Applications of PRAMs in Telecommunications*

R. Drefenstedt, J. Keller and W. J. Paul

Fachbereich 14 Informatik, Universität des Saarlandes
Postfach 151150, 66041 Saarbrücken, Germany

Abstract

ATM seems to be the most promising technique to obtain a B–ISDN with a universal interface. In order to turn this standard into an operating network, the design of high-bandwidth ATM switching nodes is of crucial importance. The need to achieve high link utilization, traffic smoothing and skew reduction sometimes requires extensive output buffering on some links. This either wastes resources if one uses private buffers for each link or decreases performance if one uses a single shared buffer for all links. We show how to use a shared memory with a bandwidth that scales with the number of links to reduce total buffer size without decreasing throughput. The shared memory also provides the routing function of the switching node.

Keyword Codes: C.1.2; C.2.1
Keywords: Processor Architectures, Multiprocessors; Computer–Communication Networks, Network Architecture and Design, ATM (Asynchronous Transfer Mode)

1. INTRODUCTION

Telecommunication networks have become an important factor in connecting the world, providing services with widely varying bandwidth requirements ranging from electronic mail and file transfer programs to multi media systems. Increasing traffic volume and throughput requirements demand new network technologies and protocols. The *Asynchronous Transfer Mode* (ATM) [1] seems to be one of the most promising standards for future digital communication networks. Of crucial importance for ATM networks are ATM switching nodes serving high bandwidth links with low latency.

In order to operate the network in a cost-efficient way especially over long distances, link utilization has to be close to 100 %. Wasting available bandwidth is hurting especially when a switching node serves expensive long distance lines, e.g. transatlantic satellite connections. In order to avoid wasting bandwidth of a link, there should always be data that are ready to be transmitted on that link. Hence there has to be a large output queue for each link. Simulations show that high utilization can lead to queue sizes up to several Megabytes [2].

As only a few of theses queues will be filled at any time, providing each link with a large output buffer is a waste of resources [3]. Providing a shared memory that contains output buffers for all links reduces memory requirements but imposes strong requirements

*This research was partially supported by DFG through SFB 124, TP D4.

on the memory bandwidth, which must be proportional to the number of links. Physical shared memory cannot provide this bandwidth.

Scalable shared memory has been extensively studied in computer science in the setting of parallel computers with a global address space (PRAMs). We show that the results obtained there can be used to implement a shared memory with the required bandwidth in an ATM switching node and present data structures necessary to administrate multiple buffers in the shared memory.

In Section 2, we briefly review some basics of ATM networks and ATM switching nodes. In Section 3, we describe the design of a scalable shared memory. In section 4, we show how this design can be used in an ATM switching node and we present data structures for buffer management. In Section 5, we discuss our solutions and hint future work.

2. RELEVANT ATM BASICS

We will restrict our description of ATM networks to the level of detail necessary to understand our design. For a detailed description, the reader is referred to [1].

ATM networks are packet switched. A packet, here called a *cell*, consists of 48 data bytes and 5 header bytes, that contain routing information and other control information. Each link carries several *virtual paths*, each virtual path several *virtual channels*. Routing information consists of a *virtual path identifier* VPI and a *virtual channel identifier* VCI, that are 16 bits and 12 bits wide, respectively (for NNI). When a connection is opened, virtual paths and channels are installed in a *signaling phase*. Each channel is assigned a certain bandwidth during signaling, the total bandwidth of all active channels must not be more than the link bandwidth, which is 155 MBit/s for the currently used STM-1 transport standard. Signaling is not standardized yet, we will not go into it.

Path and channel identifiers can be different on any link on a cell's way from source to destination. Each switching node hence maintains *header translation tables* (HTT's) for all links, that are indexed by the current VPI and VCI and give the link number from where the cell is to be sent to the next switching node, and the new VPI and VCI number.

An ATM switching node with n links consists of the parts shown in Fig. 1 and works as follows. Incoming cells are received in the *input processing unit* (IPU) and header translation is done immediately. Then the cells are routed to the appropriate link, where they are queued and sent by the *output processing unit* (OPU). Ideally, the router is a complete bipartite graph, so no delays or contentions occur during routing. In practice, the router is a non–blocking network, so that only output queueing is necessary. The central control unit serves for signaling, i.e. opening and closing of connections, traffic control, i.e. checking that channels do not exceed their assigned bandwidth, and accounting.

One difficult part to design are the output buffers. They must be able to store and output cells both with a bandwidth of 155 MBit/s, i.e. one cell every 2.8 μs. This works well if cells from different channels arrive one by one. Unfortunately, channels are not synchronized, so cells from all incoming links might arrive at an output buffer at the same time, with a time gap of $2.8 \cdot n$ μs to the next arrival of a cell. This puts some load on buffer bandwidth, but restricts buffer length to n, so that solutions can be found for this special case. Note that a smaller time gap would indicate a bandwidth violation of the channels on that output link.

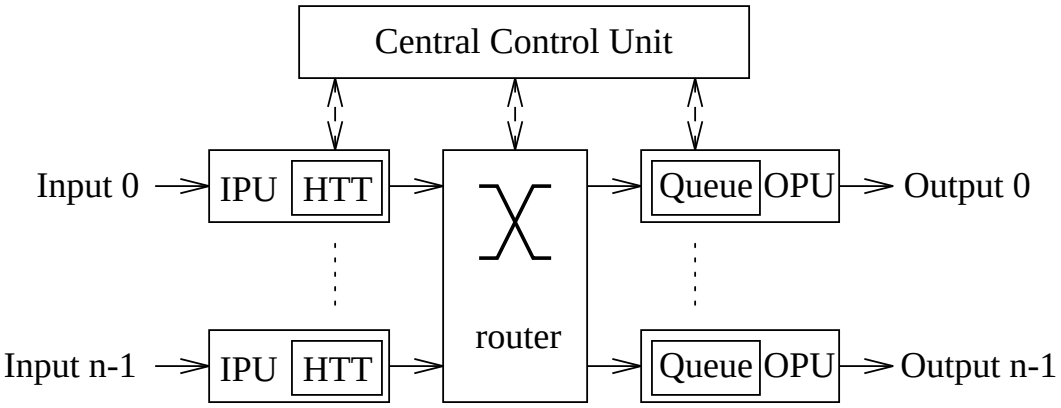

Figure 1. ATM switching node

Traffic theory and simulations show that this method achieves link utilization of 85 % for *continuous bit rate traffic* (CBR traffic) [4], where each channel sends a continuous bit stream and arrival times are bernoulli distributed. Links cannot be fully utilized because queues are too short to feed the link in times when no cell arrives. However, this model does not reflect expected future traffic characteristics, where bursts of cells with succeeding idle times are likely to occur. For traffic of this kind, link utilization will be much worse. This is a waste especially in case of links that are expensive to run, i.e. long distance satellite links. Therefore, ATM regulations propose to study *statistical multiplexing* [5] to increase utilization by providing longer queues. Channels are assigned *peak* and *mean* bit rates. Thus channels are allowed to use more than their normal bandwidth for short periods of time. This is still an active research area, so there are no final results and regulations, but statistical multiplexing seems to be very useful in increasing utilization for VBR traffic [4]. However, the use of statical multiplexing increases queue size up to 6 MByte [2].

Even if contention problems caused by multiple cells arriving simultaneously could be solved, providing an output queue of 6 MByte for every link is a waste of resources, as the sum of the required queue lengths will be much smaller than $6 \cdot n$ MByte. This holds because the total bandwidth of all incoming links is not increased, so only a few queues could grow large at the same time.

A solution would be to use a memory with an address space shared among all links, so that all queues could be implemented in that memory. The size of the memory would be not more than the sum of the queue sizes, so there would be no waste of memory. However, the bandwidth of the memory has to be n times as large as the bandwidth of a queue for one link. It follows that a physical shared memory cannot be used for large n. Furthermore, the problem of multiple cells arriving simultaneously at one queue has not been solved.

In Section 3, we briefly describe the design of a memory system with n ports and with a bandwidth proportional to n. In Section 4, we present data structures for that memory system that solve the contention problem.

3. SHARED MEMORY DESIGN

Parallel computers more and more tend to provide the user with a single shared address space to simplify parallel programming. A popular model from theoretical computer science, where processors are working synchronously and where access to global memory takes constant time independent of machine size and memory access pattern, is called *parallel random access machine* or PRAM [6].

As an implementation of a PRAM with a physical shared memory is not feasible with current technology, several approaches have been proposed to emulate a PRAM on a processor network, e.g. [7–9], see [10] for a survey. A processor network consists of p processors and memory modules, connected by an interconnection network. Emulation of a PRAM consists of distributing the address space among the memory modules, and emulating access to global memory by packet routing requests from processors to memory and back.

Distribution of the global address space has to guarantee (with high probability) that the number of requests destined for one module, the *module congestion*, will not be too large, no matter what the access pattern might look like. Otherwise, performance will be very poor. Note that in this description we omit concurrent access, where multiple processors try to read or write the same memory cell at the same step. Distribution is done by using universal hashing. Consider a class of hash functions, where each function distributes all but a few access patterns reasonably well among the memory modules. If a particular function is chosen randomly, then the application will, with very high probability, not have an access pattern that is distributed "badly" by the chosen function. Polynomials of degree $O(\log p)$ provably restrict module congestion to $O(\log p)$ with high probability [8]. Linear functions have shown sufficient results for practical purpose [11].

Access to the shared memory now consists of sending a request from a processor to the appropriate memory module across the network, accessing the requested cell there, and sending back an answer in case of a READ request. If a constant degree network is used, this needs time $\Omega(\log p)$. Randomized routing algorithms are able to route a $\log p$ relation in time $O(\log p)$ [9], which is optimal on a constant degree network. A $\log p$ relation is defined as $\log p$ requests per processor, at most $\log p$ requests are destined to the same module.

With the previous techniques, a memory system can be designed where p accesses can be completed in time $O(\log p)$. In order to provide a bandwidth proportional to p, accesses are overlapped. If each processor sends $O(\log p)$ requests, module congestion will still be $O(\log p)$ with high probability. The requests then form a $\log p$ relation that can be routed in time $O(\log p)$. Hence $O(p \log p)$ requests can be completed in time $O(\log p)$, leading to $O(p)$ requests completed per time unit.

As with many new approaches, it was first unclear whether the emulations were practical in terms of constant factors, or of pure theoretical interest. Ranade's emulation [9] was re-engineered and found to be practical [12]. A prototype of the resulting architecture with $p = 128$ processors, called the SB–PRAM, is currently being constructed [13]. The prototype does not use state-of-the-art technology. Memory access latency in this prototype is 3.8 μs, all data paths are 32 bits wide, a request can be sent every 140 ns. We will use these data in the following calculations.

4. SHARED BUFFERS IN ATM SWITCHES

We consider an ATM switching node with n links, where input and output processing units are connected to a shared memory system as described in Section 3. We hold n^2 FIFO queues in the shared memory, where queue $Q_{i,j}$, $0 \leq i,j < n$, serves to queue ATM cells that arrived at input link i and will be sent at output link j. Each queue $Q_{i,j}$ consists of a piece of global memory, and pointers $h_{i,j}$ and $t_{i,j}$ to its head and its tail. As input processing units are only writing to the queues and output processing units are only reading from the queues, the pointers can be held locally. Input processing unit i holds pointers $t_{i,j}$ for $0 \leq j < n$, and output processing unit j holds pointers $h_{i,j}$ for $0 \leq i < n$.

Providing n^2 queues instead of n solves the contention problem, as at most one cell will be written to a queue at any time. One might think of this construction as a kind of crossbar with buffers. Note also that the shared memory system provides the routing function. Hence the router is not needed anymore and is replaced by the network in the shared memory emulation. The changed switching node is shown in Fig. 2.

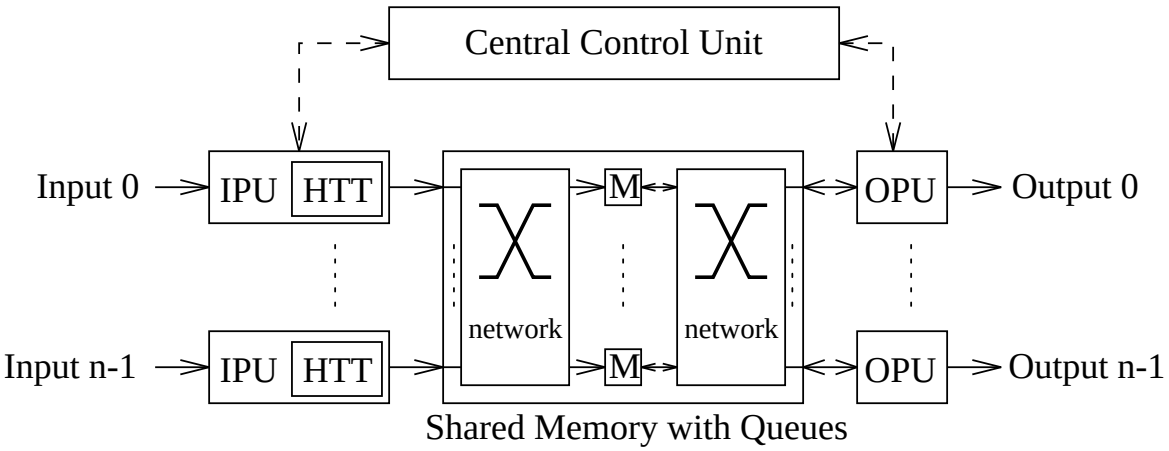

Figure 2. ATM switching node with shared memory

4.1. Queue management

An input processing unit receives a cell, performs header translation and then writes the cell with updated VPI and VCI into the appropriate queue. This requires 14 writes, as one write instruction sends four bytes, and ATM cells consist of 53 bytes. The 14 writes take time $14 \cdot 140\,ns = 1.96\,\mu s$. As every 2.8 μs a cell can arrive, this means that we need two different networks for writing to and reading from the shared memory. The network to write cells does not need facilities to send answers back to the input processing units.

An output processing unit j reads cells from queues $Q_{0,j}, \ldots, Q_{n-1,j}$ in a round-robin fashion. It has to make sure that it does not waste time with empty queues, and that it does not read cells that are not yet completely written by an input processing unit (recall that a cell is written in 14 writes).

In order to find the next non-empty queue very fast, we need some special hardware,

which works as follows. As soon as an input processing unit i has started writing an ATM cell to a queue $Q_{i,j}$ it sends a message to output processing unit j across the network that emulates the shared memory. This message causes a counter $c_{i,j}$ to be increased. The counter $c_{i,j}$ is local to output processing unit j. If an output processing unit j has completely read an ATM cell from queue $Q_{i,j}$ it decreases counter $c_{i,j}$. Finding the next non-empty queue now consists of finding the next counter with value greater than zero. As all counters are local to output processing unit j, this can be done very fast with dedicated parallel prefix circuitry. This circuit works in a manner similar to finding the most (or least) significant bit in a word that is one, an instruction that is available in several microprocessors today.

In order to forbid reading a queue entry before it is written completely, we extend each memory cell by one bit that is used as a tag. Initially the tag is zero. If a word is written to a cell, the tag is set to one. If the cell is read, the tag is set to zero again. This requires only minor additional hardware on the memory boards. An output processing unit that is reading from a queue now checks whether the tag is one. If not, it will poll on this memory cell. This can happen if an output processing unit starts reading out an ATM cell as soon as its counter is increased, and one of the words sent by the input processing unit is delayed in the network that emulates the shared memory.

If an input processing unit would send its message to an output processing unit *after* it completed writing an ATM cell to a queue, one would not have to take measures in order to prevent reading before writing. However, this would also increase delay in the queues. Polling on a memory cell normally does usually not waste any link bandwidth, as reading out an ATM cell takes time $14 \cdot 140\,ns = 1.96\,\mu s$ without polling, whereas an ATM cell can be sent every 2.8 μs.

4.2. Memory management

The shared address space is partitioned logically into pages of size s. We assume s and the total memory size to be powers of two. Note that these logical pages are independent of the distribution of memory among memory modules. The queues are implemented with linked lists of pages. New pages for a queue $Q_{i,j}$ are requested by input processing unit i, pages from a queue $Q_{i,j}$ that are completely read are returned by output processing unit j. Page management itself (granting of requested pages and re-use of returned pages) is done by the central control unit (CCU). The CCU keeps a linked list of pages that are currently not in use. To satisfy a request, the CCU takes the page from the beginning of the linked list, returned pages are appended at the end of the list. We assume that input and output processing units are connected to the CCU by a bus.

As the CCU can only handle one request per time unit, there may be some delay d between the time when an input processing unit requests a page and the time when the request is granted. Therefore, an input processing unit already requests a new page when it starts to write on the last page it has. Let $t(s)$ be the time to write a page of size s completely. Then we need to choose s such that $t(s) > d$.

To get an assessment of the page size s, let us assume a CCU with a processor that executes an instruction every 40 ns, and that granting a request takes 100 instructions and thus 4 μs. At most n input processing units can request a page simultaneously. Hence $d < 4 \cdot n\,\mu s$. If s is measured in bytes and writing four bytes takes at least 140 ns, then

$t(s) \geq 35 \cdot s\,ns$. If we choose s such that $35 \cdot s\,ns > 4 \cdot n\,\mu s$, then also $t(s) > d$. The inequality holds for $s > (800/7) \cdot n$. If we work with a switching node with $n = 128$ links, then $s > 14629$, hence we could choose $s = 16$ kBytes or 4 kWords, where a word consists of four bytes. If total memory size is 1 GByte, there will be 66536 pages in total, and memory management will not consume too much space in the CCU.

5. CONCLUSIONS

We presented very simple data structures that allow to use a shared memory with scalable bandwidth both for routing and output queuing in an ATM switching node. Clearly, the proposed design does not include all details. We did not consider the topic of providing multiple qualities of service with different queues. However, implementing $c \cdot n^2$ queues to provide c different qualities of service seems to be trivial, so the extension can be restricted to find parallel algorithms for selection of cells from $c \cdot n$ queues instead of n according to the quality rules. We also did not consider the problem of supervising, checking for bandwidth violations and accounting. It is possible to perform these tasks locally at the input processing units, without taking any advantage of the shared address space.

The network of the SB–PRAM prevents cells in the network from overtaking each other, but does not represent state-of-the-art technology. Switching over to commercially available routing chips could decrease memory access latency substantially, but could require resequencing of cells.

It might be useful to use combining networks. This could provide an elegant solution to the broadcasting problem: an IPU writes to a broadcasting queue in shared memory and all OPU's concurrently read from that queue.

Last, we would like to hint a possible future application of shared memory in ATM switching nodes. Currently, header translation tables have less than 4000 entries each and thus can be maintained in local associative memories or their emulations. Stepping over to future networks, link bandwidth could increase by two order of magnitudes. Then associative memories will be too small to hold the translation tables. Also the translation tables could vary in size. Then it might be appropriate to provide a shared memory to hold all translation tables without wasting resources [14].

ACKNOWLEDGEMENTS

For discussions the authors would like to thank Dr. Thurner and Dr. Wallmeier from SIEMENS AG, Munich.

REFERENCES

1. M. DePrycker, Asynchronous Transfer Mode: Solution for Broadband ISDN, 2nd Edition, Ellis Horwood, London, 1993.
2. E. Wallmeier, Private Communication, Nov. 1993.
3. M. A. Henrison et. al., Switching Network Architecture for ATM based Broadband Communications. In Proc. XII Internat. Switching Symposium, 1990.

4. E. Wallmeier and C. M. Hauber, Blocking probabilities in ATM pipes controlled by a connection acceptance algorithm based on mean and peak bit rates. In J. W. Cohen and C. D. Pack (eds.), Queueing, Performance and Control in ATM (ITC–13), pp. 137–142. Elsevier Science Publishers, Amsterdam, 1991.
5. CCITT Study Group XVIII, Geneva, CCITT I.371 – Traffic Control and Congestion Control in B–ISDN, Report R–117, 1992.
6. S. Fortune and J. Wyllie, Parallelism in random access machines. In Proc. 10th Symp. on Theory of Computing, pp. 114–118, 1978.
7. A. R. Karlin and E. Upfal, Parallel hashing: An efficient implementation of shared memory. J. Assoc. Comput. Mach., 35(4):876–892, 1988.
8. K. Mehlhorn and U. Vishkin, Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. Acta Inform., 21:339–374, 1984.
9. A. G. Ranade, How to emulate shared memory. J. Comput. System Sci., 42(3):307–326, 1991.
10. L. G. Valiant, General purpose parallel architectures. In Jan van Leeuwen (ed.), Handbook of Theoretical Computer Science, Vol. A, pp. 943–971. Elsevier Science Publishers, Amsterdam, 1990.
11. C. Engelmann and J. Keller, Simulation-based comparison of hash functions for emulated shared memory. In Proc. PARLE '93, Parallel Architectures and Languages Europe, Lecture Notes in Comput. Sci., pp. 1–11. Springer, Berlin, 1993.
12. F. Abolhassan, J. Keller, and W. J. Paul, On the cost–effectiveness of PRAMs. In Proc. 3rd Symp. on Parallel and Distributed Processing, pp. 2–9. IEEE Computer Society Press, Los Alamitos, 1991.
13. F. Abolhassan, R. Drefenstedt, J. Keller, W. J. Paul, and D. Scheerer, On the physical design of PRAMs. Comput. J., 36(8):756–762, 1993.
14. R. Drefenstedt, Shared Memory ATM-Switch. Patent application EP 93310502.5, Filing Date Dec 23, 1993.