Parallel Processing Algorithms and Architecture for Multimedia On-Demand Servers

Raja Neogi
Motorola Inc., Austin, TX 78720

Meghanad Wagh
Lehigh University, Bethlehem, PA 18015

Abstract: Advances in networking and storage technologies have made it possible to build multimedia on-demand servers that provide service similar to those of neighborhood videotape rental stores. In this work, we propose a specialized high-performance programmable multiprocessor architecture which allows continuous playback of media-streams to large number of clients, a critical factor in multimedia servers. This architecture supports dynamic phase shifts in media distribution which is particularly useful in a distributed environment. We introduce algorithms for maximum concurrency extraction and to avoid task migration in the presence of dynamically changing demand. We show that our solution performs an order of magnitude better than other known solutions.

1.0 Introduction

Recent advances in networking have made it possible for computer networks to support digital multimedia transmission [1]. Coupled with emerging storage technologies, they can be used to build multimedia on-demand services over wide area networks that are expected to permeate residential and commercial premises in a manner similar to existing cable TV and telephone networks. Naturally, a lot of interest exists in the design of servers that can interface between the storage media and the clients demanding information from this media. Such an on-demand multimedia server, which we refer to as Multimedia Server in this paper, provides services similar to those of neighborhood videotape rental store. A typical multimedia system is shown in Fig. 1 below. Multimedia server is the nerve center of such a system. It digitally stores media information such as the educational documentaries, entertainment movies, advertisements, etc., on a large array of extremely high-capacity storage devices that are permanently on-line. The storage used in multimedia servers comprise of optical or magnetic disks that are random accessible with short seek time. The server is connected to client display sites via a high-speed network sub-system. Clients can interactively choose a multimedia object by scanning several hierarchically organized attributes like object type (comedy, tragedy, thriller, educational etc.), few seconds of media clip extracted from the object, and request its retrieval for real-time playback [10] on their display sites. The multimedia server, if it has necessary resources such as service-time and buffer-space, serves client requests by connecting to individual display sites and transmitting the chosen multimedia object. Many such servers can share storage load or serve as temporary banks of information. One can generate delivery strategies for optimized distribution of media (video, audio, image and data) information based on unit cost for direct transmission as opposed to store and forward transmission.

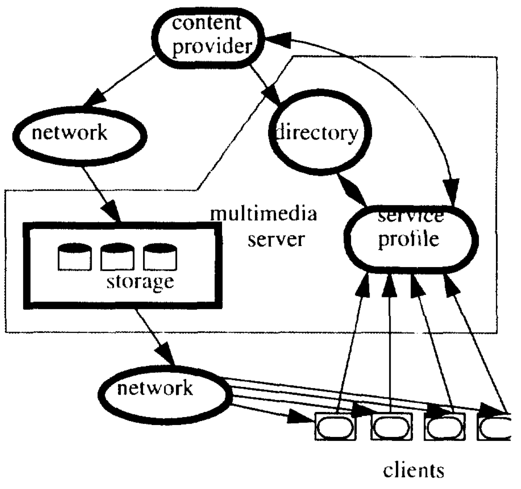

Fig. 1: multimedia system model

A critical requirement for building any multimedia on-demand service is the need for guaranteeing continuous playback of media streams. For continuous playback, server must support continuous retrieval from disk, network sub-system must support timely delivery of media quanta to display sites, and display sites ought to avoid buffer overflow or underflow. Retrieval continuity can be guaranteed by constraining the separation of successive media quanta. Network subsystem has to lookahead and reserve network resources for each client such that bounds of delay jitter are not violated. Also, display sites need to prefetch sufficient media quanta to match the slack between playback rate and network latency.

Multimedia system design is a relatively new field with most of the reported work being restricted to feasibility studies for development of such systems [11,12]. Ferrari and Verma have introduced a network subsystem design for continuous delivery of media quanta [2]. Techniques for avoiding buffer overflow and underflow at display sites in the presence of non-deterministic playback rate variations were addressed by Ramanathan and Rangan [3]. Work on multimedia storage systems for still images and/or audio were reported by several authors [4-6,8,9]. Recently Rangan and Vin [7] have introduced a generalized multimedia system model with a detailed analysis of its feasibility. It is expected that the client bandwidth for such real-time multimedia service will grow rapidly. To accommodate large number of clients, it is essential to build a fast but inexpensive multimedia servers that can directly interface to wide area networks via asynchronous transmission mode (ATM) switching fabric.

The server architectures proposed and analyzed till now cannot support expected rapid growth in the number of clients. However, an inspection of the data search, retrieval and transmission processes shows that it is possible to do many of these tasks concurrently. By using a parallel computer system, one may be able to exploit this concurrency. Further, recent advances in semiconductor technology make it possible to build application specific multiprocessor architectures on a single chip. In this paper we propose a low cost high performance multiprocessor implementation to realize a real-time multimedia server. Our architecture provides for continuous retrieval of media streams from disks for a large number of clients simultaneously. It supports interactive services, real-time media distribution and cacheing phase-shift of multimedia objects from either native or non-native content providers. Dynamic deadline driven scheduling policy is used to meet the continuity constraint for media quanta delivery. Our analysis shows that the performance of our architecture is an order better than those of the earlier systems.

2.0 Proposed Multiprocessor Architecture

The proposed architecture is shown in Fig.2. It can best be described as a single-chip, heterogeneous architecture connected to an off-chip shared memory (implementable as DRAM) and an array of high speed optical disks. The processors are assigned specific functionalities. Each local stream controller (LSC) interfaces directly to the disk subsystem and fetches data through its pre-assigned head. Assignment is made by selecting a free head in the disk that carries the wanted media segment. In order to enhance multitasking, individual stream controllers can switch between tasks. A processor dedicated to a bank of DRAMs (Dynamic random access memory) is known as a phase-shift controller (PSC). The processor specialized in communication is known as an adaptive network interface (NI). Through NI, the multimedia server can send data and messages to neighboring servers and the client sites. Similarly it can receive client and other server requests. In addition, for control of the architecture and global tasks, an interactive services processor (ISP), a transaction monitor (TM) and a master controller (MC) are used. All of the processors are connected through a high-speed bus called the media-bus to handle the periodic transactions and a local-bus to handle aperiodic transactions.

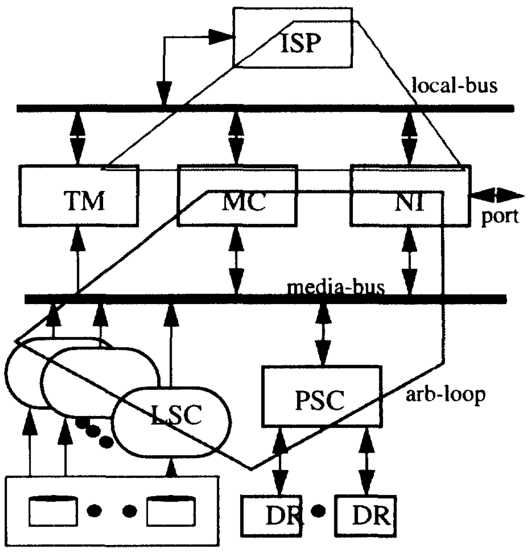

Figure-2: multiprocessor architecture

A composite multimedia object can be represented as a collection of continuous media-quanta (e.g. audio and video) tied together by synchronization information. Such multiplicity of media streams can be abstracted as a rope. For effective digital storage and retrieval it is broken up into discrete segments called media-block. Each media-block has a fixed number of packets each of which contains fixed-size compressed media stream (bits) identifiable by packet number and packet type (video or audio). In our discussion, we refer to a sequence of continuously recorded media-blocks (video frames and audio samples) that constitute a composite multimedia object as a strand. In our server model we have assumed that media strands are located in write-once read-only high performance optical disks (such as CLV and WORM) [8]. Retrieval of media-block incurs retrieval time and additional latency in the form of seek and rotation time. The disk head has to seek to the right track and then rotate along it to grab media blocks from the appropriate disk-block. Storage of media blocks belonging to a strand could either be constrained or unconstrained. Problem with unconstrained placement in a multitasking environment is that, context switch for fetching media-blocks from different strands can often result in real time playback violations. Vin and Rangan have introduced a constrained placement storage policy optimizing media-block granularity and scattering [10].

Another requirement for any multimedia server is to respond to dynamically shifting storage requirements during delivery of media quanta. For example, client is being served a two hour movie that started at 2 P.M. Another client requests the same movie at 3 P.M. Rather than fetching all media-quanta from read-only disk-array, server bandwidth is increased by fetching captured data from faster memory. In our multiprocessor architecture, PSC snoops on initial media-bus transaction to capture and later serve the required phase of the strand. Captured data is stored in DRAMs with only few nanoseconds of access-time. Placement of media-blocks in DRAM, could be either constrained or unconstrained. PSC refers to any stored strand fragment by saving handle and storage patterns for each active strand in its native buffer. As illustrated in Fig.1, multimedia systems comprise of servers connected directly to clients and other servers through networks. So, a neighboring server could either feed media-quanta or request for it. In the latter case, neighboring server is just another client but in the former, it is a content provider that supplies media quanta periodically for later consumption by native clients. This way, PSC along with its storage resources provide store-and-forward capability to the proposed multiprocessor based multimedia server architecture. Garbage collection policies for recovering free areas of memory is programmed in MC which communicate with the PSC by port-to-port type of media bus request (MBR).

At any point of time, only finite number of clients can be served. The main purpose of designing a multiprocessor based solution for multimedia servers is to expand the number of clients that may be simultaneously served. In our proposed architecture, ISP manages admission of new clients into the system. Once a client is admitted, it leaves only after its service is complete. Admission control strategy used by the ISP is introduced in the next section. Also, before requesting service or admission, clients can interactively query local server catalog database. Optimized hierarchical organization of such database is beyond the scope of this work. Transaction Monitor (TM) audits all transactions on media bus. It records admission of clients and units of media quanta delivered or received. Upon completion of service, this information is packed and delivered to the network for consolidating server charges to network charges. Security issues associated with media strands are programmed in MC which can override accepted service request. In a complicated real-time multiprocessor system, as the one proposed, it is possible that there are real-time violations in the continuous delivery of media quanta at peak load. MC monitors all periodic transactions on media-bus to detect such a violation and can be programmed to respond to faults depending upon its severity. All transactions between multiprocessor system and the ATM switching fabric are routed through an adaptive network interface (NI). It adapts to different internal and external bus rates.

3.0 Concurrency Extraction

Each transaction on media-bus must have a control, data, arbitration and snoop phase. All except data phase is completed in a fixed number of cycles. Since, bus resource is required for both data and control phase, these phases need to be sequential. Each control phase is preceded by snoop and arbitration phase, and followed by data phase. Data phase for a control phase follows the next control phase, as shown in Fig.3. The LSCs and PSC support multitasking and prefetch data for a task in such a manner that all data is available when its turn to load the media-bus comes. This is accomplished by snooping on the media-bus prior to the preceding control phase of transaction and initiating prefetch from the storage media. Null data phase is only possible when processor wins the media bus for one of its native task before prefetch from storage media for the transaction is complete. By allowing task migration in LSCs when a task dies (service to client is complete) and mapping tasks (initiate service for new client) in such a way that consecutively executing tasks do not belong to the same LSC, prefetch contention can be avoided. In arbitration phase, each LSC and PSC figure out local task winner, based on real-time deadline, and along with NI (for inbound traffic) communicates arbitration value to MC through the arbitration loop shown in Fig. 2. MC then decides the winner of media bus. Local bus control-flow and data-flow is identical to that of the media-bus except that priority encoding is based on events rather than on the deadline.

4.0 Dynamic Scheduling

Each processor driving the media-bus (LSCs, PSC, MC and NI) has a snoop module that captures the most recent control-phase to derive its local prefetch strategy. For example, in Fig. 3, *sp(i)* comes after *cp(i)*. In *sp(i)*, processor checks out its local processes to establish a winner that can potentially win the *ap(i+2)* arbitration and consequently reserve data-phase *dp(i+2)* through control-phase *cp(i+2)*. In order to explain the algorithm to establish such a winner, we introduce three terms: *deadline*, *laxity* and *readiness*.

Definition 1: deadline for a process or task, *D(i,j)*, is defined as the time remaining (normalized to clock-cycles) between now and the absolute deadline for task *j* in processor *i* to reach the network. Both the local scheduler for each processor and the MC maintain this parameter for each active task.

Definition 2: Laxity for a process or task, *L(i,j)*, is defined as the time difference (normalized to clock-cycles) between now and the time media-bus control should be obtained to successfully dispatch media-blocks without violating any real-time deadline. This can be calculated by subtracting bus-latency from *D(i,j)*.

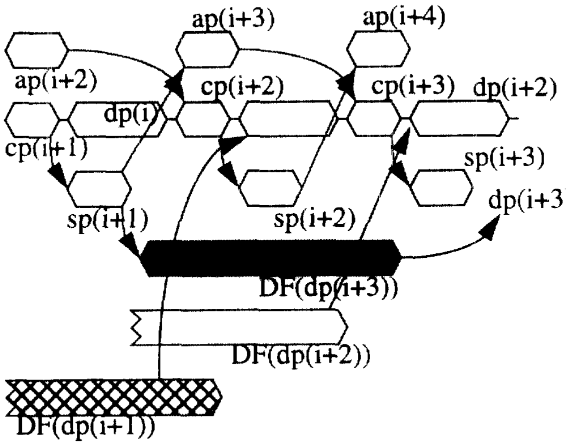

Fig. 3. Execution concurrency in the architecture

Fig. 4 below illustrates the relation between the deadline and laxity.

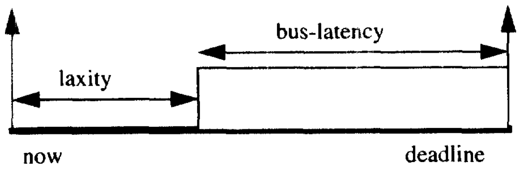

Fig. 4. Relation between Laxity and Deadline

Definition 3: Readiness of a processor for a task or process, *R(i,j)*, is a condition defined as its ability to prefetch all data before execution of the corresponding data-phase transaction.

The symbols used in this and the next section are listed in Table 1. Media-bus winner is derived in two stages. Each media-bus driver that supports multitasking (LSCs and PSC) comes up with a winner by selecting the highest priority task from its native pool of active tasks. This is performed by the native snoop module in snoop phase and is given as,

$$P^i = min_{j \in T_i}\{D((i,j) \wedge R(i,j))\}$$

where P^i indicates the id of the winning task in processor *i* and T_i, the set of active tasks in processor *i*. Each media-bus contender (LSCs, PSC and NI) fills up its slot in the arbitration packet (initiated by MC along the arbitration-loop) with the local winner id and bus-latency (for laxity calculation). Arbitration is performed by MC in arbitration phase and is given as,

$$W = min_{1 \le i \le L}\{D(i, P^i) \wedge L(i, P^i)\}$$

where *W* indicates the arbitration winner and *L*, the number of processors competing for data flow transaction on media-bus.

Table 1: List of symbols used

Symbol	Explanation	Unit
η_{sm}	granularity of storage	frames per media block
s_{vf}	size of video-frame	bits per frame
β	media-bus width	n/a
R_{vp}	video play-back rate	frames per second
R_{sm}	disk-data transfer rate	bits per second
l_{rot}	rotational latency	seconds
l_{seek}	seek latency	seconds

Lemma 1: At least three LSCs are needed for maximal concurrency.

Proof: We consider the worst case condition i.e. all clients requesting service require original media-quanta (LSC service). Maximum extent of *DF(dp(i))* overlaps with *dp(i-1)*, *dp(i-2)* and portion of *dp(i-3)*. This means *DF(dp(i- 2))* and *DF(dp(i-1))* concur with the execution of *DF(dp(i))*. Noting that each LSC has only one head to perform disk prefetch, three consecutive tasks need to be on separate LSCs.

Lemma 2: Earliest response time is variable, but is bounded and is greater than next three data-phases, i.e.,

$$T_{response} \geq \frac{\sum_{-1}^{1} dp\,(i+k)}{f_{clk}}$$

where $T_{response}$ indicates acceptance-to-service lag time and f_{clk} the operating frequency of the system.

Proof: Let a high priority task be granted admission before snoop-phase *sp(i)*. Then, even if it wins local arbitration and global arbitration *ap(i+2)*, data transaction slot can only be reserved through *cp(i+2)*. This implies data-phases *dp(i-1)* (concurrent to *sp(i)*), *dp(i)* (concurrent to *ap(i+2)*) and *dp(i+1)* (follows *cp(i+2)*), have to come before *dp(i+2)*. So, earliest an incoming task can be served is the sum of the next three data-phases. Each data-phase can be of different size depending upon the number of media-blocks being served. However, media decompression parameters can fluctuate boundedly (eg. playback rate, frame size are not widely different), so one can derive an average response time that is marginally different from the actual response time. Therefore the response time is bounded.

Each media-bus driver needs to have a data-buffer to save prefetched data, and control buffer to store control data structures for active tasks. Moreover, processors that support multitasking by context switching need to have context buffer. Both control-buffer and context-buffer sizes scale linearly with control parameters monitored and maximum multitasking limit, respectively. Bits required per control parameter or task state is also relatively small. The data-buffer, on the other hand is huge and it is expensive to have more than one data-buffer per processor. Hence, we introduce the following lemma.

Lemma 3: maximum data-buffer size for any of the media bus controllers (LSCs and NI) is given as,

$$M_{DataBuffer} = \frac{R_{sm} dp_{max}}{f_{clk}}$$

where dp_{max} indicates maximum cycles allowed per dataphase.

Proof: Each prefetch from storage-media is triggered by a snoop phase *sp(i)*. This allows the processor data-phases *dp(i-1)*, *dp(i)* and *dp(i+1)* to get data from storage-media. Assuming that there is no prefetch contention and an upper bound for the maximum number of cycles that can be allocated to any processor on the media-bus, we conclude that the data-buffer size is bounded.

5.0 Performance Analysis

Consider a multimedia server that is required to concurrently serve n clients by supplying them strands S_1, S_2, ..., S_n. Since each request is periodic, multimedia server can service them by proceeding in rounds. Suppose that during each round, multimedia server retrieves k_1 media-blocks of strand S_1, k_2 of strand S_2, etc. The total time required to complete the round should not exceed the minimum playback durations of $k_1, k_2, \ldots, k_n$ blocks. The the continuity requirement for each strand can be satisfied *if and only if* the service time per round does not exceed minimum of the playback durations of $k_1, k_2, \ldots, k_n$ blocks. That is,

$$\sum_{i=1}^{n} \left(\frac{k_i s^i_{vf} \eta^i_{sm}}{f_{clk} \beta} \right) \leq min_{1 \leq i \leq n} \left(\frac{k_i \eta^i_{sm}}{R^i_{vp}} \right) \qquad (1)$$

The multimedia server can service all n clients simultaneously if and only if $k_1, k_2, \ldots, k_n$ can be determined. Since we have n variables and one equation, determination of $k_1, k_2, \ldots, k_n$ require additional techniques (all the other parameters besides this are supplied by the client seeking service). We use two approaches to solve this problem. In the first case, we use same value of k_i for all clients yielding a round robin service approach. This gives the number of media blocks using the Round Robin strategy [1], k_{rr}, as

$$k_{rr} = \frac{f_{clk} \beta U^{min}}{\sum_{i=1}^{n} s^i_{vf} \eta^i_{sm}}$$

where U^{min} denotes the right hand side of (1).

This however may not be the optimal number of clients, because, strand with maximum playback rate would have retrieved exactly the number of media-blocks it needs for the duration of service round, other strands with smaller playback rates will have retrieved more media blocks than is needed in each service round. To get around this problem, we allow retrieved media-blocks to be proportional to clients' playback rate (also called quality proportional approach [1]). The number of media blocks, k_0, is obtained in this case as,

$$k_0 = \frac{f_{clk} \beta U^{min}}{\sum_{i=1}^{n} s^i_{vf} R^i_{vp} \eta^i_{sm}}$$

On the other hand, in a uniprocessor scenario, the number of media blocks must satisfy the following equation,

$$n\,(l^{max}_{seek} + l^{max}_{rot}) + \sum_{i=1}^{n} \left(\frac{k_i s^i_{vf} \eta^i_{sm}}{R_{sm}} \right) \leq U^{min} \qquad (2)$$

The first term in (2) is a result of the fact that latency due to seek and rotation is incurred every time context is switched from one client to next in every period.

Using quality proportional approach, mentioned above, (1) and (2) can be rewritten as,

$$\frac{k_0 \sum_{i=1}^{n} s^i_{vf} R^i_{vp} \eta^i_{sm}}{f_{clk} \beta} \leq U^{min}$$

(3)

and,

$$n\,(l^{max}_{seek} + l^{max}_{rot}) + \frac{k_0 \sum_{i=1}^{n} s^i_{vf} R^i_{vp} \eta^i_{sm}}{R_{sm}} \le U^{min}$$

(4)

Written in this form, inequalities (3) and (4) show that the number of clients served in both multiprocessor and the uniprocessor architectures is bounded above. Note now that the typical values of R_{sm}, f_{clk} and β are 1 Gbps, 50-100 Mhz and 64 respectively. Thus the product $f_{clk}\beta$, denominator of the left hand side of (3), is *6.4* times larger than the corresponding denominator of (4). By comparing (3) and (4) in the light of this, one can see that (3) admits a much larger n than (3) (at least *6.4* times larger). The term corresponding to the seek times in (4) further reduces the maximum n satisfying (4). Thus using our multiprocessor architecture, a much larger number of clients can be served through the multimedia server as compared with earlier architectures which were based on a single processor. Finally, it should be pointed out that as semiconductor technology advances, both f_{clk} and β are expected to increase, further enhancing the performance of our multiprocessor multimedia server.

6.0 Scalability

From (3) one can derive the maximum number of clients supported, N^{max} as,

$$N^{max} = \frac{f_{clk}\beta}{R_{vp}s_{vf}}$$

(5)

Note that N^{max} is independent of block sizes. However, each client's compressed bitstream buffering capacity is limited and this limits the maximum size of a media block. Typical values of R_{vp} and f_{clk} are 30 frames per second and 66.7 kilobits per frame on average. This yields a maximum broadcast capacity of about 3200 clients. However, in near future, advances in semiconductor technology will permit use of higher clock rates f_{clk} and bus-widths β for single chip implementations. This will scale up performance of our proposed system even further. For example, a system with 128 bit wide media bus operating at 200 MHz can support more than 12000 clients.

Scaling performance to these high values requires one to effectively combat the problem of task migration. Task migration is a result of using less number of LSCs than the number of clients being supported (for obvious economic and feasibility reasons). Thus each LSC is supporting multiple strands. The LSCs are provided a periodic access to the media bus to dump their data. This ensures that each LSC has maximum possible time to retrieve and buffer the data from the disk. Tasks are serviced in the order of their priority in each of these rounds. When a new task enters the system, the servicing sequence has to be changed by taking the priority of this new task into account. This might force migration of all the tasks that have a priority lower than the new task. To illustrate this, consider three LSCs marked *a*, *b* and *c*, and seven tasks P_1 through P_7 with decreasing priorities. (Each task is a request from a client for a particular media strand.) One may let LSC *a* service tasks P_1, P_4 and P_7, *b*, the tasks P_2 and P_5 and *c*, tasks P_3 and P_6. By sequencing through the LSCs *a*, *b* and *c* in that order repeatedly, one would then be able to service tasks in their priority order and each LSC will have atleast two data phases to fetch its data from the disk. The processing would follow the timing shown in Fig. 3 and will proceed without any problems. Now, if a new task, $P_{2.5}$ of priority higher than that of P_3 but lower than P_2 comes into the system, than one would have to modify the mapping of tasks to the LSCs to take into account this new reality. LSC *a* would now service tasks P_1, P_3 and P_6, *b*, the tasks P_2, P_4 and P_7 and *c*, tasks $P_{2.5}$ and P_5. By sequencing through the LSCs *a*, *b* and *c* as before, one would be able to service tasks in their priority order. But this new mapping of tasks to LSCs calls for migration of the five tasks of a priority lower than the new task (P_3 through P_7) from an LSC to another.

In our earlier performance analysis, we had ignored the delays due to migration. However with high performance systems, data phases take less time and task migration might become a significant burden. A system will service all the designed number of clients satisfactorily if it is possible to map all the tasks on the available LSCs such that by sequencing through the LSCs periodically, all the tasks are serviced in their order of priority and each LSC gets at least two data phases (as in Fig. 3) to fetch the data for any task assigned to it. We now show that if the system has atleast *5* LSC processors, then it is possible to insert a new task into the system without any task migration.

Theorem 1: No task migration (due to remapping to accommodate a new task) is required if the system has at least *5* LSC processors.

Proof: Denote the tasks in their order of priority by P_i, $i = 1, 2, .. n$ and $\phi(P_i)$, the LSC processor to which task P_i is mapped. Since the tasks need to be processed in their order of priority, the sequencing of the processors in each round is,

$$\phi(P_1), \phi(P_2), \ldots, \phi(P_n)$$

(6)

If the number of tasks is less than the number of processors θ, we map one task to each processor. Clearly this would avoid the bus contention. We now assume that each processor has at least one task and a new task P_{new} with a priority higher than that of P_t comes in. The new mapping of tasks to processors, ϕ_n, may be defined as:

$$\phi_n(P) = \begin{cases} \phi(P) & \text{if P is an old task} \\ \phi(P_{t+2}) & \text{if } P = P_{new} \end{cases}$$

Clearly this mapping avoids any task migration. The new periodic sequencing through the processors is now given by

$$\phi_n(P_1), \phi_n(P_2), \ldots, \phi_n(P_{t-1}), \phi_n(P_{new}), \phi_n(P_t), \ldots \phi_n(P_n)$$

From the definition of the function ϕ_n, this sequence can be rewritten as:

$$\phi(P_1), \ldots \phi(P_{t-1}), \phi(P_{t+2}), \phi(P_t), \phi(P_{t+1}), \ldots, \phi(P_n) \quad (7)$$

It is easy to verify that the sequencing of (7) does service the tasks in their priority order. To show that it avoids bus contention, we demonstrate that no processor is accessed before it is allowed two data phase cycles so as to have sufficient time for its data fetch. Since the only difference between the sequence of (6) and that of (7) is due to the new task mapped to processor $\phi(P_{t+2})$, one only has to verify the absence of bus contention for this processor. But this is also obvious from (7). Thus this mapping and sequencing avoids bus contention without any task migration.
Finally, note that the processor sequence around this new addition $\phi(P_{t+2})$, i.e., $\phi(P_{t-2})$, $\phi(P_{t-1})$, $\phi(P_{t+2})$, $\phi(P_t)$ and $\phi(P_{t+1})$ should all be distinct processors, or else some processor may be accessed earlier than two data phases. This proves that there should be at least five processors to avoid bus contention without any task migration.

7. Conclusion

Multimedia servers are different from other servers by virtue of their real-time continuity requirements for distribution of media streams. Rapidly increasing popularity of multimedia necessitates the design of new multimedia servers that can simultaneously satisfy a large pool of clients.
In this paper we have proposed a multiprocessor based multimedia server to address this problem. Our architecture takes full advantage of the concurrency between the disk fetch, bus transfer and network delivery. The processors in our architecture are specialized to do the assigned tasks and use an interconnection strategy that maximizes the performance of the system besides meeting the demands of the application. The data transactions are pipelined for optimal utilization of resources. Finally, each local stream controller (LSC) and the phase shift controller (PSC) uses a scheduler for swapping active tasks to achieve multitasking. To make the architecture more flexible, we introduce programmability in handling fault tolerance, security, unit cost estimation for media-quanta delivery and prioritizing events on the local bus. This architecture can also handle dynamic phase shifts during media-distribution. We have introduced algorithms for maximum concurrency extraction and for avoiding the task migration even when the demand on the server changes dynamically. With the current technology, our solution can perform an order of magnitude better than other known solutions.

8. References

[1] P. V. Rangan, H. Vin and S. Ramanathan, "Designing an on-demand multimedia service", *IEEE Communications Magazine*, vol. 30, no. 7, pp. 56--65, Jul. 1992.

[2] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks", *IEEE Journal on selected areas of Communication*, vol. 8, no. 3, pp. 368--379, Apr. 1990.

[3] S. Ramanathan and P. V. Rangan, "Adaptive feedback techniques for synchronized multimedia retrieval over integrated networks", *ACM Transactions on Networking*, vol. 1, no. 2, pp. 246--260, Apr. 1993.

[4] C. Abbot, "Efficient editing of digital sound on disk", *Journal of Audio Engineering*, vol. 32, no. 6, pp. 394--402, June 1984.

[5] B. C. Ooi and A. D. Narasimhalu, "Design of a multimedia file-server using optical disks for office applications", *IEEE computer society Office Automation Symposium*, Gaithersberg, MD, pp. 157--163, Apr. 1987.

[6] J. Gemmel and S. Christodoulakis, "Principles of delay sensitive multimedia data storage and retrieval", *ACM Transactions on Information systems*, vol. 10, no. 1, pp. 51--90, 1992.

[7] H. M. Vin and P. V. Rangan, "Designing a multiuser HDTV storage server", *IEEE Journal on selected areas of Communication*, vol. 11, no. 1, pp. 153--164, Jan. 1993.

[8] C. Yu and W. Sun, "Efficient placement of audio data on optical disks for real-time applications", *Communications of ACM*, vol. 7, no. 1, pp. 862--871, Jul. 1989.

[9] P. V. Rangan and H. Vin, "Efficient Storage Techniques for Digital Continuous Multimedia", *IEEE Transactions of Knowledge and Data Engineering*, vol. 5, no. 4, Aug. 1993.

[10] Raja Neogi and A. Saha, "Embedded Parallel Divide and Conquer Video Decompression Algorithm and Architecture for HDTV Applications", *manuscrpt* in Review

[11] Ron Buck, "The Oracle Media Server for nCUBE Massively Parallel Systems", *Proc. of the 8th. Intl. Parallel Processing Symposium*, pp. 670-673, Apr. 26-29, 1994.

[12] B. A. Gennart and R. D. Hersch, "Multimedia Performance Behavior of the Giga View Parallel Image Server", *Proc. of 13th. IEEE Symposium on Mass Storage Systems*, pp. 90-98, Jun. 1994.