices. dbex Interface is used for I/O processor attachment via a specially developed interface card. Each DSP has also front panel interfaces serving for monitoring and debugging purposes. Two DSPs on the same board also share a SRAM of 256K words and a DRAM of 4M words attached to A-BUS of both processors. They both have access to arbiter, dual RS232 port and VME Bus interface via this common A-Bus.

2.2 The Frame Buffer

The frame buffer system, ISP 400, manufactured by DVS, is used for video input/output and buffering [DVS93]. FIFO buffered I/O processors are used to connect I/O devices and DSPs to memory bank. The memory bank has a capacity of 32 Mbytes which can be expanded up to 2 GBytes. All of these devices are attached to the high speed bus with 400 Mbytes/second and all activities are controlled by a 68030 based host processor. An interface compatible with CCIR 601/656 is available for video connection and is attached to existing I/O processors.

3 Software Overview

The software consists of a collection of systems software and application modules running under three different operating systems which are UNIX, OS-9, and WKernel (developed at QMW). The communication between UNIX and OS9 has been implemented over TCP/IP and NFS. The communication between OS9 and WKernel has been implemented by RPC, based on shared data objects and VME interrupt mechanism.

On the UNIX side of the software the user interface and application tools are the main concerns. The only low level system software is the implementation of socket communication. There are three different tools developed on the UNIX side, which are the Administration Tool (AdmTool), Command Tool(ComTool) and Studio [LeFl95].
Almost all of the software developed under OS9 is low level systems software, including the device drivers, the pool manager, and the VME interrupt handler. OS9 serves as a system administrator, a supervisor, and a resource manager on the ISP-VME Bus which can be considered as the backbone of ML-PVA. OS9 communicates with UNIX and WKernels.

OS9 provides the host environment for the utilisation of the frame buffer and the processor pool. The system management processes for the ML-PVA system run in this environment. These processes consist of two distinguishable parts which are the Pool Manager and the ISP Server. The Pool Manager performs the functions of scheduling, object managing, dispatching, and name serving in our implementation [Sahi95].

The DSPs run a light weight kernel called WKernel which provides the process creation and process management functionalities for parallel server execution on the Processor Pool.

The three separate software platforms, programs running on these platforms and the nature of data used in communication for a typical client-server based application are shown in Figure 2. Client-server based approach is supported by SAPS model based on Single Program Multiple Data type of programming. SAPS provides a number of tasks running concurrently over the processor pool. Client initialises the tasks by sending a list to the Pool Manager. Prior to this, client sends data to the frame buffer if it is needed. The Pool Manager dispatches task to DSPs on their requests. DSPs read/write data directly from the frame buffer. The SAPS model and use of it is explained in the next section.

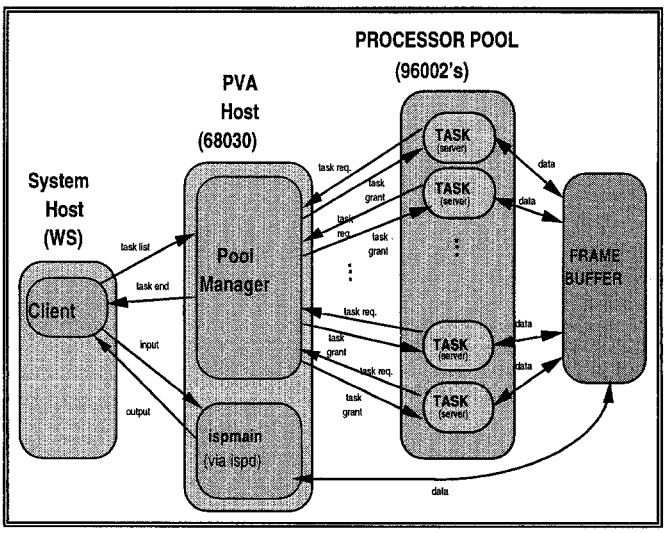

Figure 2. ML-PVA Software environments and the nature of data in communication.

4 The SAPS Model

The SAPS model is based on the Single Program Multiple Data (SPMD) parallelism [Sahi91]. In this model, a parallel application is composed of a number of copies of the same sequential program, each running on a separate processor node. This form of parallelism is widely used on message passing multi-processor systems because it has a relatively simple and well defined structure [Gabb90] [Lucc87]. SPMD style also provides a framework for developing parallel application software using sequential programming techniques, and, therefore, enables application software already developed for sequential machines to be used for parallel architectures, without undergoing major changes.

Under the SAPS model, the mechanisms for SPMD parallelism are encapsulated within servers. A server when requested executes multiple copies of an associated sequential procedure in parallel in SPMD mode. The data is provided by the client within the service request. The interface between the application programs, as clients, and the parallel servers is conveniently hidden in the procedure call mechanism which is a well-understood facility to develop modular programs.

The interaction between a SAPS and an application as its client, for the actual provision of the service, is based on standard remote procedure call which is structured as a service-request and a service-reply is also supported by a data objects scheme which enables data decomposition. Servers fetch application data via operation invocations on data objects.

The server-application (client) duality provides the means for the separation of concerns and therefore the separation of the building of servers and their utilisation by the applications. Applications are conventional sequential programs developed independent of the concerns for parallelism. The main building block of a server is also a sequential procedure; a parallel SPMD structure is obtained when this procedure is interfaced to a standard template. This scheme allows building servers using existing sequential software without major modifications.

A SAPS has a multi-process structure (Fig. 2). This structure contains the mechanisms for the reception of service requests, their processing, and the transmission of the results back. A dispatcher process, implemented within the Pool Manager, and a pool of workers form a process farm where the dispatcher farms out work to workers and each worker when becomes idle, requests for more work from the dispatcher process. It is the multiplicity of the workers that provide parallelism within the server. Workers run on the pool processors in a one node per worker fashion. The number of nodes used by the workers of a particular server is not fixed, it depends on run time availability of nodes. A server can start operating with a single worker and dynamically increase its worker population at run time as more nodes become available.

Within the SAPS structure it is the scheduler process, implemented within the Pool Manager, which is responsible for the resource management activities. It manages the configuration (it creates the dispatcher and worker processes), and it reconfigures the SAPS structure by adding or deleting workers. The scheduler coordinates its activities with other SAPS schedulers through the pool manager. The structure of a server is completely transparent to its clients. Each server has two message communication access points: one for handling service requests (service access point), and the other for monitoring the processor resources and coordinating its resource usage with other computing agents (resource management access point).

The SAPS structure as a whole is supported by a standard software template. The complete functionality of a SAPS as a generic server object is programmed within this template. To create a SAPS blueprint for a particular service all that is required is the interfacing of a conventional sequential procedure (the task proper) to a copy of the template. This is achieved via a local procedure call interface.

5 The Frame Buffer Management

The frame buffer management has an important impact on the overall performance of the architecture. In order to improve the performance of the system's throughput, a number of access schemes have been developed.

5.1 Frame Buffer Access Protocols

The frame buffer acts as a disk space under OS9 operating system which enables us to define memory allocation and memory management on the basis of disk management. A video sequence is created (allocated) prior to its use. If there is not enough space for a specified file it is not created as there is no pre-emption/swap space available. Processes running on either the 68030 main processor or the DSPs can access such a file through input/output processors (IOPs). Several IOPs can be attached to the same file, providing a powerful means of sharing the frame buffer.

The DSPs' frame buffer accesses are controlled by the pool manager via the name server, implemented on the 68030, which handles the name table, availability and access rights. Since one IOP serves four DSPs, this authorisation mechanism is essential for resolving conflicting access requests coming from within the same cluster.

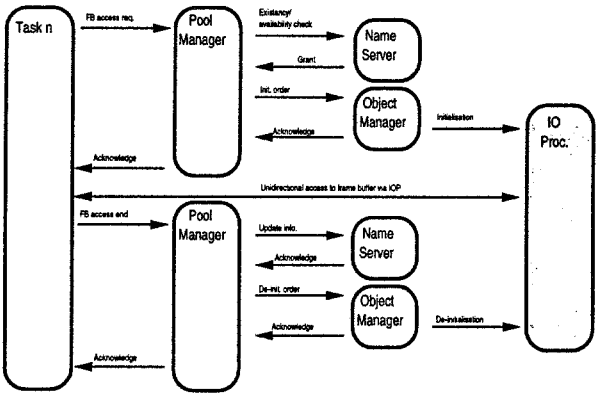

Figure 3. Standard frame buffer access protocol (SFBA)

The Standard Frame Buffer Access Protocol, SFBA, has been developed to access the frame buffer by a request/grant/free mechanism between the pool manager and any one of the DSPs (Fig. 3). Only one DSP from a cluster of four is allowed to access the frame buffer at a time. The pool manager keeps the connection between DSP and the frame buffer until DSP sends a detach command. This is a generic protocol. However, the frame buffer management, initialisation and communication itself results in considerable overhead as this is repeated whenever a new frame is required by any of the DSPs.

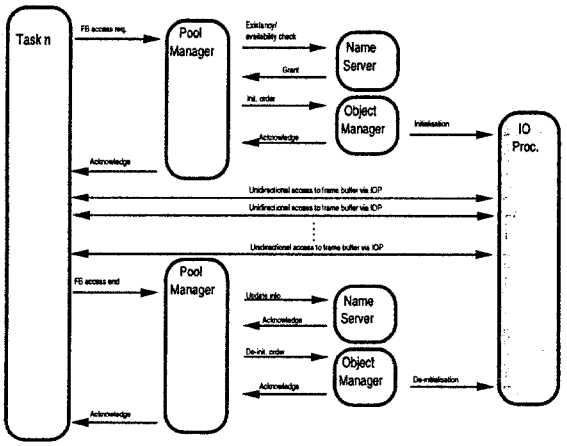

Figure 4. Dedicated frame buffer access protocol (DFBA)

In order to reduce this overhead a protocol called Dedicated Frame Buffer Access Protocol (DFBA) has been implemented whereby a process holds an IOP throughout its lifetime (Fig.4). Once allocated, the process can access the frame buffer via the dedicated IOP without asking or waiting for any grant from the pool manager. Accessing through only one process running in the same cluster is a constraint of this protocol. The DFBA protocol is useful when dealing with continuous stream of frames such as for the implementation of camera tracking algorithm[Rout95] whereas SFBA is useful for frame based processing like image processing.

On top of these two protocols, various kinds of video sequence access schemes have been realised. These schemes are only the programmers concern and they are not known to the end-users. As it can be seen in Figure 1, four IOPs are attached to the frame buffer: one to video in device, one to video out device and two to DSP clusters. For simplicity suppose that only each DSP runs one process and process numbers are identical to DSP numbers. The schemes are explained as follows, where the first two schemes use the SFBA Protocol and the last two use the DFBA Protocol.

1) Eight processes access to the frame buffer in turn, called *generic parallel execution mode* which is a typical use of the SFBA Protocol. The requests that can not be met immediately are queued to serve when the related IOP becomes available. There is a separate queue for each IOP to provide separate control of IOP connections.

2) Only four processes (one from each DSP board) access the frame buffer. By making use of the shared memory on the board, it is possible to run processes dealing with the same data (frame) on the same board which is called *generic shared memory execution mode* One processor from a board reads data into DRAM and signals the second one on the same board to inform the existance of the frame.

3) Two processes (one from each cluster) access the frame buffer without breaking the connection till the processes are completed, called *dedicated simple execution mode*. Thus a stream based connection is provided in which initialisation happens once so that both the high speed and the dbex buses could be optimally used for data transfer.

4) In the *dedicated simple execution mode* there is only one processor active on the board. However, by using shared memory on board as in protocol 2, the number of processors can be doubled, which is called *dedicated shared memory execution mode*.

6 Benchmarks and Evaluation

A number of experiments have been carried out to evaluate the performance of the ML-PVA with respect to some of the above explained properties and constraints. The main objective has been to investigate:

- the effects of using one IOP for a cluster (4 DSPs)
- system performance dependence on I/O intensive and compute intensive processes
- the effects of frame size
- the effects of the using shared DRAMs by two DSPs on the same board
- optimal DSP configuration with respect to a given application

Two groups of benchmark programs have been run to investigate the above mentioned objectives. The first group used an edge detection algorithm that takes a PAL frame of 576x720 pixels and produces a new one having only the edges. The number of the neighbourhood pixels taken into account makes the benchmark more or less compute intensive. Each task, once the computing is finished also writes back the computed frame into the frame buffer. The purpose of this group of benchmarks is show the effect of shared input/output processors (IOP) each IOP serving one cluster which has four DSPs. Also because of sharing the same DRAM, board based effects are important. This benchmark will also lead us to define a DSP loading scheme to get the best performance.

In general there are three different steps in a stream manipulation: The first step is getting the frame into a DSP's RAM, the second step is to process the frame and the third step is to put the processed frame back to the frame buffer, if required. The first and the third steps are called input/output and the second step is called compute cycle.

	Operation/frame		Operation/pixel			
Operation	Read	Write	+	-	*	Total
Edge Detection (Type A)	1	1	0	4	1	5
Edge Detection (Type B)	1	1	0	8	1	9
Edge Detection (Type C)	1	1	4	8	2	14

Table 1. Benchmark Operations.

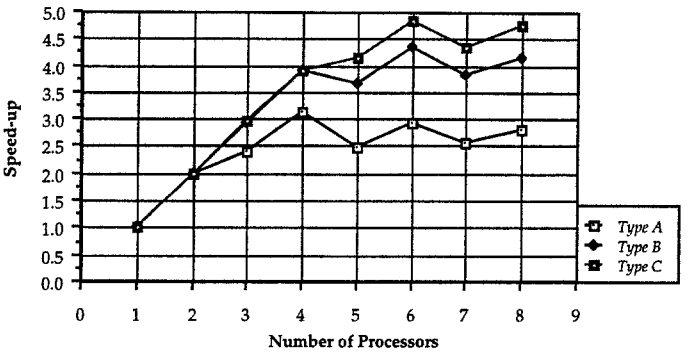

Figure 5. The best performance curves.

6.1 Benchmark Group 1

For the benchmarks carried out each of the tasks copies a PAL frame into its DRAM and writes it back after processing. These benchmark programs, A, B, and C execute different number of operations per pixel as shown in Table 1. After running a number of benchmarks for different schemes, the best performance curves are plotted in Figure 5 which show the following:

a) There is a trade-off between I/O and CPU intensive programs as expected. I/O dependency causes lower speed-up values as frame buffer accesses are done via a shared resource IOP. Speed-up for CPU intensive task is almost 8 when using 8 processors since I/O initialisation requests are processed immediately by the pool manager as I/O requests do not overlap each other because of relatively longer data processing time.

b) The practical highest speed-up value we obtained is 4.8 for 6 processors running concurrently for the benchmark program Type C which is the most compute intensive one. After this value, speed-up for processors decreases a bit since concurrent frame buffer access requests are queued and requests are met in turn.

c) As the number of frames used is even, configurations having odd number of processors give worse speed up values than expected since runs over odd number of processors, always one processor stays idle at the very last step. A closer look shows that the utilisation of processors decreases of the same reason as well. This problem can be overcome by decomposing data into the multiplies of the number of processors.

d) Best performance curves shows that processor allocation algorithm should have a priority scheme. Given the first four processors P0-P3 constitutes the first cluster and the next P4-P7 the second one, the frame allocation should be done by selecting processors from different clusters, then different boards within a cluster. In other words, there is a balanced way of processor allocation at cluster and board levels. An example for the order of selecting processor to get the best performance is P0, P4, P2, P6, P1, P5, P3, and P7.

6.2 Benchmark Group 2

The second group of benchmarks demonstrate the impact of data size and the impact of the SFBA and DFBA given above. Basically 100 are frames read with no computation. Thus, the maximum real data transfer rate can be measured. While running a benchmark over two clusters concurrently, 50 frames are read by each DSP at separate clusters. These benchmarks are run for five different frame sizes: 720*576, 360*288, 180*144, 90*72, and 45*36 pixels. Four different schemes are tested:

- Single processor with SFBA protocol (S-S)
- Double processors with SFBA protocol (D-S)
- Single processor with DFBA protocol (S-D)
- Double processors with DFBA protocol (D-D)

The actual measured values for reading 100 frames are given in Table 2 and the data size vs. reading duration curves are plotted in Figure 6.

Frame Size	720*576		360*288		180*144		90*72		45*36	
Procs.	Single	Double	Single	Double	Single	Double	Single	Double	Single	Double
SFBA	16.210	16.440	10.790	16.580	9.290	17.200	8.920	17.330	8.770	17.020
DFBA	7.880	3.940	1.980	0.990	0.495	0.248	0.126	0.064	0.032	0.017

Table 2. Reading times of 100 frames (in milliseconds).

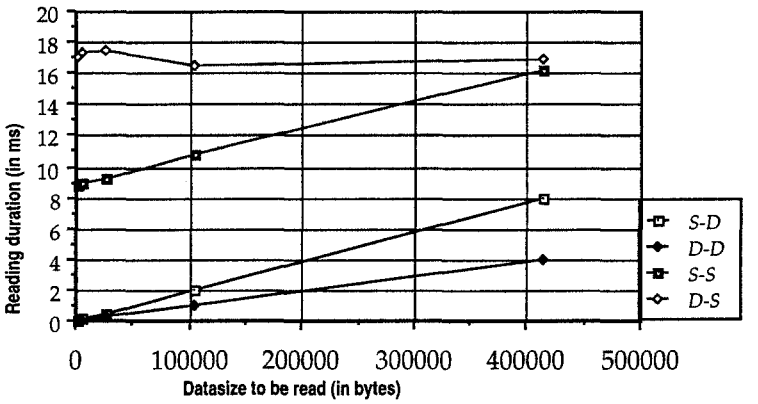

Figure 6. Performance comparison of protocols.

Concerning the four curves drawn in Figure 6 the following evaluations are done:

a) The curves S-D and D-D show linear change with respect to data size. Evaluating these two curves together shows that there is a speed up of 2 in data transfer rate when 2 clusters are being used. Thus, it is observed that DFBA Protocol provides linear transfer rate, irrespective of the data size, proportional to the number of clusters.

b) On the other hand, although SFBA protocol, is suitable for frame by frame processing, can not provide real-time performance requirements for continuous video processing as data transfer initialisation takes 87ms per frame.

c)Although the curve S-S is linear, it shows the initialisation overhead of 87ms. As there is no other active processor in the system the pool manager serves just for this particular DSP. Thus, there is no queuing impact. The curve D-S shows the queuing effect when there are two processors running concurrently to access the frame buffer under the control of a unique authority, the pool manager. As data transfer time is less than initialisation time, almost a constant value is observed which is around the twice of the initialisation time.

6.3 Analysis of the DFBA Protocol Performance

DFBA Protocol is capable of running real-time applications. In order to see the scaleability of the system, let the total execution time for a single frame be T_t, then

$$T_t = T_i + T_r + T_c + T_w$$

where T_i is data transfer initialisation and termination time, T_r is data read time, T_c is computation time, and T_w is data write time. Assume that α is the data transfer rate of the bus (in Mbytes/sec), d_i is the size of input data (in bytes), d_o is the size of output data, and β is the number of pixels computed per second. If the computation is data independent then,

$$T_r = d_i / \alpha, \quad T_w = d_o / \alpha, \quad T_c = d_i / \beta$$

For the same size of frame read and written d_i becomes equal to d_o and, therefore, $T_r = T_w$.

As the DFBA protocol initialises data transfer once, there is practically no overhead introduced by this initialisation process, therefore T_i=0. If N_c is the number of clusters then N_c frames are transmitted concurrently. Therefore the average time for processing each frame, so long as a sequence of frames are fed to all clusters, becomes

$$\overline{T_t} = (2 T_r + T_c) / N_c$$

or in terms of the data size

$$\overline{T_t} = d_i (2 / \alpha + 1 / \beta) / N_c \qquad (1)$$

To satisfy the real-time requirements, $\overline{T_t}$ should be less than 40 ms for a PAL frame. Thus, from Equation 1 one can derive

$$N_c \geq d_i (2 / \alpha + 1 / \beta) / 0.04 \qquad (2)$$

For ML-PVA Machine α is 5 Mbytes/sec (Figure 6) restricted by the cluster bandwidth eventhough the IOP rate is 20 Mbytes/sec. For a simple frame inversion algorithm β is 1/300 10^9 pixels per second. For a sequence of 360*288 pixels Equation 2 yields

$$N_c \geq (360*288) (2/5242880+300/10^9) / 0.04$$
$$N_c \geq 1.766$$

Therefore

$$N_c = 2$$

In order to run the same application over full sized PAL frame (576*720), we would need higher data transmission rates to reduce the data transmission time. For more complex operations more powerful processors like C80 would be required. For example for a machine with α=20 Mbytes/sec and β=1/60 10^9 pixels/second

$$N_c \geq (720*576) (2/20971520+60/10^9) / 0.04$$
$$N_c \geq 1.611$$

$$N_c = 2$$

7 Conclusion

A multi-processor machine has been built to meet the computational requirements of video processing tasks for Virtual Studios and other similar applications. The architecture has been applied successfully to a camera tracking algorithm in real-time. The accelerator is based on a pool of DSP processors tightly-coupled with an intelligent frame buffer system over concurrent buffered I/O channels. It has an OSF/Motif windows based user interface running on a host graphics workstation which makes the application acceleration completely transparent to the end-user.

In this paper, we focused on the frame buffer access protocols. Two different protocols, SFBA and DFBA are described. The DFBA Protocol is essential for real-time processing as it is a stream based protocol with one initialisation and one termination step. An analysis has been presented for the DFBA Protocol to explain the benchmarking results. The interdependencies of frame size, frame rate, channel transfer rate and number of clusters have been demonstrated. This shows that the ML-PVA architecture is a scaleable parallel system under the DFBA protocol. The disadvantage of this protocol is the dedication of an IOP to a single cluster.

The SFBA protocol is important as it provides the ability of processing a video sequence of independent frames. Although a speed-up of 4.8 has been achieved for 6 processors, the SFBA

protocol suffers from the overhead of initialisation and termination. Sharing an IOP among the processors of a cluster is an advantage of the SFBA protocol.

The current architecture provides concurrent access to the frame buffer including the same video sequence on the frame buffer. The bandwidth of high speed bus is shared among the attached clusters via IOPs equipped with buffers. This structure provides a good way of accessing the frame buffer. However, we observe that having a single authority in resource management like frame buffer management introduces severe bottlenecks if resource requests are come from different processing units. A stream based protocol overcome this difficulty.

8 References

[Blon96] Blonde L., Buck M., Galli R., Niem W., Paker Y., Schmidt W., Thomas G., A Virtual Studio for Live Broadcasting: The MonaLisa Project, IEEE Multimedia, Summer 1996.

[DVS93] Image Storage and Processing System ISP500 User Manual, DVS GmbH, Hannover, Germany, 1993.

[Feit95] Feitelson D.G., Rudolph L. , Parallel Job Scheduling: Issues and Approaches, Job Scheduling Strategies for Parallel Processing, IPPS'95 Proceedings, Springer, 1995.

[Gabb90] Gabber E., VMMP: A Practical Tool for the Development of Portable and Efficient Programs for Multiprocessors, IEEE Trans. on Par. and Dist. Systems, Vol.1, No.3, July 1991.

[LeFl95] Le Floch P., Sahiner A.V.,PakerY., Visual Tools for Parallel Server Handling, Proceedings of the European Workshop on Combined Real and Synthetic Image Processing for Broadcast and Video Production, Paker Y. and Wilbur S., (Eds.), Springer, 1995.

[Lucc87] Lucco S.E., Parallel Programming in a Virtual Object Space, Sigplan Notices, Vol.22, No.12, December 1987.

[Rout95] Routsis D., Le Floch P., Sahiner A.V., Real-Time Camera Tracking Server on the ELSET Accelerator, Proceedings of the European Workshop on Combined Real and Synthetic Image Processing for Broadcast and Video Production, Paker Y. and Wilbur S, (Eds.), Springer, 1995.

[Sahi91] Sahiner A.V., A Computation Model for Parallelism: Self-Adapting Parallel Servers, Ph.D. Thesis, The Polytechnic of Central London, 1991.

[Sahi95] Sahiner A.V., Le Floch P., Paker Y., A Parallel Accelerator for Using Synthetic Images in TV and Video Production, Proceedings of the European Workshop on Combined Real and Synthetic Image Processing for Broadcast and Video Production, Paker Y. and Wilbur S.(Eds.), Springer, 1995.

[Sing94] Singh J.P., Gupta A., Levoy M., Parallel Visualisation Algorithms: Performance and Architectural Implications, IEEE Computer, July 1994.

[Stei94] Steinmetz R., Data Compression in Multimedia Computing: Standards and Systems, ACM Journal of Multimedia Systems, March 1994.

[Whit94] Whitman S., Hansen D.C., Crockett T.W., Recent Developments in Parallel Rendering, IEEE Computer Graphics and Applications, July 1994.