Using scalable distributed computers in telecommunications

Jean-Marc Jézéquel

Using Scalable Distributed Computers in Telecommunications

J.-M. JÉZÉQUEL

IRISA/CNRS
Campus de Beaulieu
F-35042 Rennes Cedex, FRANCE
E-mail: jezequel@irisa.fr
Tel: +33 2 99 84 71 92 ; Fax: +33 2 99 38 38 32

Abstract. While it has long been suggested to use parallelism to boost the performances of telecommunication switching equipment, existing solutions do not easily scale up to very high throughput unless dedicated hardware is used. In this paper, we investigate how off-the-shelf distributed computers could still be used along with relevant software architectures to build truly scalable telecommunication switching equipment, taking as an example the implementation of an SMDS (Switched Multi-megabits Data Service) server. We first analyse performance figures of an experimental SMDS server, and deduce the hardware and software architecture that is needed to make it scalable. We then describe a prototype implementation, and discuss performance results. We conclude on the interest and perspectives of this kind of scalable architecture.

1 Introduction

Modern telecommunication networks are built around switching systems (SS) connected through high speed communication lines. Upon entering the network, the data to be transmitted (either data, voice, pictures, video, etc.) is cut in small chunks called *packets*. These packets are directed towards their destinations through a number of switching systems. Depending on whether the path between the source and the destination is computed once for all the packets of a message, or independently for each packet, the network is said to be *connected* (e.g., X25 or ATM) or *connectionless* (e.g., the Internet Protocol, IP).

Depending on the level of the service provided to the user, the computation of routing informations can become a complex task far beyond what an hardwired solution might provide. Thus most highier level SS rely on a combination of dedicated hardware to handle low level processing, plus more and more sophisticated software to handle complex route computations.

Because all traffic must go through SS, their performances can be critical to avoid them becoming bottlenecks in the network. While it has long been suggested to use parallelism to boost the performances of telecommunication switching equipment, existing solutions do not easily scale up to very high throughput and low-delay transmissions unless full custom hardware is used. In this paper, we investigate how off-the-shelf distributed computers could still be used along with relevant software architectures to build fully scalable switching equipment, with the benefit of an

easier portability. As a case study we consider the design and the implementation of a scalable SMDS (Switched Multi-megabits Data Service [1]) server. We first analyse performance figures of an experimental SMDS server, and deduce the hardware and software architectures needed to make it scalable (Section 2). We then describe a prototype implementation (Section 3), and discuss performance results with respect to the perspective of providing Gigabit data flow rates, that is cited as an unlikely reachable limit in [11]. We conclude on the interest and perspectives of this kind of scalable switching architectures.

2 Performance issues in a SMDS server

2.1 The Switched Multi-megabits Data Service

Switched multi-megabits data service (SMDS) [1] is a connectionless, packet-switched data transport service running on top of connected networks such as the Broadband Integrated Service Digital Network (B-ISDN), which is based on the asynchronous transfer mode (ATM).

SMDS was designed to provide high throughput and low-delay transmissions, and to be able to maintain them over a large geographical area. As a result, it can be used to interconnect multiple-node local area network (LANs) and wide area networks (WANs), and provide them with "any-to-any" service (a capability sometimes referred to as the *dial-tone for data*). SMDS relies on an overlay network with non-ATM switching to transfer connectionless messages [11]. This network consists of a set of interconnected *connectionless servers*. Clients who are willing to do connectionless traffic have to access the nearest connectionless server using any available protocol, such as an ATM connection.

2.2 A Reusable Software Architecture

To learn about performance issues in an SMDS server, we have built an experimental SMDS server on a Unix workstation (Sparc SS20). Since the idea was to end up with a parallel implementation on a distributed architecture, we invested some effort to make our experimental implementation generic and machine independent enough as to be able to reuse large parts of it. Thus we used an object oriented analysis and design method followed with an implementation in an object oriented language, Eiffel (this is described as a case study in [9]).

Reusability and portability are not yet established ideas in the telecommunication world where most systems are of a real time nature, and are finely tuned to get the best performances out of a given architecture. However, this is changing because the versatility of the new value added telecommunication services induces huge software development costs that need to be paid off on more than one hardware generation (itself becoming shorter and shorter).

Using an OO approach and an implementation language such as Eiffel greatly facilitates the development of large systems. However, high level features always have a cost. Fortunately this cost stays reasonable, thanks to recent advances in compiling and runtime technologies for OO languages. Experiments show that, with current Eiffel compilers, the execution time for most Eiffel-written applications is usually only 5% to 15% longer than that of an equivalent hand-written C one.

2.3 Performance Analysis

They are a number of studies in the literature identifying bottlenecks in high-performance communication systems, concerning most notably header processing speed, data movement inside the system, execution environment, and interface between the processing system and the physical layer.

Kind of processing	headers processed/s	mean processing time
SNI to ISSI	5025	199 us
ISSI to SNI	8093	124 us
Switching (ISSI to ISSI)	5347	187 us

Table 1. Header processing speed in an SMDS server

Header processing speed The packet header processing speed determines an absolute limit on the global performances of the communication system: the system may not have a throughput per I/O board larger than the maximum packet size divided by the header processing speed. This header processing speed has then to be optimized as much as possible. The introduction of parallelism has been considered at this level, but it does generally not pay off [5, 14] because of the limited intrinsic provision for concurrency in such kind of processing. This tendency is even enforced with recent "light-weight" protocol (XTP, SMDS, etc.) featuring simplified header processing, actually leaving nearly no room for parallelization.

In our SMDS server, the significant figures are the speed of header processing in different contexts: transmission (packet received from an SNI and then injected in the SMDS network), reception (packet coming from an ISSI link and delivered to a SNI), and switching of traffic (from an ISSI link to another one). For these specific measures, we use a specialized version of our SMDS server, where the lower layers are simulated: a transmission only consists in incrementing a counter; and as for receptions, the server is always told that a packet is ready to be read and the reception is simulated (a fixed set of predefined packet is used). The measures consist in performing continual operations on the server and compute their mean durations (which is more realistic than exploring the assembly language listing to add up individual times of machine language instructions on a given path of the header processing).

The performance results presented in Table 1 have been made on Sun SPARC20 workstation. They show that the internal speed of a sequential server is sufficient to reach Gigabit flow rates on a standard processor: the slower operation reaches 3.29 Gb/s with 64k packets.

Execution environment Since these tests involve no network connection, the results are directly proportional to the processing power of the processor. This implies that the execution environment has a major influence on performances. The raw power of each processor actually determines the throughput of the system, and the bandwidth

of the channels connecting the different nodes limits the global data flow. If the software is portable, it can directly benefit from the very fast speed improvement of hardware components.

Data movement inside the system Another important source of performance loss is linked to data movement inside the system [15]. To be efficient, an implementation should avoid to copy packets from memory to memory because of its time cost (DRAM bandwidth does not follow the exponential increasing of processing power). Also, for parallel implementations, data transfers between two separate nodes should be minimized because excessive internal communications can lead to link saturations and even to a global system slowdown. A classical solution [2] is the use of a shared memory where are stored all the packets. The processing nodes would only access the headers and trailers of these packets and would never deal with the data they include.

Interface between the processing system and the physical layer The last bottleneck may be the interface between the processing system and the physical layer [2]: its throughput should be sufficient not to limit the system. To check that, we determine

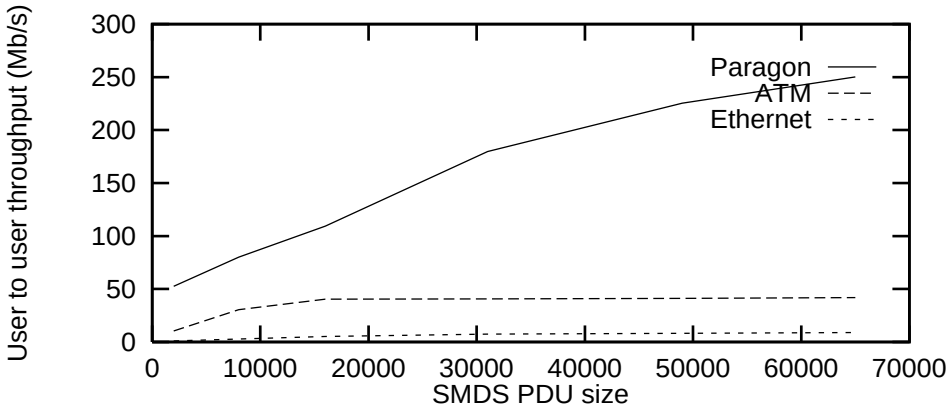

Fig. 1. Maximal data flow rate of a SMDS server

the maximal data flow rate of our SMDS server for a set of representative physical network technologies. We measure a one way, user to user, actual data flow rate through two SMDS servers communicating through a unique link. This measure takes into account the header processing, the operating system and the SAR overheads, and the actual data transmission between the two users. This experiment has been done with a range of packet sizes, allowing us to get information on both the latency and the throughput of the network.

The Ethernet (maximum bandwidth 10 Mb/s) and ATM experiments were led on Sparc workstations, with Fore System SBA-200 SBus interface boards in the later case (maximum bandwidth 150 Mbs/s). To experiment with an higher bandwidth, we also used the internal network of a parallel computer, the Intel Paragon XP/S.

The version of this computer available in our lab. is made of 56 processing nodes, linked by high-speed communication channels (having a maximum bandwidth of 640 Mbs/s for 64K messages) in a 2D grid topology. Each node of this machine has, in addition to the main processor (i860), a co-processor dedicated to communications with the other nodes. The transmission rate for these measures is different for each packet size and optimized to fit the throughput of the receiving server, thus avoiding artificial congestions. The results appear in Figure 1. In any of these cases, the physical network bandwidth and the internal processing speed (since a i860 has approximately the same processing power as a Sparc) of our SMDS server both widely exceed the server maximal unidirectional flow rate. This means that in our system, there exists a bottleneck located in the lower layers of the protocol (probably due to some inefficiencies at the network-server interface, to segmentation and reassembly of packets, and to OS overhead).

3 A Scalable Proposal for the SMDS Server

3.1 Circumventing the bottleneck with parallelism

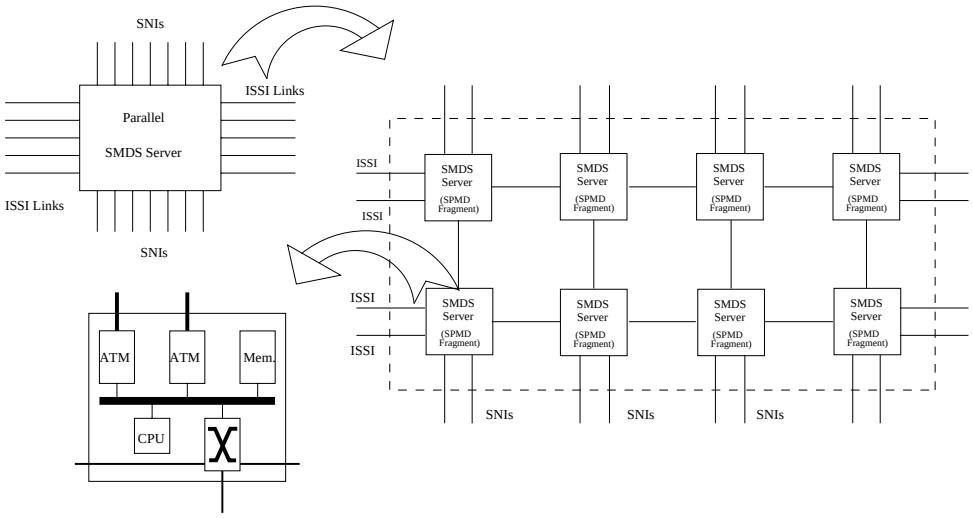

Fig. 2. Distributing the network connections to get a Parallel SMDS server

The logical solution to this problem consists in handling various network access points in parallel. For that, we distribute the ISSI and SNI connections among the various nodes of a parallel computer (see figure 2), each one having its own OS and interface(s) with the SMDS network, and collaborating with other nodes to provide the SMDS service.

But in most of the classical parallel switching systems, the collaboration among processors is carried on through the use of a shared memory. This has the advantage

of providing a simple and well understood programming model to the switch designers. However, this kind of architecture suffers from a serious drawback. Because each data has to transit through the bus at least twice (at incoming and outgoing times, plus access/update from the controlling software), the shared bus becomes yet another bottleneck limiting the total data flow rate of the switch. Even if better bus technologies (e.g., segmented buses) can be used to improve a bit performances, they not not solve the scalability problem. That is, there is a limit above which adding new processing elements or network interfaces does not bring new performance improvement [5].

Circumventing bus bottlenecks in number crunching parallel computing was the main reason why Distribute Memory Parallel Computers (DMPC) started to be built around 10 years ago. Today, this kind of architecture tends to be ubiquitous where high performance computing is a concern, from simulation and numerical computing to database management. It is our strong belief that they could also be use more widely in the telecommunication domain, as a cheap replacement for dedicated hardware solutions. Indeed, in most commercial products, a full-custom fixed-size crossbar-like interconnection matrix is used as the core of the switching system (see e.g., [10]).

Modern DMPCs are built around scalable, high-bandwidth, low-latency internal packet switched networks. They free programmers from having to concern themselves with interconnect topology. All nodes appear to be connected to all other nodes, and communication performance is uniform.

3.2 Scalable Parallelization Method

It has long been a problem to provide good programming environments and parallelization methods for DMPCs. However as the field is maturing, a set of well working approaches has been worked out [3, 6, 7, 8]. The parallelization technique we use is related to the SPMD (Single Program, Multiple Data) model. The code of the SMDS server is duplicated on each node of the DMPC, and at runtime most of the objects (such as the Data Transfer Entity, the Routing Management Entity, etc.) are also duplicated on each node. Only globally needed objects (such as the actual routing table, the LSA database, etc.) need to be shared among the processors of the DMPC. To achieve that, we can choose between two strategies:

1. Allocate these objects on a virtually shared memory (VSM), if it is available on the DMPC[12]. It provides a virtual address space that is shared by a number of processes running on different processors of a DMPC. In order to distribute the virtual address space, the VSM is partitioned into pages which are spread among local processor memories according to a mapping function. Each local memory acts as a large software cache for storing pages. When a page fault happens, a copy of the source page is obtained in case of a read request, whereas an invalidation protocol is used for write requests. In our SMDS server, the writing of routing information occurs much less often than the reading of these informations. This means that after a while, all processors needing a specific part of the routing table would eventually have it available locally, without paying the price of a page fault.

2. Distributed each one of these global objects into a set of fragments in such a way that the well known *local write* [3] principle applies. Basically, this means that the data relevant to a given network connection is allocated on the processor controlling this network connection. Since updating information may come from this link only, there is no need for remote writes, but just for remote reads. These remote reads are implemented as some kind of RPC (e.g., in the context of CORBA) with an intelligent caching of the information.

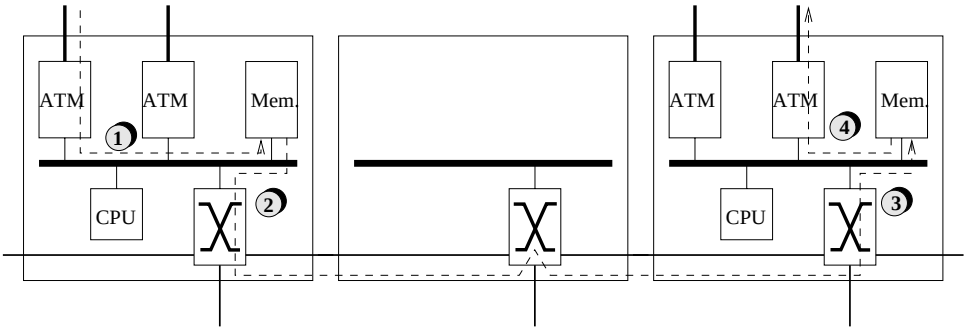

Fig. 3. Distributed Basic Switching

In both cases, we may distinguish between two modes of operation depending on whether basic switching processing or signaling processing is the matter.

Basic switching works as illustrated in Figure 3. Upon detecting an incoming packet (1), the network interface board (ATM, DQDB or any other) reassembles the cells as to produce a level 3 packet that is stored into the node memory. When the CPU is available to process this incoming packet, it computes its outgoing route (using the virtually shared routing tables) and modifies its header accordingly. If the corresponding network connection is local to the node, the computation carries on as it was in the experimental SMDS server. If the network connection is managed by another node, then the outgoing packet is routed (2) through the internal network (without transiting through the bus of the intermediate nodes) of the DMPC until it arrives (3) at the destination node memory. When the CPU is available to process this outgoing packet, it simply forwards it to the precomputed network connection (4). That way, at most two processors have to handle the packet, whatever the size of the DMPC.

The processing linked to signaling is a bit more complex, because some synchronizations are needed to update routing information atomically. For example, an SMDS server is notified of the fact that a link goes down somewhere in the SMDS network through the use of LSAs (Link State Advertisements). But since various LSAs can reach a server through distinct paths, they might arrive to different processors of a parallel SMDS server, which must synchronize to update their routing tables (full details of the distributed SMDS signaling processing are described in [4]). However, since network topology changes occur not that frequently (may be on the

order of a few seconds) with respect to basic switching (ten or hundred of thousands per second), the total processing time devoted to signaling processing remains small in the SMDS server, and does not impact much on the overall basic switching performances.

3.3 Prototype Implementation

To compare the performances of a parallel SMDS server with those of our experimental SMDS server, we implemented the distributed basic switching algorithm part of our parallel server on the Paragon XP/S. For that, we made use of our Eiffel Parallel Execution Environment (EPEE) [8], that can be seen as a kind of a toolbox. It mainly consists of a set of cross-compilation tools that allow the generation of executable code for many different DMPCs (Intel Paragon XP/S, Fujitsu AP1000, SGI, network of Unix workstations, etc.). It also includes a set of Eiffel classes that provide facilities for sharing data across a DMPC and for handling data exchanges between the processors of a DMPC in a portable way. Common data distribution patterns and computation schemes are factorized out and encapsulated in abstract parallel classes, which can be reused by means of multiple inheritance. These classes include parameterized general purpose algorithms for traversing and performing customizable actions on generic data structures.

The parallel SMDS server we obtain with this prototype implementation is a fully functional one, with the restriction that routing tables are hard-wired at boot-time, and that all routing related signaling information is simply ignored.

3.4 Measuring Aggregate Bandwidth

These performance tests consist in measuring the maximal switching capacity of a parallel server, depending on the number of processor it has. The test architecture includes:

- a parallel server implemented on 1, 2, 4, 8, 12, 16 and 20 nodes of the Paragon XP/S
- an environment (surrounding the parallel server) made of a number of other SMDS servers (e.g., 24), each one implemented on one node of the Paragon XP/S and achieving traffic generation and absorption.

We are interested in seeing how faster does a parallel server work, depending on the number of nodes it has. So we measure data flow rates for various packet sizes, under different conditions of (random) traffic load. We got the kind of results represented on the figure 4. These aggregate bandwidth results were obtained with 8k packet under three different traffic densities. As expected, the aggregate bandwidth of a parallel server grows with the number of its nodes. An interesting point is the behavior of the server when switching heavy traffic: small servers (less than 4 nodes) are congested and deliver less than 100 Mb/s, whereas the larger ones are able to manage it, delivering all the offered traffic at 620 Mb/s. This is because the SMDS policy is to discard incoming user data packet when the server is overwhelmed.

In case of a light traffic, the discarding rate is low and the total throughput of the server does not benefit from the larger number of processors. Large servers

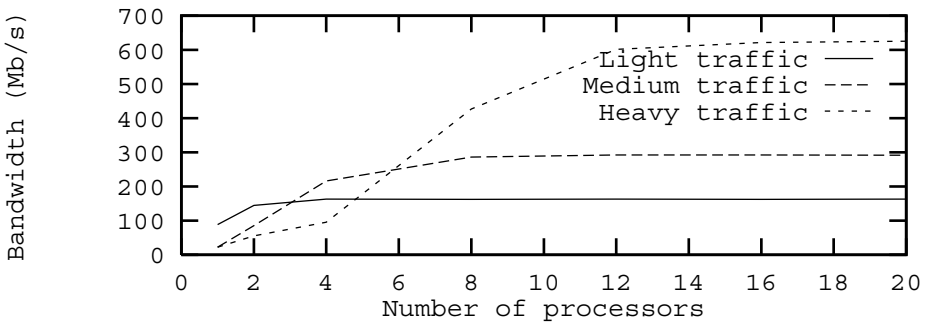

Fig. 4. Total user data flow rate for 32k packets, with various traffic densities

work under their maximal capacity, and are quite overkill in this context. On the other hand, if the traffic is heavier, the small servers crash down: their throughput falls, and the discarding level may becomes really high (up to 87%!). Each parallel SMDS server thus has an optimal traffic range, under which it does not use the maximum of its capabilities, and over which it is congested. By the way, an entity called the *Congestion Management Protocol* is included in the SMDS server to react in real time to the congestion status of the server. When a server is undergoing a congestion, its CMP entity tell the neighbor servers to slash a part of their traffic directed to it.

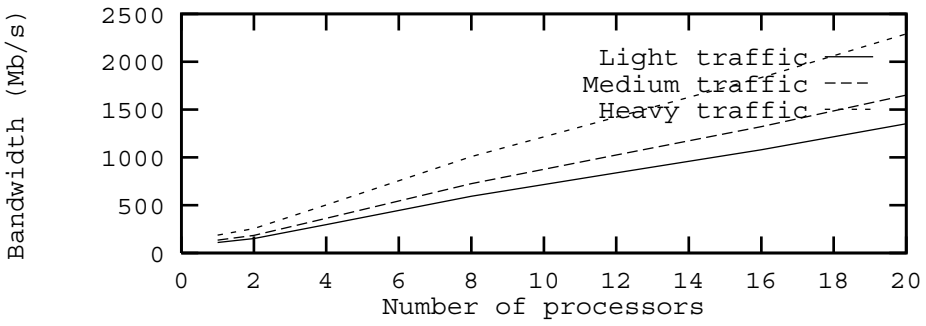

Fig. 5. Maximal aggregate bandwidth in optimal conditions

3.5 Maximal global performances

From all previous measures (various packet sizes and traffic densities), we extract the best aggregates bandwidth results for each size of SMDS servers (from 1 to 20 nodes). These results are displayed on figure 5.

A single processor server achieves user data switching at a speed of 180Mb/s, whereas a 20 processors parallel servers reach nearly 2.3Gb/s, widely above the Gigabit data flow rate given as an unlikely reachable limit in [11]. We obtain a quasi-linear speed-up, with an average efficiency of 66%. Extrapolating this result, even greater flow rates should be achievable by increasing the number of nodes of the parallel SMDS server.

4 Conclusion and Future Prospects

We have outlined the development of a scalable SMDS server using an high level programming environment for distributed memory parallel computers. Through experimental performance studies, we have shown that a bottleneck existed at the network-server interface. This bottleneck was circumvented with a scalable parallelization technique based on the distribution of the network connections across processing elements, and a virtual sharing of globally needed objects such as routing information. Prototype implementation results show that this approach offers interesting performances, since such a parallel SMDS server has a total aggregate bandwidth increasing linearly with the number of its processors to reach 2.3Gb/s for 20 processors on a Paragon XP/S.

Using a high level programming environment along with off-the-shelf DMPCs presents a number of advantages. First, since the parallelization method is fully scalable, the aggregate bandwidth of the server is proportional to the number of supporting nodes, hence making available a *range* of performances easily adjustable to the user needs (because no new software has to be written: since in Eiffel everything is dynamic, the software can configure itself at boot-time). Also, since most of the software is fully portable (only the code dealing with device drivers is system dependent), we can benefit from the exponential increasing of processing power to produce servers that run twice as fast as the previous generation for free. Then, DMPCs have well-known fault-tolerance features that can be used quite transparently in our approach. And finally, because they are themselves built from cheap off-the-shelf components, DMPCs should provide very cost effective solutions for the telecommunication market.

Our approach could be applied everywhere the aggregate bandwidth of a node in a network can be a bottleneck. For example, in the Internet context, both the recent increased demand in bandwidth for multimedia applications and the explosion in the number of hosts attached put a lot of stress on IP routers (e.g., instead of route tables with a few hundred routes, route tables with hundreds of thousands of routes are now considered). Using stock DMPCs could be a cost effective way to handle such multi-megabits, large IP routers, the OO approach providing for an easy migration path toward the unavoidable mutations that can be expected from such a rapidly evolving network.

References

1. Bellcore. Generic requirements for smds networking. Technical Report TA-TSV-001059, Bell Communication Research, 1992.
2. T. Braun and M. Zitterbart. Parallel XTP implementation on transputers. In *The 1991 Singapore International Conference on Networks*, pages 91–96. G.S.Poo, Sep 1991.
3. D. Callahan and K. Kennedy. Compiling programs for distributed-memory multiprocessors. *The Journal of Supercomputing*, 2:151–169, 1988.
4. X. Desmaison. Parallélisation d'un serveur SMDS. Master's thesis, DEA Informatique Rennes I, September 1995.
5. C. Diot. *Architecture pour l'implantation hautes performances des Protocoles de communication de niveau transport.* PhD thesis, Institut National Polytechnique de Grenoble, January 1991.
6. J.J. Dongarra and B. Tourancheau, editors. North Holland, Amsterdam, 1993.
7. D. Gannon, J. K. Lee, and S. Narayama. On using object oriented parallel programming to build distributed algebraic abstractions. In *Proc. of CONPAR92*, 1992.
8. J.-M. Jézéquel. EPEE: an Eiffel environment to program distributed memory parallel computers. *Journal of Object Oriented Programming*, 6(2):48–54, May 1993.
9. J.-M. Jézéquel. *Object Oriented Software Engineering with Eiffel.* Addison-Wesley, March 1996. ISBN 1-201-63381-7.
10. D Kachelmeyer. A new router architecture for tomorrow's internet. White Paper available from NetStar as *http://www.netstar.com/fast/solutions/papers.html*, 1996.
11. J.-Y. Le Boudec, A. Meier, R. Oechsle, and H. L. Truong. Connectionless data service in an atm-based customer premises network. *Computer Networks and ISDN Systems*, 0(26):1409–1424, July 1994.
12. Kai Li. *Shared Virtual Memory on Loosely Coupled Multiprocessors.* PhD thesis, Yale University, September 1986.
13. M. May, P. Thompson, and P. Welch. *Networks, Routers and Transputers.* IOS Press, 1993.
14. A. Tantawy. Réalisation de protocoles à haute performance. In *Actes du colloque CFIP'93 sur l'ingénièrie des protocoles, Monreal.* Hermès, September 1993.
15. M. Zitterbart. High-speed transport components. *IEEE Network Magazine*, pages 54–63, January 1991.

This article was processed using the LaTeX macro package with LLNCS style