Handling Parallel Processing in Multimedia Systems

Document Number TR 29.2084

Andrew P. Citron
George Samaras

IBM Micro Electronics
Dept. DD8a
Research Triangle Park, N.C.

ABSTRACT

This paper introduces a way of capturing and processing data in a multimedia real-time environment of parallel processes, where the data must be processed quickly before it becomes obsolete. It presents a mechanism that achieves this performance requirement by avoiding the overhead of locking or buffering while permitting concurrent access to shared data. Its novelty is based on the fact that in a multimedia environment, data loss isn't catastrophic, but it must be recognized when data loss occurs. Thus, this approach effectively relaxes concurrency control by permitting the processing component to recognize whether captured data is obsolete prior to processing. If the data is obsolete, it is discarded. This is useful and acceptable in a multimedia environment where data to be processed has to be complete and not corrupted.

ITIRC KEYWORDS

Multimedia

Telephony

Parallel Processing

Bakery Algorithm

Processor Scheduling

Mwave

Digital Signal Processing

DSP

ABOUT THE AUTHORS

Andrew Citron is part of the Mwave FAX and TAM PC programming team in Raleigh. Other publications include BYTE 1984 (on software review methodology), 1993 IEEE Conference on data engineering (two phase commit optimizations), 1991 and 1993 High Performance Transaction Systems workshop (APPC and LU 6.2). He also holds 2 software patents (one on using switched telephone lines for computer communications, the other on a two-phase commit optimization). Prior to joining team Mwave, Andy was the lead architect with the SNA APPC group (also known as LU 6.2). He can be reached at CITRON@.VNET.IBM.COM

George Samaras has worked in the IBM SNA APPC group. He is currently on leave from IBM and is a faculty member at the University of Cyprus.

CONTENTS

TECHNICAL DETAILS

INTRODUCTION

In a computing environment where there is more that one processor, there must be a means of synchronizing access to shared data. Locking mechanisms [i.e., 1], double buffering, the bakery algorithm [2] or other shared scheduling algorithms [4,5] provide a synchronization mechanism for traditional computing systems. Multimedia has different performance characteristics. Key characteristics are that information must be available when needed, but data that is not consumed in a timely fashion can simply be discarded. Locking mechanisms, double buffering, and the bakery algorithm address the problem of data being written as it is being read, by forbidding more than one process simultaneous access to the shared memory. However, these algorithms cause processes to block. This is not necessary, or acceptable in many multimedia applications.

The specific problem that led us to come up with the solution discussed in this paper occurs in various multimedia applications. The solution was invented for a particular application to telephony where one processor received information, and moved it to a storage buffer that was accessed by another processor. When the information is completely written to the buffer, another processor is sent an interrupt that indicates the data is present. Normally, the interrupted processor would process the data immediately. However, in the world of parallel processors, or even parallel processes on the same processor, there are timing considerations. The specific problem arises if the interrupted processor does not get around to processing the data, and more data arrives. In this case, the new data takes precedence over the old data. Hence, a mechanism is needed so the new data can safely be written over the old data without any possibility that the other processor is reading the old, and possibly partially overwritten, data without realizing it (when identified the corrupted or old data is discarded). This paper describes such a mechanism. The presented algorithm has already been implemented and shipped in IBM's Mwave [1] Telephone Answering Machine (TAM) offering [3].

APPLICATION OF THE PROBLEM TO TELEPHONY

Consider the case where a personal computer (PC) is acting as a telephone answer machine (TAM). The PC contains a digital signal processor (DSP) that is connected to the telephone line. The TAM device driver and DSP code deliver caller ID information from the DSP to the PC. The DSP captures the caller ID from the phoneline. However, the DSP can not be

---

[1] A separate processor, designed to handle real time digital signal processing such as audio and telephony. Mwave is a trademark of IBM Corporation.

exactly sure when the PC will get around to reading the information into the PC memory. The PC is controlled by various multi- tasking operating systems. Since the PC TAM driver is not dispatched at a deterministic interval, it can't be certain that the DSP won't be re-writing the caller ID information at the same time the PC is reading it. Failing to process the caller ID data before the next one comes in (i.e., another phone call arrives) renders the old data obsolete. What we need here is a mechanism that makes sure that the PC knows when the caller ID information in the DSP is complete and uncorrupted.

The key characteristics of such multimedia applications are:

a) the data has to be processed in a timely fashion, otherwise it is obsolete b) data loss isn't catastrophic

Furthermore, these multimedia systems do have some 'real-time' type constraints. That is, the hardware has got to be fast enough to handle the task the majority of the time. Earlier solutions to this type of problem came from the operating system or transaction processing arena. The processor-speed requirement is not something earlier solutions could depend upon.

For caller ID processing, the PC actually has about six seconds to read the data before it can possibly be obsolete. So in most cases, even the slowest PC can get there most of the time. One situation where the caller ID data can become obsolete is when you get one call, then the caller hangs up before the call is actually answered, and then another call comes in. That typically takes more than six seconds. Another related situation where the data can become out-of-date is when the TAM application is reading the caller ID data as a new call arrives.

These characteristics led to the following algorithm that satisfies the demands of multimedia applications by relaxing the traditional concurrency control mechanism. It accomplishes this by making sure that the PC knows whether the received data is complete or corrupted. If complete, then it is properly processed while if obsolete, it is discarded.

THE MWAVE CALLER ID ALGORITHM

Terms:

One processor will be called the data-capturer. The other processor will be called the data-processor.

Data Structures:

There are two important shared storage locations: the data identifier, and the data buffer. Both the data-capturer and the data-processor have access to these storage locations. Only the data capturer writes to those locations. The data processor reads them.

The data-capturer executes the following logic:

The data-identifier location is initialized to zero. A non-shared variable, next-ID, is set to one. After this, the data-capturer loops (or has an interrupt mechanism) to see if information is arriving. When information arrives, the data-capturer writes zero to the data-identifier storage. Zero indicates that the data stored in the data buffer is not currently valid. The data-capturer then moves the data, as it is captured, to the data buffer. When all the data is captured, the value in next-ID is moved to the data-identifier, and the data processor is informed that data is available for it. The next-ID is incremented, so that the next time data arrives it is given the next sequential id. The data-capturer then loops, waiting for the next data to arrive.

The mechanism that the data-capturer uses to inform the data processor can be an interrupt, or any other inter-process or interprocessor communication method. The method used is inconsequential to this algorithm. What is important is that there are parallel processes accessing the same storage locations.

The data-processor's logic is as follows:

The data-processor loops (or waits for an interrupt), waiting to be told that data is available. When data is available, the data-processor firsts read the data identifier, and then saves it. Then it reads the data buffer, and saves it. Then it re-reads the data identifier. If the re-read data identifier is the same as it was when it was first read, and the data identifier isn't zero, then the data has successfully been saved. It can now be processed. However, if the re-read data-identifier is not the same as it was when first read, the data in the buffer is obsolete, and possibly partially overwritten. The data is therefore obsolete, and can be discarded. The data processor then loops again waiting to be told that data is available. Figure 1 presents a pseudo code of the algorithm.

DISCUSSION

This simple mechanism allows parallel processes to access the same volatile storage, and efficiently and safely determine when data is successfully and completely captured, or when the data is obsolete, before it gets processed. A single data buffer is all that is needed since it is permissible to discard data that is obsolete. This mechanism guards against the case where one processor is reading data, as the other processor is overwriting the data with

newer, more up-to-date information. Using this logic, there is no case where obsolete, and possibly overwritten data is mistaken as being current, and correct data.

Theoretically, this algorithm is vulnerable to the "consumer starvation problem". The consumer is susceptible to race conditions: depending on the timing, it is theoretically possible for the producer (the data-capturer) and the consumer (the data-processor) to get into tandem. When the producer and the consumer are in tandem the producer can invalidate everything that the consumer reads, hence the consumer is never able to read any data. However, as mentioned previously, these multimedia systems do have some 'real-time' type constraints. In telephony applications, the PC actually has, by the nature of the application, typically at least six seconds to read the data from the buffer. That's more than enough time even for the slowest machine. Remember, the only time the caller-id data is obsolete is if you get one call, then the caller hangs up, and then another call comes in. However, the PC is not deterministically dispatched since its processing schedule is dictated by the whims of a multi-tasking operating system. On today's PC operating systems the PC driver cannot be assured that it will run in six seconds in every situation.

****

The data-capturer pseudo code:

```
Data-identifier location is initialized to zero.
Next-ID, is set to 1.
While data capturing still needed
  If information has arrived,
     Write 0 to the data-identifier storage.
     Move the data, as it is captured, to the data buffer.
     Set data-identifier to the value in next-ID.
     Inform data-processor that data is available for it.
     Increment next-ID.
  Endif
End While
End of the data-capturer
```

The data-processor pseudo code:

```
While data processing still needed
   Wait to be told that data is available.
   Read the data-identifier, and save it.
   Read the data buffer, and save it.
   Re-read the data identifier.

   If the re-read data-identifier is the same as it was when it was
    first read, and the data identifier isn't 0, then
```

```
      The data has successfully been saved.
       It can now be processed.
   Else
      The data in the buffer is obsolete, and
       possibly partially overwritten.  The data
       is therefore obsolete, and can be
       discarded.
   Endif
End While
End of the data-processor
```

Figure 1: Pseudo code of the Mwave Caller ID algorithm

****

For video processing this starvation issue might be more serious. In that case, double buffering might be necessary. However video data is very big, so the extra buffer might be costly. PCs are getting faster, and the ones involved with video processing have minimum speed requirements. So it is still possible that the Mwave Caller ID algorithm might work for video (or audio). The only way to avoid losing new data is to 'double buffer' it. Then you have the expense of the second buffer. Its a trade off: fast hardware vs. slower hardware and extra storage.

RELATED WORK

The producer/consumer problem is not new, its been around for some time and variations of the problem occurred in different disciplines. Similar mechanisms to the one proposed in this paper have been used to implement optimistic concurrency control or thread scheduling [i.,e., 4, 5] for example, where a data item, bit, or an optimistic lock is used to detect the preservation of invariants. With optimistic locking both transactions make updates assuming no changes. At commit time, during the validation phase, if the updates violate serializability, then one (or both) of the transactions back out. The data-identifier used in our algorithm, though, is a data item that is used to concurrently detect the invariant of the data buffer itself. Furthermore, with this multimedia technique, first of all, only one side (the producer) is making updates, the second (the consumer) is just reading. If the consumer reads something it determines is out of date, nothing backs out, no transaction is actually aborted. Rather the consumer simply reads again and gets more up to date information. Our schema can be viewed as one that is biased towards the writer. In case of a conflict the reader aborts and is reactivated. The thread scheduling approach [4] (i.e., the single bit approach) is again different from ours in that it does not allow data to be lost (overwritten) and is used to coordinate two consumers. Allowing pure concurrent access to the shared buffer, as in the Mwave Caller ID algorithm, renders the one-bit approach insufficient.

This paper describes such a producer/consumer mechanism tuned towards multimedia applications. The basic idea is not completely new (producer/consumer), the motivation and application of this idea is, though, quite new.

CONCLUSIONS

Multimedia has different performance characteristics. Key characteristics are that information must be available when needed, but data that is not consumed in a timely fashion can simply be discarded. Taking that into consideration this paper presented an efficient algorithm for processing shared data by relaxing the traditional ways of controlling buffer concurrency. This algorithm has already been implemented and shipped in IBM's Mwave Telephone Answering Machine (TAM) offering.

ACKNOWLEDGEMENTS

Acknowledgments: The authors wish to thank Bob Schule, Allen Mitchell, Ali Sadri, and Bill McCormick for working out the details of this algorithm and finding problems in other similar solutions. We also like to thank Panos Chrysanthis for his helpful feedback on this work.

REFERENCES

[1]Dennis, P. J., E. C. Van Horn. Programming Semantics for Multiprogrammed Computations. Comm. of ACM, vol 9, no. 10. pp 143-155, Mar., 1966

[2]Lamport, L. A new solution to Dijkstra's Concurrent Programming Problem. Comm. of ACM, vol 17, no. 8. pp 453-455, Aug., 1974

[3]Mwave Application Programmer's Guide, IBM Microelectronics, Document Distribution Center, PO Box 7932, Mt. Prospect, IL 60056-7932.

[4]H.V. Jagadish and O. Shmueli. A Proclamation-Based Model for Cooperating Transactions. Proceedings of the 18th International Conference on Very Large Databases, pp. 265-276, 1992

[5]K. DiBella, K. Ramamritham, P. K. Chrysanthis, S. Raghuram. Scheduling Algorithms and their Performance on Shared Memory Multiprocessors, The Proceedings of the 5th ISMM Conference on Parallel and Distributed Computing and Systems, pp. 133-139, Pittsburgh, PA, October 1992.