Performances of Multiprocessor Multidisk Architectures for Continuous Media Storage

Benoit A. Gennart, Vincent Messerli, Roger D. Hersch

Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

ABSTRACT

Multimedia interfaces increase the need for large image databases, capable of storing and reading streams of data with strict synchronicity and isochronicity requirements. In order to fulfill these requirements, we consider a parallel image server architecture which relies on arrays of intelligent disk nodes, each disk node being composed of one processor and one or more disks. This contribution analyzes through bottleneck performance evaluation and simulation the behavior of two multi-processor multi-disk architectures : a point-to-point architecture and a shared-bus architecture similar to current multiprocessor workstation architectures. We compare the two architectures on the basis of two multimedia algorithms : the compute-bound frame resizing by resampling and the data-bound disk-to-client stream transfer. The results suggest that the shared bus is a potential bottleneck despite its very high hardware throughput (400Mbytes/s) and that an architecture with addressable local memories located closely to their respective processors could partially remove this bottleneck. The point-to-point architecture is scalable and able to sustain high throughputs for simultaneous compute-bound and data-bound operations.

Keywords : image computing for image, video and multimedia database ; multiprocessor multidisk storage architectures ; architecture throughput and jitter.

1 Introduction

In the fields of scientific modeling, medical imaging, biology, civil engineering, cartography and graphic arts, there is an urgent need for huge storage capacities, fast access and real-time interactive visualization of pixmap images and multimedia streams. Interactive real-time visualization of full color pixmap image data or of high-quality video streams requires at the application level a throughput of 1 to 100 MBytes/s. With the availability of high-speed networks such as FDDI, fast Ethernet and ATM broadband, high-performance high-capacity image and media servers must provide client stations located on local or public networks with a set of adequate services for immediate access to image, video and sound streams. Parallel mass storage devices are required in order to access and manipulate multimedia data at high speed.

A multimedia server does not only need to access requested data streams at the required throughput. It also needs to be able to offer processing services such as conversion from one video format to another, from compressed to uncompressed and from uncompressed to compressed stream formats. Existing video streams need to be converted from one resolution to another, and from one frame rate to another to suit the client's requirements. Furthermore, a multimedia server also needs to provide continuous media editing facilities for creating superimpositions, titles, inserts and special effects, such as transitions between clips.

In Proc. Storage and Retrieval for Still Image and Video Databases IV, 1-2 February 1996, San Jose, USA, SPIE Proc. 2670, 286-299

The RAID-II architecture approach[1] offers high-bandwidth disk arrays hooked directly onto high-speed networks. Although the RAID-II approach offers very high bandwidth, it requires an expensive high-speed network and its interface. The present contribution considers architectures where storage devices are closely coupled with processing power in order to avoid unnecessary and costly transfers of data across high-speed networks. In the authors' mind, an image and multimedia server should only transfer those pieces of data which are of immediate use to the clients. All preprocessing operations such as frame resizing, stream compression, format conversion, overlay of streams, and windowing operations should be executed by the server. The server should also be able, by reducing the frame rate, resizing the frames to a lower size and increasing the compression factor, to adapt the video stream rate to the network throughput.

We consider two different multiprocessor-multidisk (MPMD) architectures as multimedia server architectures. We are interested both in the throughput of the architecture and in the delay jitters introduced by hardware devices such as disks and shared buses.

In order to evaluate the architecture's capabilities and to compare the point to point and the shared bus architectures, we define two operations to be executed in parallel by the multiprocessor-multidisk multimedia server running on these architectures. These operations are representative for the services which have to be offered by multimedia servers. The first operation consists of continuous accesses to video streams segmented in strands striped over the set of available disks. The second operation consists of accessing in parallel an uncompressed video stream striped across the disks, resizing it by applying to its frames a resampling operation and compressing it by dedicated compression hardware.

The two analyzed multiprocessor-multidisk systems are a point-to-point-communication architecture based on the GigaView concept,[2] and a shared-bus architecture similar to high-end general-purpose workstation cluster architectures. Our approach is to evaluate through experiments on single-processor single-disk workstations individual component performance (e. g. memory initialization throughput, memory read, write and copy throughputs, disk access throughput and latency). We then use the performance parameters to establish a qualitative performance model based on the analysis of potential bottlenecks. Bottlenecks are determined by analyzing the time spent by each component to produce one byte at the output of the multiprocessor multidisk architecture. The slowest component in the architecture is the bottleneck. We give a qualitative prediction of the performances we obtain for the considered operations and validate these predictions by simulating the modeled MPMD architectures.

The result of the analysis suggests that the two considered multiprocessor multidisk architectures both offer a very high throughput for both stream access and stream data processing. It also shows that, in theory, a shared bus architecture can provide a high throughput since it is hard to saturate a 400MBytes/s backplane bus. However, the caching mechanisms that improve the computing speed of such architectures have an adverse effect on the performance of data-bound applications, by requiring the data to travel many times over the bus. We suggest that an architecture where the data flow is under the control of the application developer and where addressable distributed local memories are available to run processing operations locally offers a better performance than a single shared memory architecture where the data flow is controlled by cache coherency schemes.

Section 2 lists related research and section 3 specifies the modeled architectures : the intelligent disk-node, the shared-bus architecture, the point-to-point architecture. Section 4 specifies the modeled operations. Section 5 presents the results obtained from estimations and simulations.

2 Related work

Multimedia storage architectures. Besides the RAID architectures presented in the introduction, several authors propose specific solutions for storing multimedia streams. Reddy[3] discusses disk scheduling strategies in a multimedia I/O system. His analysis concerns the way multiple-streams should be allocated on a single disk. Lougher presents a continuous multimedia storage server (CMSS[4]) consisting of a disk array connected to an interface processor. Tobagi[5] presents StarWorks, a video applications server and discusses both the required disk

array bandwidth and the memory size in the case of simultaneous access to multiple video streams. Vin and Rangan[6] describe a multiuser HDTV storage server, but do not consider striping single streams across multiple disks. Druschel discusses the use of workstations for multimedia access and processing.[7] He points out the inadequacy of cache-based workstation architectures for multimedia applications and discusses ways of reducing the number of memory copies in multimedia processing applications. He predicted that shared bus throughput will not increase at the same rate as processing power and that therefore the shared bus will remain a major bottleneck in multimedia applications. However, the recent evolution of bus transfer rates shows that the raw bus throughput has increased nearly as fast as processing power (for example by a factor of 5 between a Sparc 2 and a Sparc 20 workstation). The present contribution shows that the shared bus is not a bottleneck anymore for single processor workstations, but that it tends to remain a bottleneck for certain classes of operations in the case of multiprocessor multidisk workstations.

Multiprocessor-multidisk point-to-point architectures. The first attempt to combine processors and disks in a point to point architecture was presented by Wilkes in 1991 in order to speed up disk input-output operations[8] . The authors have implemented a multiprocessor-multidisk image server, called the GigaView. The GigaView consists of a server interface processor based on a T800 transputer connected through transputer links (1.6MBytes/s) to a number of intelligent disk-nodes.[2] Each intelligent disk-node consists of a T800 transputer connected through a SCSI-2 to a single magnetic disk. A 4-disk GigaView connected through a SCSI-2 standard interface to a host computer (Macintosh, Unix Workstation) provides a throughput of up to 5MBytes/s at the application level, and the ability to browse through images and maps of arbitrary size at the rate of three to four 512-by-512 3-byte-pixel visualization windows per second. At 5MBytes/s, the standard GigaView architecture reaches its peak performance, because its interface processor becomes saturated. Higher throughputs require more server interface processors. Within the limits of its component performances, the GigaView architecture demonstrates the benefits of associating disks and processors, and of having full control of the data flow inside the architecture. The authors have analyzed the behavior of the GigaView architecture for simultaneous access to multiple continuous streams.[9]

3 Architecture models

The two analyzed multimedia server architectures are formed by associating disks and processors so as to form an array of intelligent disk nodes capable of executing in parallel local preprocessing operations. In the point to point architecture, disk nodes are connected to a server interface processor by communication links. In the shared bus architecture, disk nodes communicate through the shared bus.

Section 3.1 describes the modeling methodology. Section 3.2 describes the intelligent disk node. Section 3.3 details the elements of a point-to-point based distributed memory architecture. Section 3.4 describes the model of the shared bus architecture made up of a set of intelligent disk-nodes communicating through the shared bus. Section 3.5 analyzes in detail the throughput performance of the backplane bus.

3.1 Modeling

The modeling methodology in this contribution is based on (a) measuring the performance of *elementary* operations ; (b) defining the *resources* required to execute elementary operations ; (c) combining elementary operations and resources to model *composite* operations of complex systems.

Elementary operations such as memory to memory data access, disk data access, or resampling are measured experimentally on existing single-processor single-disk workstations (Sparc20, DEC 3400), and relevant parameters such as throughput and latency are evaluated. Simulation models for elementary operations (e. g. the time to transfer a data packet from disk to shared memory) are created using the measured performance parameters.

A system is modeled as a set of components (bus, disks, processors, links). Components perform elementary

operations sequentially. Systems perform composite operations concurrently. Composite operations (e. g. a multi-processor multi-disk architecture accessing a multimedia stream) are specified as a series of elementary component operations. Simulated systems are prospective systems such as a 4-disk-node 16-disk point-to-point architecture or a 4-processor 8-disk shared-bus architecture. The simulator derives the system performance for specific stimuli.

System dependent software overheads have not been included into the model. On Sparc Unix systems, for example, reading a file from a disk requires for each piece of data one DMA transfer from disk to kernel space and an additional memory copy operation from the kernel to the application memory space. Mechanisms are being developed to avoid this additional memory copy operation.[10]

3.2 Intelligent disk node

An intelligent disk-node optimized for multimedia applications consists of a processor connected to memory, to a compression-chip and to a SCSI-2 controller interfacing disk storage devices through a SCSI-2 channel (Fig. 1). As is the case in many workstations, the memory is capable of sustaining the peak backplane bus throughput rate. We therefore assume that the memory throughput is limited by the bus throughput. The SCSI-controller enables data to be transferred directly from secondary storage to main memory by direct memory access (DMA). The processor performs software resampling operations. Compression, which in the case of MPEG[11] is a compute-intensive task, requires dedicated hardware such as for example the C-Cube MPEG processor.

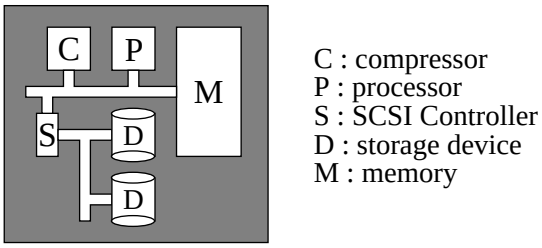

Figure 1 : Intelligent disk node

From an architectural standpoint, an intelligent disk node behaves like a low-end workstation. The model for the intelligent disk-node is hierarchical. A disk-node consists of a processor (CPU), a compression circuit, a backplane bus and a composite I/O component. The I/O component consists of a controller, a SCSI-2 bus, and a number of magnetic disks. Processors, compression circuits, backplane, SCSI-controllers, SCSI-busses, and disks act as resources carrying out elementary operations. The performance of each of these components is measured experimentally. The following paragraphs list the performances of elementary operations.

Disk access. For our simulations, we consider disks with a 10msec average seek-time, and a 4MByte/sec throughput. Current disks support a throughput of a few MBytes/s. Despite raw-throughput increases, the application-level throughput is unlikely to increase much, as the disk latency will remain of the order of a few msecs. We make no assumptions as to the position of streams on the disks, and count an average seek-time for each access. This simplification has little effect on the results, as the manipulated packets of data are large (more than 50KBytes). Detailed analysis of stream allocation on disks can be found in other contributions.[6,5,3]

Disk modeling. The disk delay is modeled as the sum of a latency and a data transfer delay. The data transfer is linear in the size of the data transferred. By assuming equiprobable cylinder positions for strands, the disk latency is modeled as a random variable of the form $S(t) = \frac{2}{3*mst}(1 - \frac{1}{3*mst}t)$, where mst is the mean seek-time. For this random variable, the standard deviation is $\frac{mst}{\sqrt{2}}$. For a 10ms seek-time, the standard deviation is 7.07ms.

SCSI-bus. We consider a fast-wide SCSI-2 bus, in synchronous mode. Experiments made by connecting several disks to a single SCSI-2 bus in synchronous mode show that 3 IBM disks achieve a maximum throughput of 9.2MBytes/s, for a raw SCSI-2 bus throughput of 10MBytes/s. The fast-wide SCSI-bus in synchronous mode has a raw throughput of 20MBytes/s. We therefore assume the maximum throughput sustained by the SCSI-bus to be 18.4MBytes/s.

Stream resampling. The stream resampling procedure uses backward mapping and nearest neighbor resampling. The delay for running this resampling code on a 4MByte output strand is 0.57s on a Sparc20/502, resulting in a throughput of 7.35MBytes/s. The same code runs in 0.38sec on a DEC alpha 3400, resulting in a throughput of 10MBytes/s.

Processors. 100MIPS processors are common, and 400MIPS processors exist. However, algorithms that require less than 100 instructions per bytes are rare, and therefore processors that can process data faster than 1MByte/sec are rare. The nearest-neighbor-resampling algorithm applied to 3-byte-pixel images achieves a rate of 7.35MBytes/s on a Sparc20, and 10MBytes/s on a Dec3400. Decompression algorithms such as MPEG decompression achieve surprisingly good rates, because their throughput depends on the size of the input data, which is small. Run-length decompression algorithms also perform very well. MPEG compression algorithms perform very slowly, as their input data is large and the MPEG compression scheme is strongly asymmetric. A dedicated compression circuit is required if acceptable performance is to be achieved.

MPEG compression. For modeling MPEG compression, we rely on performance figures announced for the C-Cube CL-4000 chip. The CL-4000 is a single-chip encoder circuit that can compress a 352x240 stream at the rate of 30 frames/s, resulting in an input throughput of 7.6MBytes/s. We assume that uncompressed stream frames are loaded into the compression circuit's input buffer by direct memory access.

3.3 Point-to-point architecture

The point-to-point architecture consists of a server interface processor connected through communication links to an array of intelligent disk nodes (Figure 2). The communication links are modeled as T9000-transputer links, rated at 100Mbits/s. Experiments show that the T9000 links can achieve throughputs of up to 8MBytes/s. The communication link controllers transfer data between memory and communication links by direct memory accesses, allowing the processor to perform other tasks while communication takes place. A number of architectures based on transputer links have been proposed and built, and achieve excellent performance and load balancing.[12]

The transputer links support the concept of virtual channels, allowing any number of virtual channels to be allocated on a physical link. A single Inmos C-104 crossbar switch is capable of connecting up to 32 bi-directional links and provides the means to transfer packets between any two connected transputers without noticeable delay. Such serial communication links limit the single-wire connection bandwidth, but enable crossbar chips of moderate complexity and with excellent throughput to be designed. The nominal raw throughput of the C-104 crossbar chip (32 * 100MBits/sec = 400MBytes/s) is as high as the nominal throughput of a Sparc-20 backplane bus. In the architecture shown in figure 2, the four server interface communication links are a shared resource and may limit the global throughput. Adding server interface processors enables this potential bottleneck to be removed and renders the architecture completely scalable.

3.4 Shared bus architecture

The shared-bus architecture consists of a number of intelligent disk-nodes all sharing the same single backplane bus (Fig. 3). This multiprocessor workstation cluster architecture is modeled according to the Sparc20. The actual architecture of a SPARC 20 I/O board, as shown in figure 3 is more complex than the intelligent disk-node model of figure 1 : (a) there is an additional I/O bus , called the Sbus ; (b) the compressor board and the SCSI devices are hooked onto the Sbus rather than directly onto the shared bus. However, the SBus offers sufficient bandwidth

(49MBytes/s read throughput and 55MBytes/s write throughput[13]) to support the combined throughputs of the disks (at most 4MBytes/s times the number of disks connected to the SCSI bus) and of the compression circuit (roughly 8MBytes/s). In our model, we therefore ignore the details of the SBus behavior and assume that the SCSI-2 controller and the compression circuit are directly connected onto the shared bus.

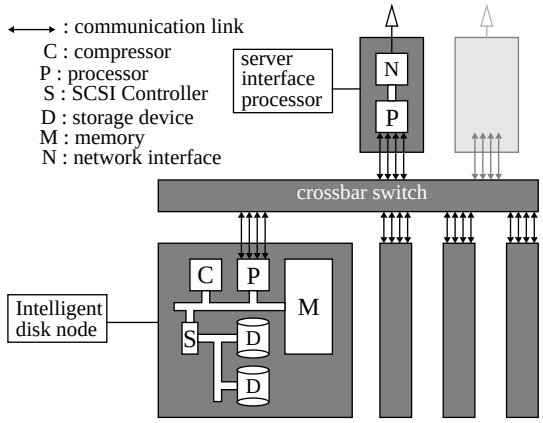

Figure 2 : Point to point architecture

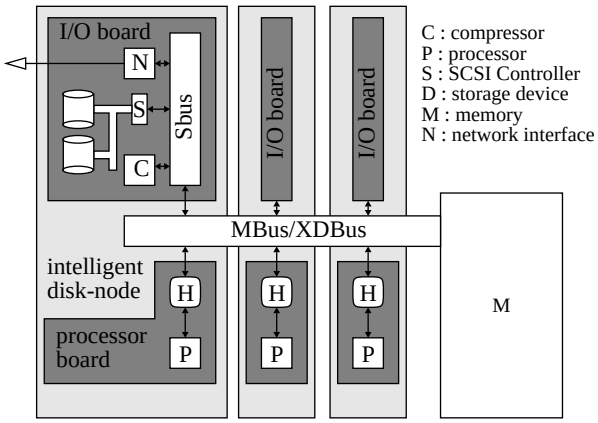

Figure 3 : Shared bus architecture

Section 3.5 gives a detailed analysis of the backplane bus performance and of the cache behavior, as these are critical in the shared bus architecture performance.

3.5 Backplane-bus throughput

The detailed bus performance measurements were performed on a 2-processor SPARC20 workstation. The information on the SPARC20 architecture is found in two SUN publications.[13,14] The Sparc20 backplane bus is an 8-byte wide synchronous bus called MBUS, running with a 50Mhz clock. Its raw throughput is therefore 400MBytes/s. We are interested in measuring the throughput for simple operations such as the UNIX memset and memcopy operations, and in establishing the relationship between the memset operation throughput and the raw bus throughput. The biggest difficulty in this study is to understand how many times the data actually travels through the bus for a given operation, taking into account the effect of the cache. Intuitively, the memset and memcopy operations only require one and two bus transfers respectively. In fact, due to cache interaction, the memset operation operation requires *two* bus transfers and the memcopy operation requires three transfers. When setting at a specific address, the memset operation at first causes a cache miss, resulting in the entire cache line (32 bytes in the case of a Sparc20) to be loaded into the cache. The line is immediately overwritten by the memset operation, and at the next cache miss, the data will be transferred back to memory resulting in two transfers over the bus for a single memset operation.

For the experiments, we used the Solaris *libfast.a* library, which guarantees that data transfers are carried out using all 8 bytes of the bus and we aligned the data to 32 bytes boundaries, so as to ensure that 32 bytes packets fit exactly in one cache line.

Read operation. To measure a single read transfer on the bus, we specified a test program that repeatedly *reads* 4 bytes in every cache line (program 1, left). As a result of each instruction, a 32-byte data packet is transferred from memory to processor cache. As the data is clean, it never gets transferred from the processor cache back to memory. The memory read or cache fill throughput of the test program is 105.3MBytes/s, or an average of 300nsec (15 clock cycles at 50Mhz) per cache line.

Considering that the instruction costs 1 clock cycle, we can derive that a cache miss costs 14 cycles. This

is compatible with the MBUS specification which guarantees a 90MByte/sec memory read throughput with a 40Mhz clock. The MBUS blocks the bus during read accesses. The newer XDBus (SparcCenter 2000) with a packet-based protocol does not block the bus during read accesses.

```
LD [%io],%lo
LD [%io+32],%lo
LD [%io+64],%lo
LD [%io+96],%lo
LD [%io+128],%lo
...
```

```
ST [%io],%lo
ST [%io+32],%lo
ST [%io+64],%lo
ST [%io+96],%lo
ST [%io+128],%lo
...
```

Program 1 : Test programs (left : read bus-transfer ; right : read-write bus-transfer)

Write operation. To measure the write throughput, we specified a test program that repeatedly *writes* 4 bytes in every successive cache line (program 1, right). As a result of each write instruction, a 32-byte data packet is transferred from the memory to the cache. As the cache data is now dirty, it gets transferred from the cache back to memory at the next cache miss. The throughput obtained in this experiment is 63MBytes/s, or 500nsec per 32Byte-cache-line. Considering that 300nsec are taken by the read operation to fill the cache line, this leaves 200nsec for the pipelined write operation, resulting in a pure cache to memory write throughput of 160MBytes/s. The announced MBUS write performance is of 200MBytes/s. Our experiment exhibits a performance which is closed to the announced peak performance. This test program sustains the same throughput as the memset operation.

Memory copy operation. The third experiment measures the throughput for two memory read and one memory write accesses, corresponding to the memcopy operation. One memory read access fills the cache line with the source data, the second read access fills the cache line with the destination data, and the write access transfers the modified cache line back to its destination location in memory. The expected time for a 32Byte packet is 800nsec, with 2*300nsec for the two read operations and 200nsec for the write operation, resulting in a 40MByte/sec throughput. The actual experiment results in a 39.4MByte/sec throughput, or almost the predicted 40MBytes/s.

Disk access. The fourth experiment measures the effect of disk accesses on the bus throughput. In the experiment, there are two processes running on two separate processors. The first process applies a memcopy operation to a 500MBytes data set. The second process reads continuously data from the disk using the raw device driver. By comparing the delay of the memcopy operation in the third experiment (memcopy alone) and in this experiment (memcopy combined with disk read operation), we derive the time required by the bus to perform I/O operations. The experiment is performed with disk-read requests divided either in 50KBytes- or in 500KBytes-packets. Table 1 shows the results of this experiment.

packet size	500MB memcopy	memcopy + disk read	disk read time	disk read size	nbr. of cycles per 8-byte word
50KB	12.82s	14.37s	1.55s	57.52MB	10
500KB	12.82s	13.84s	1.02s	55.66MB	7

Table 1 : Bus disk-read utilization

For 50KByte-packets, the time used by the disk-read operation is 1.55s, and the amount of data transferred is 57.52MBytes. This means that, on the average, the bus takes 200nsec (10 cycles) to transfer an 8-Byte word. The maximum bus throughput for a read operation using the raw driver is therefore 38.9MBytes/sec.

Summary. Let's summarize how throughput decreases as we increase the complexity of operations. The raw bus throughput is 400MBytes/s. Yet pure read, respectively pure write transfer throughput is 105, respectively 160 MBytes/s for the MBUS. A memset operation results in two transfers at a throughput of 63MBytes/s, and

a memcopy results in three transfers at a throughput of 40MBytes/s. Roughly, the memcopy operation is done at 1/10th of the raw bus throughput. For reference, we ran tests of the memset and memcopy operations on the DEC alpha 3400 workstation, resulting in throughputs of 84MBytes/s and 42MBytes/s respectively.

Table 2 summarizes the throughput figures that will be used in the simulations. These figures are a summary of all the performance measurements. We consider the best performance for each operation. Concerning the bus and memory access performance, a write-to-memory operation requires two bus transfers (equivalent to the memset operation), and a read from memory operation requires a single bus transfer.

operation	input (MB/s)	output (MB/s)
Disk.Read		4MB/s+10ms lat.
I/O.WriteToMemory		38.9MB/s
Processor.WriteToMemory		60MB/s
Processor.ReadFromMemory		100MB/s
Cpu.MPEGDecompression	0.33MB/s	5MB/s
Cpu.Resample		7.5MB/s
MPEGCompressor.Recompress	7.5MB/s	0.5MB/s
Link.Transfer		8MB/s
ScsiBus.Transfer		18.4MB/s

Table 2 : Throughput summary

4 Multimedia operation modeling

As mentioned in the introduction, we consider two multimedia operations which are representative for the operations of a parallel multimedia server : (1) the data-bound multimedia-stream reading operation ; (2) the compute-bound *uncompressed* multimedia-stream resizing-by-resampling and compression operation. We prefer the expression data-bound to I/O-bound, as it is not certain that in data transfers, the I/O is always the slowest part. To achieve parallelism, multimedia streams are divided into *strands*, allocated on the various disk nodes in the architecture. Load balancing is achieved by placing consecutive strands on consecutive disk-nodes. Previous research has shown that the overriding concern in designing multimedia architectures is in achieving throughput. A further concern consists in evaluating the delay jitter induced by hardware, such as the random disk access time and the contention over shared resources (shared bus, shared communication links). We are however mainly interested in demonstrating the architecture's throughput capabilities. We have suggested that if the throughput is large enough, respectively the utilization rate of the architecture's components is low enough, isochronicity and synchronicity requirements may be fulfilled by appropriate buffer allocation strategies and possibly by constraining the frame rates to some integer multiples of a basic frame rate .[9]

Strands are sufficiently large (at least 50KBytes) so as to ensure that data transfers take at least as long as disk seek-times. On the other hand, strands are not too large to allow pipelining within the architecture and with the network interface.

Intelligent disk-node operations on strands. A disk-node is capable of accessing a strand, and resampling a strand. Accessing a strand requires generating appropriate SCSI requests for disks, sending the requests to the I/O SCSI-controller, accessing the disks and transferring the data back to memory by direct memory accesses. In the case of multimedia operations, the data is transferred immediately to the right place in memory, without intervention from the disk-node processor.

Reading and resampling a strand consists (a) of applying the previously described operations for reading the strand, which consist essentially in transferring the strand from disks to memory (1 bus transfer) ; (b) resampling the strand (CPU, 2 bus transfers) ; (c) compressing the resampled strand (compressor hardware, 2 bus transfers).

The resampling operation requires therefore stream parts to be read once from memory to cache, to be processed within the cache and to be written once from cache to memory. Each cache-miss requires 25 bus cycles (10 to write the cache line back to memory, and 15 to read the new line from memory to the cache) or 500nsec. For the compression operation, we assume that the compression circuit's input buffer is loaded by direct memory accesses.

Point-to-point architecture operations. The parallel read operation running on the point-to-point architecture requires dividing a stream access request into multiple strand access requests and sending the strand access requests to the appropriate disk-nodes. Each disk-node reads its requested strands from the disks and transfers them back to the server interface processor which merges the strand requests to form the requested stream. The parallel resize-by-resampling operation on the point-to-point architecture requires dividing a stream request into multiple strand requests and sending the strand requests to the appropriate disk nodes. Each disk-node accesses its strands from its disk, resamples them independently and transfers the resampled strands to the server interface processor.

Shared bus operations. The continuous stream read operation on a shared bus architecture requires broadcasting a stream access request to all disk nodes in the architecture. All disk nodes then generate in parallel strand access requests and access the strands from their disks. Accessing a strand consists in transferring the strand from disk storage to its appropriate location in shared memory.

The stream *read and resample operation* on the shared bus architecture requires broadcasting a stream access and resampling request to all disk nodes in the architecture. All disk nodes generate in parallel strand access requests, read the strands from the disks and resample them.

5 Simulation results

In this section, we run simulations using the models of section 3 and 4, and the performance parameters gathered in table 2. For the two operations under consideration (stream access, and stream *read and resampling*) we plot the global throughput and component utilization as a function of the number of disks and disk nodes in the architecture. For all experiments, the number of disks per disk-node is one.

5.1 Reading streams in parallel from the disks

In this experiment, a stream is requested from the server. Whether the stream data is compressed or uncompressed does not affect the experiment since reading an uncompressed stream at a low frame rate is equivalent to reading a compressed stream at high frame rate. All strands are requested at the beginning of the experiment and the experiment is finished when the last strand has been processed. The *throughput* is defined as the total stream size divided by the maximum time. Reading the stream requires accessing the strands in parallel from secondary storage, and assembling them in shared memory. The requested stream consists of 256 352-by-240 2-byte-pixel uncompressed frames. Each strand consists of a single 352-by-240 2-byte-pixel uncompressed frame.

The numbers selected for the frame size, the number of frames per stream and the number of frames per strand are chosen so as to optimize the point-to-point architecture's internal pipelining. This pipelining occurs at the strand-level : streams are divided into strands, and strands stored on the same disk-node are processed in pipeline : disk-node accesses are followed by transfers to the server interface processor. The largest detrimental effect on the delay comes from ineffective data pipelining. If strands are too large the total delay becomes too large as only a few strands are processed by each node. If the strand becomes too small, disk access time increases due to an increased number of seek-times.

Figure 4 summarizes the results of running the throughput experiment for architectures consisting of 1 to 16 intelligent disk-nodes. The figure contains throughput and utilizations numbers. The utilization of a component

is defined as the ratio of a component's busy time vs. the total simulated time. The left-hand scale displays throughput numbers (MBytes/s) and the right-hand scale average utilizations (from 0 to 1). The average utilization is plotted for the disks (D-line), the SCSI busses (S-line), the backplane busses (B-line), the disk-node processors (P-line), the communication links (L-Line) and the server interface processor (I-Line).

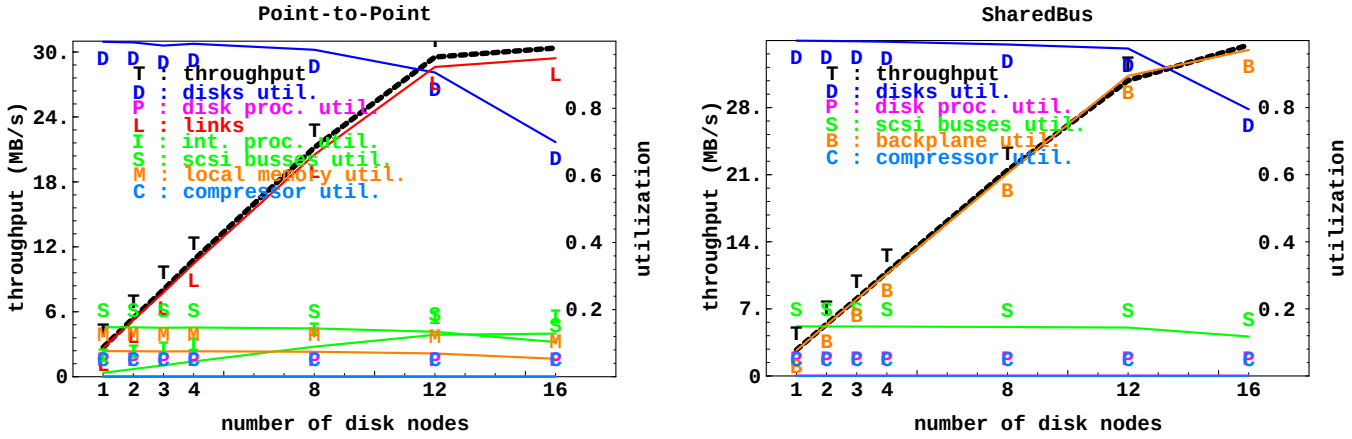

Figure 4 : Reading streams : throughput and utilization

Let us estimate the sustainable throughputs for both architectures. The average delay for a 40KByte-disk-request is 20ms (10ms seek-time and 10ms data transfer time at 4MBytes/s), or a 2MBytes/s throughput. Larger disk request size would increase the latency of the architecture, and the ability of the server to start pipelining the stream onto the network. Smaller request sizes would decrease the disk throughput. For a 16-disk point-to-point architecture, the disks are capable of sustaining at most 16 times 2MBytes/s or 32MBytes/s. The four server interface processor communication links become saturated at 32MBytes/s. In the case of the shared-bus architecture, the backplane bus becomes saturated at 38.9MBytes/s when writing data from the disks to shared memory (see Table 2). This simple analysis shows that the point-to-point architecture will saturate at around 32MBytes/s throughput, whereas the shared-bus architecture should be able to sustain throughputs of at most 38.9MBytes/s.

Figure 4 confirms these results. The point-to-point architecture reaches its maximum throughput when all links become saturated, at approximatively 30MB/s (roughly four times each link throughput). The interface processor is far from being saturated. The throughput of the point to point architecture could be improved by increasing the number of links connected to the server interface processor or by increasing the number of server interface processors. The shared-bus architecture reaches saturation for architectures with more than 12 nodes, and sustains a maximum throughput of 35MBytes/s.

However, the simulated operation is favorable to the shared bus architecture. In the simulations, data has only reached the architecture's shared memory. Attempting to transfer the data to the network will require at least one additional bus transfer, resulting in the architecture being saturated at a much lower throughput. If the network protocol requires the data to be transferred once across the intelligent disk-node processor, the throughput will drop very low. Considering a 35MByte/sec transfer from disk to memory, a 40MByte/sec memory-to-memory round-trip through the disk-node processor for protocol purposes (creation of data packets with headers) and a 100MByte/sec transfer from the memory to the network, the shared bus throughput drops to $\frac{1}{\frac{1}{35}+\frac{1}{40}+\frac{1}{100}}$ = 15.73MBytes/s. In the case of the point-to-point architecture however, once the stream data reaches the server interface processor, the server interface processor can take care of network protocol requirements without interfering with stream access from the disks.

To analyze the jitter of the architecture, we run a different experiment, where strands are requested at regular intervals, and we plot as a function of delay and architecture utilization the probability to process a strand *within* the given delay. This is the *jitter* experiment. The architecture utilization is controlled by modifying the delay

between strand requests.

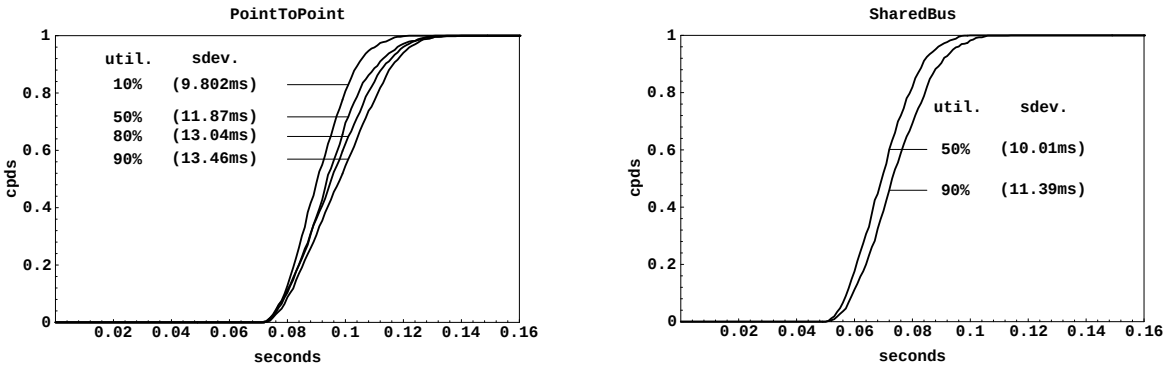

Figure 5 : Reading streams : delay cumulative probability distributions

Figure 5 shows typical strand delay distributions for an 8-disk-node architecture, as a function of the architecture utilization. The utilization of an architecture is defined as the maximal utilization of any component in the architecture. Each curve represents a cumulative probability distribution (cpd). A $(delay, cpd)$ point on the curve represents the probability that a strand is retrieved in less than $delay$. The disk induced delay standard-deviation is 7.07msec. It represents the major part of the delay's standard-deviation for both architectures (around 10ms). The remaining parts are due either to shared bus contention (shared-bus architecture) or in the case of the point to point architecture, due to contention on the communication links or on accesses to the server interface processor's memory.

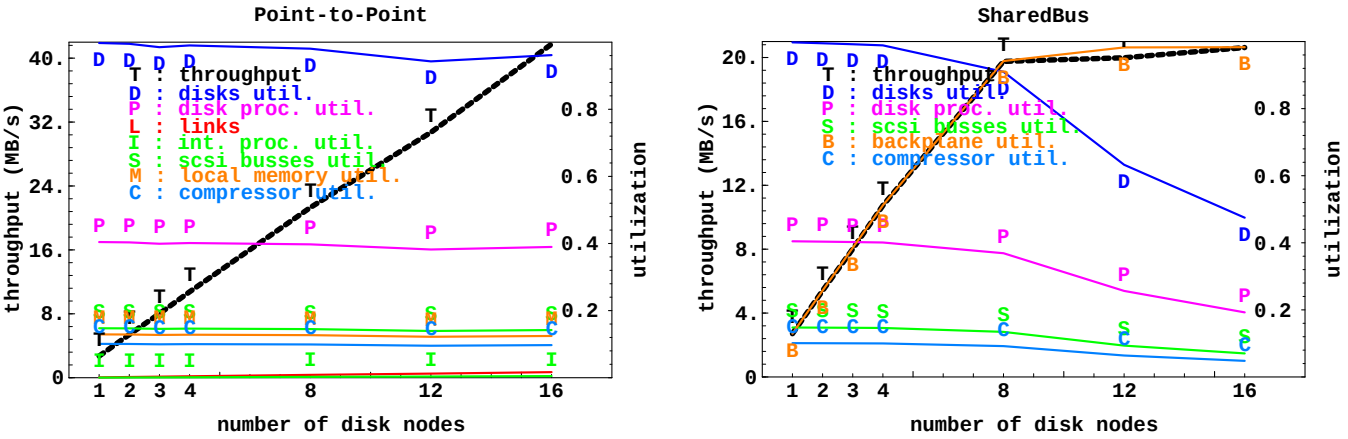

Figure 6 : Resampling uncompressed streams: throughput and utilization

5.2 Resampling and compressing streams

This experiment consists of reading an uncompressed stream from the disks, resampling it, compressing it and storing it in memory, in order to transmit it to the client through the network . The requested stream consists of 256 352-by-240 2-byte frames, and each strand of a single 352-by-240 2-byte frame. The resampling step divides the size of each frame by a factor of 4.

A simple evaluation shows that for a 16 disk-node architecture, the disks will achieve a 32MByte/sec throughput. Each processor resamples its strands at the rate of 7.5MBytes/s (see table 2) and therefore achieves a combined throughput of 120MBytes/s. The compression circuits compress data at the same rate. The bus must transfer the data from secondary storage to memory (40MBytes/s), from memory to processor for resampling

purposes, (100MBytes/s), from processor to memory (a quarter of the initial data size at 60MBytes/s), from memory to the compression circuit (a quarter of the initial data size at 100MBytes/s) and from the compression circuit to memory. Since the last transfer concerns the transfer of compressed resampled data and a compression factor of 15 is assumed, only 1/60 of the initial data size needs to be transferred to memory. This last transfer step can therefore be neglected. The maximum bus throughput for this operation is therefore $\frac{1}{\frac{1}{40}+\frac{1}{100}+\frac{1}{4*60}+\frac{1}{4*100}}$ = 24MBytes/s.

Figure 6 summarizes the throughput and utilization for point-to-point and shared-bus architectures. In the case of the point-to-point architecture, the disks are the bottleneck, regardless of the number of disk nodes. By adding disks to each node, it is possible to make the disk-node processors become a processing bottleneck. In no case do the SCSI busses and disk-node memory bandwidth become a bottleneck. The processor utilization rate of 0.4 shows that two disks per disk-node would increase the processor utilization to 0.8 and double the input throughput. The point-to-point architecture can be further expanded by adding additional disk-nodes, up to 28 disk-nodes connected to a single crossbar chip. in order to sustain a linear throughput increase for up to 112 Mbytes/s (28 disk-nodes * 2 disks/disk-node* 2 Mbytes/s) of input data. Since the data stream gets reduced by a factor of 4 and is further compressed, no bottleneck will occur at the server interface processor. The shared-bus architecture already saturates at 8 disks and 21 Mbytes/s due to the shared bus bandwidth limitation.

The throughput numbers in figure 6 refer to the *uncompressed* data throughput *before* having applied the resampling operation. These are the effective throughput values at the disk and disk-node processor levels. Assuming a compression factor of 15, and a resampling factor of 2 in both X and Y directions, the throughput after resampling and compression is 1/60th of the throughput displayed on the figure. In the case of the shared bus architecture, the backplane bus becomes a bottleneck when the architecture consists of more than 8 disk-nodes. The maximum sustainable throughput is 21MBytes/s, slightly less than our estimated 24MBytes/s. The difference between the estimated and simulated result is due to the contention between the large number of packets traveling on the bus. Many small packets represent strand and SCSI requests that travel between processors and input/output controllers.

5.3 Discussion of the results

With up to 8 disk-nodes, both analyzed multiprocessor-multidisk architecture offer excellent performances for accessing and processing multimedia streams striped over multiple disks. The point to point architecture is well balanced : up to 30 MBytes/s can be accessed from 16 disks, processed and transferred through the links to the server interface processor. In the case where an uncompressed stream is accessed from the disks and is down-sampled or compressed, the bottleneck resides in the disk access throughput. In such a case, the point to point architecture can be scaled up to 28 disk nodes and 56 disks, reaching an uncompressed data throughput of 112 Mbytes/s, whereas the shared bus architecture saturates at only 8 disk-nodes.

Compared with the point to point architecture, the shared bus multiprocessor-multidisk architecture offers a similar disk access performance due to its high shared bus throughput (400MB/s raw throughput). Nevertheless, due to the fact that caches do not behave well in data intensive applications, and to the fact that reading and writing continuous media require shared bus accesses, the bus may become a bottleneck. This occurs when combining processing and data intensive operations such as for example the pipeline composed of parallel disk accesses and parallel resampling of video streams.

In the case of the point-to-point architecture, it is possible to run both disk-reading and reading-with-resampling operations simultaneously on independent streams, and to achieve a substantial fraction of the throughput of both. With sufficient disk resources, the disk-reading operation saturates the server interface processor links, with minimal use of the disk-node processors. Under the same circumstances, the reading-with-resampling operation saturates the processors, with minimal use of the server interface processor links. The two operations can therefore run with minimal interaction. In the case of the shared-bus architecture, both disk-reading and reading-with-resampling operations saturate the bus. These two operations are therefore mutually exclusive.

The point-to-point architecture is well suited for building scalable multimedia servers storing highest quality data, possibly in uncompressed form. Services may be offered for editing data (creating special effects requiring the combination of frames, adding text into frames, etc..), for processing frames as series of images and for reducing the amount of information to be transferred according to the client application requirements and to the network bandwidth. As long as the operations can be applied independently to strands, every disk node processor reads its requested stands from the disks, performs the required operations and either stores the result back onto the disks or transfers them to a server interface processor connected to the client through the network. The only limitation of the point to point architecture is the maximal 30Mbyte/s communication link throughput of the server interface processor.

The analysis of the shared bus architectures suggests that the potential bottleneck represented by the shared bus could be alleviated by considering an architecture where processors and globally addressable local memories would form subsystems able to execute algorithms in parallel and where the shared bus would be accessed only to transfer information between disks and subsystems, to exchange data between the subsystems and to transfer data from subsystems to the network interface. What is needed is a general-purpose DMA controller capable of executing large-data-set move operations, that transfer packets from one component to another in one pass across the bus. The large-data-set move operations are similar to the bit block transfer operations (BitBlt) familiar to the computer graphics community. An architecture where the data would be transferred between local memories under the control of the application would sustain a higher performance than current workstation architectures where data flows *on demand* as a result of cache coherency algorithms that have no prior knowledge of the application domain. Once the shared bus becomes just a means for communicating between processing subsystems, its very large raw bandwidth will be hard to saturate.

6 Conclusion

We have analyzed the behavior of two multiprocessor-multidisk architectures for multimedia oriented stream access and processing operations such as accessing in parallel video streams striped over a number of disks, down-sizing stream frames by parallel resampling, and parallel compression by special compression circuits located in the proximity of disk node processors.

The performance evaluation and the simulations show that the point to point architecture offers scalable performances for a reasonable link throughput (8 MB/s link throughput). The shared bus architecture offers similar performances but at the cost of a sophisticated shared bus offering a hardware throughput of 400MB/s. The shared bus is a potential bottleneck for operations combining data-bound and compute-bound operations such as the pipeline of operations required for reading continuous media from multiple disks and for reducing its size by applying in parallel resampling operations.

The shared bus bottleneck could be removed with an architecture composed of subsystems with globally addressable local memories allowing data to be directly transferred from disks to the subsystems local memories and processing operations to be executed locally.

Both the point to point and the shared bus multiprocessor-multidisk architectures offer multi-media access and processing throughputs in the range of 20 to 40MB/s. Such throughputs are sufficient to design multimedia information servers offering a large palette of services to client stations located on the network. Such powerful servers will become more and more needed, since the growth of the Internet and of similar networks induces a large number of users to access the same servers. These servers have to respond simultaneously to hundreds of requests for data streams at various rates. We are currently addressing this problem by developing on a UNIX-based multiprocessor-multidisk architecture a multimedia server capable of handling a large number of simultaneous continuous media access and processing requests.

7 REFERENCES

[1] Ann L. Drapeau et al. RAID-II : A high-bandwidth network file server. In *Proc. 21th Int. Symp. Computer Architecture*, pages 234–244, Chicago, Illinois, 1994.

[2] R. D. Hersch, B. Krummenacher, and L. Landron. Parallel pixmap image storage and retrieval. In Grebe et al., editor, *Proceedings of the World Transputer Congress*, pages 691–699. IOS Press, 1993.

[3] A. L. Narasimha Reddy and Jim Wyllie. Disk scheduling in a multimedia I/O system. In *Proc. First ACM Conf. on Multimedia*, pages 225–233, Anaheim, California, August 1993.

[4] P. Lougher and D. Shepherd. The design of a storage server for continuous media. *The Computer Journal*, 36(1):32–42, February 93.

[5] Fouad A. Tobagi, Joseph Pang, Randall Baird, and Mark Gang. Streaming RAID — a disk array management system for video files. In *Proc. First ACM Conf. on Multimedia*, Anaheim, California, August 1993.

[6] H. M. Vin and P. V. Rangan. Designing a multi-user HDTV storage server. *IEEE Journal on Selected Areas in Communication*, 11(1):153–164, Janvier 1993.

[7] P. Druschel, Mark B. Abbott, Michael Pagels, and Larry L. Peterson. Analysis of I/O subsystem design for multimedia workstation. In P. Venkat Rangan, editor, *Network and Operating Systems Support for Digital Audio and Video*, volume 712 of *Lecture Notes in Computer Science*, pages 289–301. Springer-Verlag, 1993.

[8] J. Wilkes. Parallel storage systems for the 1990s. In *Proceedings of the 11th IEEE Symposium on Mass Storage System*, pages 131–136, Monterey, 1991.

[9] Benoit Gennart and Roger D. Hersch. Comparing multimedia storage architectures. In *Proc. Int. Conf. Multimedia Computing and Systems*, pages 323–328, Washington, May 1995.

[10] Orran Krieger, Michael Stumm, and Ron Unrau. The alloc stream facility: A redesign of application-level stream I/O. *IEEE Computer*, 27(2):75–82, March 1994.

[11] D. Le Gall. Mpeg : A video compression standard for multimedia applications. *Comm. of the ACM*, 34(4):46–58, April 91.

[12] Richard Stephens. Parallel benchmarcks on the Transtech Paramid supercomputer. In *Proc. World Transputer Congress*, pages 136–146, Como, September 1994.

[13] Adrian Cockcroft. *Sun Performance and Tuning : SPARC & Solaris*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1995.

[14] Ben Catanzaro. *Multiprocessor System Architectures*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1994.