Improved Multimedia Server I/O Subsystems

Michael Weeks, Hadj Batatia, Reza Sotudeh
Computer Architecture Research Unit
University of Teesside
Middlesbrough
England, UK.
{michael.weeks, h.batatia, r.sotudeh}@tees.ac.uk
Telephone: 44+ (1642) 342494
Fax: 44+ (1642) 342401

Abstract

The main function of a continuous media server is to concurrently stream data from storage to multiple clients over a network. The resulting streams will congest the host CPU bus, reducing access to the system's main memory, which degrades CPU performance. The purpose of this paper is to investigate ways of improving I/O subsystems of continuous media servers. Several improved I/O subsystem architectures are presented and their performances evaluated. The proposed architectures use an existing device, namely the Intel i960RP® processor. The objective of using an I/O processor is to move the stream and its control from the host processor and the main memory. The ultimate aim is to identify the requirements for an integrated I/O subsystem for a high performance scalable media-on-demand server.

1. Introduction

Continuous media, such as audio and video, have different characteristics compared to conventional data. They are typically data intensive, even when compressed, and time dependent. These characteristics place a number of requirements on the servers' [1 & 2], the communication, and even on the end-system configuration. At the server level, these isochronous media impose many constraints mainly on the architecture, the storage [3], and the operating system. They require real-time handling especially at the I/O subsystem.

The design of a media-on-demand (audio or video on demand) server must take into account a number of issues. The following sections describe some of the design factors.

a) Server access style

Two access technologies influence multimedia server design, client 'pull' and server 'push' [4]. Client pull technology is similar to that used for file servers for handling text and other aperiodic data types, whereby the client explicitly requests data from the server. In this case, the design of the client application requires greater complexity than the design of the server. Server push is the traditional choice for continuous media server designs as it is more suited to the provision of, and interaction with, concurrent streams. To initiate a media stream the client transmits a request to the server, whereupon the server delivers, and manages, the selected data stream to the client. These types of server require a more complex design, as it must store the state of each media stream.

b) Transfer rate

The transfer rate should be sufficiently high to support multiple simultaneous clients. The transfer rate is dependent not just upon hardware, but also upon the operating system and the application software.

c) QoS

The server should provide streams to the client with a guaranteed Quality of Service (QoS), by implementing disk scheduling, and admission control algorithms. Real-time disk scheduling routines ensure continuity of the media stream by determining the most efficient method for retrieving rounds of data from the hard disk. This is more easily implemented with 'Constant Bit Rate' (CBR) coded streams rather than with 'Variable Bit Rate' (VBR) coded streams. Similarly, read only files enhance disk-scheduling performance due to contiguous data placement on disk.

Admission control algorithms guarantee end-to-end performance by preventing stream overload. Admission control does not only guarantee QoS, but other features may also be necessary. For example, the media contents of a video-on-demand server are a marketable commodity, therefore security measures will be necessary to validate the user before access permission is granted, and accounting services will be required to charge the users.

d) Scalability

A scalable architecture allows an increase in the

number of client streams for a proportional increase in the cost.

e) Interactivity

With multiple media files on a server, the user must be able to browse or search the server content, before making a selection. In addition to file management, the server should store metadata that characterises each file's content.

To provide true 'media-on-demand' features, the user must be able to interact with the data stream to perform features such as PLAY, STOP, FAST FORWARD, REW, PAUSE, etc.

The purpose of this paper is to investigate the I/O subsystems of continuous media servers. Improved I/O subsystem architectures based on current technologies are suggested. The ultimate aim is to identify the requirements for an integrated I/O subsystem for a high performance scalable media-on-demand server.

2. Investigation

The server design under initial investigation is the traditional single CPU system that utilises 'push' technology (Figure 1). To simplify matters, we consider only non-editable CBR coded media streams. The system described utilises a single PCI bus for its I/O devices, which for this case study are SCSI for storage, and ATM for the network interface. The software drivers for the I/O devices utilise a double buffering scheme in the system's main memory, which enable the smooth transfer of media data from the SCSI adaptor to the network interface card.

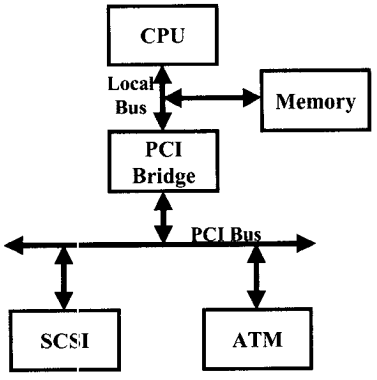

Figure 1: Architecture target for improvement

Although client stream interaction will occur, it will be random and infrequent, and can therefore be considered negligible from the viewpoint of I/O subsystem design. The most frequent state for a stream will be in playback mode, whereby multimedia data is streamed to the client without any user interaction. During playback, the majority of the CPU's local bus traffic will be due to media data streaming from the storage device, via the dual buffering scheme in primary memory, to the network device. This duplication of traffic on the CPU local bus is greater than twice the actual data being transferred. This creates a bottleneck when accessing primary memory, which degrades CPU performance. Consequently, this bottleneck makes the scalability of the single CPU design very poor.

Using semi-autonomous I/O devices, CPU stream control can be reduced substantially. Instead of the CPU supervising the transfer of every item of data, it simply initiates the I/O device to transfer a block of data. On completion, the I/O device informs the CPU by the use of interrupts. For one stream these interrupts can amount to hundreds per second, each requiring a CPU response that switches context, and executes an interrupt routine. With 10^2 to 10^3 media streams, this can amount to a sizeable proportion of the CPU's processing time.

3. I/O subsystem improvement

To maximise CPU utilisation, the stream and its control must be migrated from the processor. To achieve this, we looked at utilising current technology, in particular, Intel's i960RP® intelligent I/O processor, modelled in several variations. The next section contains an overview of the i960RP® device.

3.1 i960RP®

The i960RP® is a high performance embedded processor that has been designed for use as an intelligent I/O processor [5]. The device's main features are (Figure 2):

- i960JF® core processor;
- PCI to PCI Bridge unit;
- Primary and secondary PCI Address Translation Units (ATU);
- Messaging Unit (MU);
- Primary and secondary PCI DMA units;
- Memory Controller;
- Bus arbitration units.

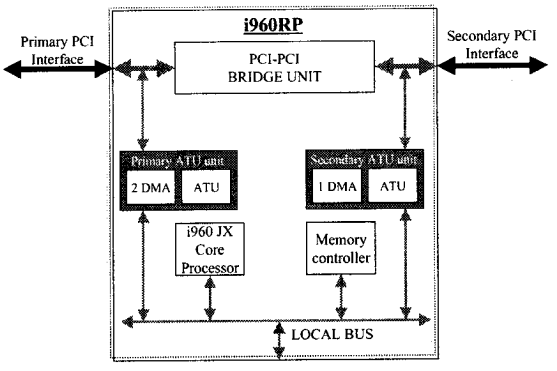

Figure 2: Simplified block diagram of the i960RP®

The core processor is a 32-bit superscalar RISC design

that operates at 33Mhz, and utilises interleaved 32-bit memory via the 80960 local bus. This bus is a 32-bit wide local bus with multiplexed address and data lines. The i960RP® connects to a host processor via its primary PCI bus and appears as a multi-function PCI device.

The ATU's are the interfaces between the PCI buses and the 80960 local bus. The i960RP® contains two ATU's, one for the primary and the other for the secondary PCI buses. The ATU's can burst transfer up to 2kB, and allow inbound and outbound address translations. They can handle multiple inbound transactions by simultaneously processing PCI read and write transactions. Address translation is achieved using an address windowing scheme that determines which addresses to claim and translate.

The PCI-to-PCI bridge operates as an address filter between the two PCI buses, in addition to extending the number of loads that a PCI bus may have. The bridge is programmed with a range of addresses that determine the secondary address space. All PCI read transactions traversing a PCI-to-PCI bridge are processed as delayed transactions.

3.2 Single I/O processor

Removing the bottleneck caused by media streaming to main memory would enhance the performance of the system under investigation. A first step in the design improvement consists of migrating the dual buffering scheme from the main memory to the i960RP® local memory, thereby increasing the host CPU's processing efficiency. In such a case, the i960® core processor is idling.

However, host CPU efficiency can be further improved by migrating stream control, and in particular, I/O device interrupt processing to the i960RP®. To investigate this further, the PCI bus and i960RP® PCI-to-PCI bridge handling of interrupts are first presented.

The vehicles for interrupt passing between the system devices are the PCI buses INTx# lines. With the i960RP® connected to the primary PCI bus, the PCI bus interrupt lines are as shown in Figure 3.

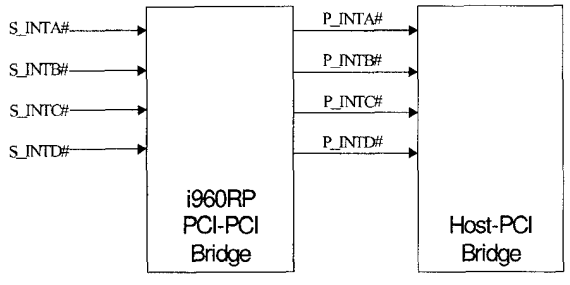

Figure 3: PCI interrupt lines

The PCI-to-PCI bridge may individually route the secondary PCI bus interrupt lines onto the primary bus interrupt lines, or to the i960RP® core processor depending upon the contents of the associated memory mapped register. The interrupt lines always travel upstream. The ATM and SCSI devices use the interrupt lines to signal to their drivers that they have finished their current task. Therefore a software driver must be executing on a processor upstream of the corresponding I/O device, in order to avoid complicated and time-consuming interrupt routing schemes. Providing maximum system performance requires rapid interrupt processing, which restricts possible designs to the following.

a) The ATM and SCSI devices are on the secondary PCI bus. The device drivers for both PCI devices reside on the i960RP®;
b) The ATM device is installed on the primary PCI bus, with its driver on the host processor. The SCSI adaptor is installed on the secondary PCI bus with its driver on the i960RP®;
c) The SCSI device is installed on the primary PCI bus, with its driver on the host processor. The ATM adaptor is installed on the secondary PCI bus with its driver on the i960RP®;
d) Any of the above device interconnections, but with the device drivers staying on the host processor, and the i960RP® acting as a PCI Memory controller/PCI-PCI bridge.

The first connection scheme can be ignored as we wish to balance the sub-streams between the two PCI buses. Similarly, the objective is to remove the I/O drivers from the host processor, so the fourth scheme is not relevant. The second and third designs are compromises, therefore, the second design has been evaluated (Figure 4).

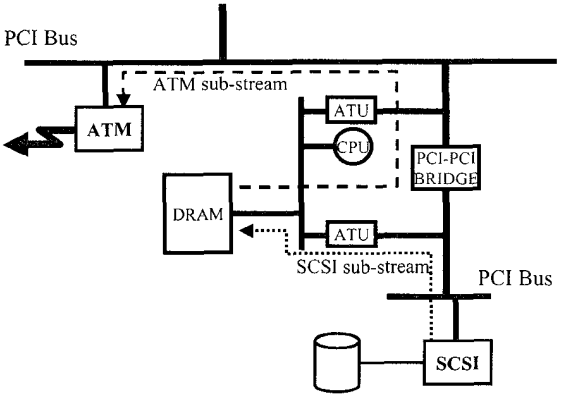

Figure 4: Proposed I/O subsystem using a single i960RP®

Performance figures were calculated based on data streaming over the system buses. These figures incorporate the effects of interrupt latencies but do not include the effects of the operating system on

performance. We have assumed that all PCI transfers will not be broken into multiple transactions. This assumption will not hold for high streaming scenarios, as the ATU queues are of insufficient size for media transfer, especially the SCSI traffic.

Table 1 shows these performance figures. It illustrates the scale of traffic over the system buses and their percentage utilisation.

Table 1: Bus Utilisation

No. of Streams	Host CPU Bus(%)	Primary PCI Bus (%)	i960RP Bus (%)	Secondary PCI Bus (%)
1	0.056	0.7	1.1	0.5
10	0.5	7.1	11.4	4.7
50	2.3	35.3	57.1	23.3
60	2.8	42.3	68.5	27.9
75	3.5	52.9	85.6	34.9
90	4.1	63.5	102.7	41.9

The chart in Figure 5 shows the comparative performances of the system using a single I/O processor with the original target architecture, based on worst-case bus traffic. From this comparison it must be noted that the streaming affects the software running on the original target architecture, whereas the streaming on the i960RP® only affects the SCSI software drivers.

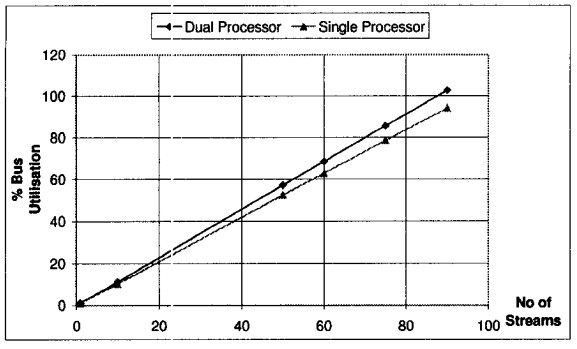

Figure 5: Performances with a single I/O processor

With the proposed design (using a single I/O processor), the host CPU has recovered its access to main memory, previously lost to the stream, enabling it to manage more streams. However, each stream on the PCI bus has become less efficient, due to the i960RP® ATU delayed PCI transactions.

A new bottleneck has appeared on the i960RP® local bus. For ninety streams, the 80960 local bus would be over-loaded when streaming data to and from i960RP® memory, whereas the PCI buses are under-utilised. This is due to the two 32-bit, 33MHz PCI buses trying to access a single 32-bit, 33MHz local bus.

This clearly shows that the proposed architecture would not be a large improvement over the target system. This analysis does not contain any inter-processor communication, which would be necessary for communication between the operating system and the SCSI driver. This additional overhead would further reduce the performance of the proposed architecture.

The scalability of this architecture can be achieved by introducing a PCI-to-PCI bridge to isolate the I/O subsystem. This obviously incurs an added cost, but allows multiple i960RP® devices to be attached to the system for added stream capability.

3.3 Dual I/O processor

The single i960RP® device used in the previous design, removed the stream from the main memory, but created another bottleneck at its own memory. The i960RP® could not run both I/O software drivers due to the interrupt problem stated earlier, therefore the ATM driver was operated from the host. One solution to this I/O driver problem could be to utilise two I/O processors, one for each of the I/O devices. The drivers could reside in their respective i960RP®'s, whilst the memory space of one I/O processor could contain the dual buffering scheme. Whilst this would remove the drivers and their interrupts from the host CPU, their would still be the i960RP® memory bottleneck.

An improvement to this design would be an alternating dual buffering scheme, whereby the buffers would be equally split between the memory spaces of the two I/O processors. Operation for a single stream would be as shown in figure 6 and Figure 7.

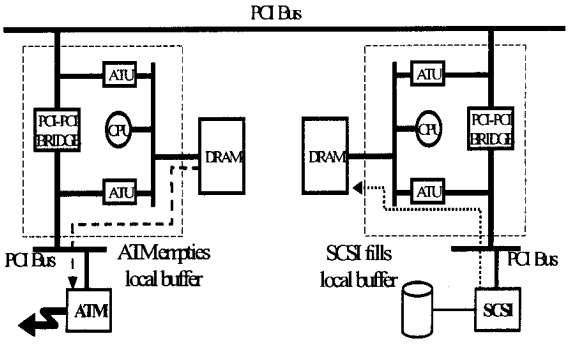

figure 6: Proposed I/O subsystem with two i960RP®

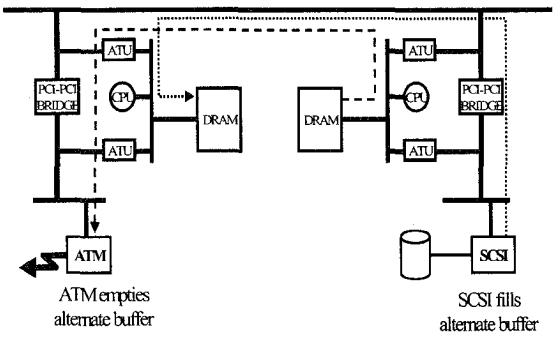

Figure 7: Alternate substreams

Table 1 shows the bus utilisation in cycles per second per stream, for all of the components in the dual i960RP® I/O sub-system.

Table 2: Bus utilisation for a single stream (cycles/sec/stream)

	Sec PCI (ATM) bus	i960 local bus (ATM)	Primary PCI bus	i960 local bus (SCSI)	Secondary PCI SCSI bus
Streaming to local i960	232699	237739	0	153565	153498
Streaming to alternate	232699	170169	373070	221135	153498
Mean streaming	232699	203954	186535	187350	153498

With multiple concurrent streams, the mean value will be the important figure, and as can be seen from Table 2 the sub-streams have been more closely balanced around the system buses. The alternating buffer scheme has removed the I/O processor memory bottleneck, with the most activity being on the secondary PCI bus to which the SCSI adaptor is attached. Plotting this data onto a graph (Figure 8) illustrates the comparative performance between the dual i960RP® design and the initial target architecture.

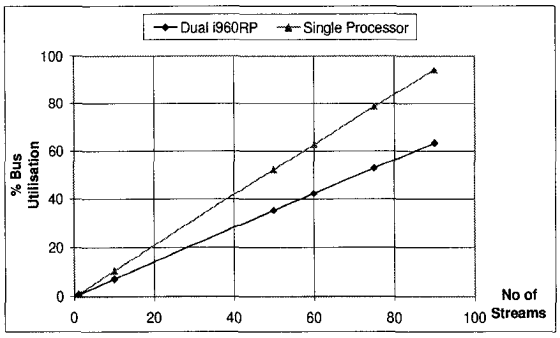

Figure 8: Comparative performances

It can be clearly seen that this design has reduced maximum bus utilisation by 33%, but at the expense of increased complexity, and cost. Again scalability will only be achieved at the cost of an additional PCI-to-PCI bridge.

4. Conclusion

This paper has focused on the design of an I/O subsystem for a continuous media server. Several improved architectures have been proposed and their performances evaluated. All the proposed architectures were designed using an existing device, namely the Intel i960RP® processor.

The utilisation of the single i960RP® I/O processor solved the main memory bottleneck problem, but created a new bottleneck in i960RP® memory. This has highlighted the requirement for a streaming memory bandwidth twice that of the PCI bus.

The twin i960RP® proposed I/O subsystem utilising an alternating dual buffer arrangement, removed this bottleneck but at the expense of scalability, complexity, and cost.

This investigation clearly shows the need for an integrated I/O processor, optimised for continuous media. Such a processor would incorporate the following characteristics.

a) Two separate subordinate PCI buses for the I/O devices to isolate the sub-streams;
b) Memory bandwidth twice that of a single PCI bus;
c) Larger PCI-memory buffer queues, optimised for the transmission of media data;
d) Low interrupt latency to reduce the time taken to process streams;
e) High scalability so that multiple devices can be attached to the primary PCI bus to increase the number of streams.

Figure 9 shows the system's architecture using such a hypothetical I/O processor. On-going research are investigating the feasibility and characteristics of this architecture.

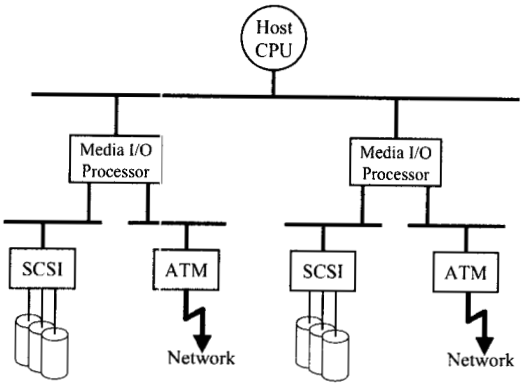

Figure 9 - Scalable Server Architecture utilising Media I/O processors

5. References

[1] Gemmell, D. J., Vin, H. M., Kandlur, D. D., Venkat Rangan, P., and Rowe, L. (1995). Multimedia Storage Servers: A Tutorial and Survey. **IEEE Computer**, 28 (5), pp. 40-49.

[2] Shenoy, P., Goyal, P., and Vin, H. M. (1995). Issues in Multimedia Server Design. **ACM Computing Surveys**, 27 (4), pp. 636-639.

[3] Lougher, P., & Shepherd, D.(1993). The design of a Storage Server for Continuous Media. **The Computer Journal**, 36 (1), pp. 32-42.

[4] Rao, S., Vin, H. M., and Tarafdar, A. (1996). Comparative Evaluation of Server-push and Client-pull Architectures for Multimedia Servers. In **Proceedings of 6th Networks and Operating Systems Support for Digital Video and Audio**, April 1996.

[5] Gillespie, B. (1996). PCI Intelligent I/O Design for High Performance Servers. Intel® white paper.