An Efficient Communication Architecture for Commodity Supercomputers

Stephan Brauss[1], Martin Frey[2], Martin Heimlicher[2], Andreas Huber[2], Martin Lienhard[1], Patrick Müller[2], Martin Näf[1], Josef Nemecek[1], Roland Paul[2], and Anton Gunzinger[1]

[1] Swiss Federal Institute of Technology Zurich (ETHZ),
{brauss, gunzinger, lienhard, naef, nemecek}@ife.ee.ethz.ch
[2] Supercomputing Systems (SCS),
{frey, heimlicher, huber, mueller, paul}@scs.ch

Abstract. The goal of the Swiss-Tx project is to develop, build and install the first Swiss tera-flop supercomputer called Swiss-T2, which is mainly based on commodity parts. Only the communication hardware and communication software is custom-made, because available off-the-shelf products, such as Ethernet with the socket interface, do not offer the necessary bandwidth, latency, and functionality. In this paper, we present a new efficient communication architecture for commodity supercomputing called *Fast Communication Interface (FCI)*, and we introduce T-NET, the custom-made high-performance communication hardware for the Swiss-Tx supercomputers. The highlights are low-latency, high-bandwidth, and portability. Portability means that the communication hardware and software is mainly platform independent and that a large number of modern workstations and standard operating systems can be used as they are. A full implementation of the standardized MPI (Message Passing Interface), written entirely on top of FCI, is also available.

1 Introduction

Modern general-purpose supercomputers often consist of custom hard- and software. A bad price-to-performance ratio is the result of the long and expensive development time and the low volume of production. By using off-the-shelf workstations and standard operating systems, it is possible to overcome these problems. To prove that it is possible to get supercomputing power with off-the-shelf hard- and software, a new two-year-project called Swiss-Tx [1–3] was launched at the end of 1997 by the Swiss Federal Institutes of Technology in Lausanne (EPFL) and Zurich (ETHZ), the Swiss Commission for Technology and Innovation (CTI), Supercomputing Systems (SCS), and Compaq Computer Corporation. This is the first time that such a project takes place in Switzerland. It is based on a cooperation with two laboratories in the United States: the Sandia National Laboratory and the Oak Ridge National Laboratory. The goal is to develop, build and install a one tera-flop commodity supercomputer with reasonable manpower [1] until beginning of the year 2000. Three additional machines will be built: two small systems with 8 and 16 processors called Swiss-T0 and Swiss-T0 (Dual), and one medium-sized system with 132 processors called Swiss-T1. Baby T1 is a small machine that will consist of the same hardware as T1. It will exist only for a short time and is used to validate the Swiss-T1 communication hardware and software. The tera-flop machine called Swiss-T2 will have 504 processors, 252 Gbytes of main memory, and 5 Tbytes of disk storage (see table 1).

Machine Name	No. of Processors	Peak Perf. (Gflop/s)	Main Memory (Gbytes)	Disk Storage (Gbytes)
Swiss-T0	8	8	2	64
Swiss-T0 (Dual)	16	16	8	170
Baby T1	12	12	6	130
Swiss-T1	132	132	66	1000
Swiss-T2	504	1008	252	5000

Table 1. Parameters of the Swiss-Tx machines

All Swiss-Tx machines are based on Compaq AlphaServers. Swiss-T0 and Swiss-T0 (Dual) are already installed at EPFL and use Compaq Tru64 UNIX as operating system. Originally, Swiss-T0 (Dual) was delivered with Microsoft Windows NT Version 4.0, but it has been changed to Compaq Tru64 UNIX after three months. Baby T1 and Swiss-T1 will use Compaq Tru64 UNIX. The operating system for the Swiss-T2 is not yet chosen.

All Swiss-Tx machines mainly consist of commodity parts. Only the communication hardware and software is custom-made because off-the-shelf products (e.g. Ethernet with the socket interface) do not offer the bandwidth, latency, and functionality required for the Swiss-Tx machines. Currently available specialized communication hardware and software (like Myrinet [12] in conjunction with Fast or Active Messages [13, 14] or MEMORY CHANNEL2 [15] in conjunction with MPICH [9]) do not fulfill our needs either. Our main goal beside low-latency and high-bandwidth is portability. Portability means that the communication hardware and software is mainly platform independent and that a large number of modern hardware platforms (e.g. PCs and Alpha-based Workstations and Servers) and standard operating systems (e.g. Compaq Tru64 UNIX, Linux, and Microsoft Windows NT) can be used without modifications. To reach our goals, we designed a new efficient communication architecture called *Fast Communication Interface (FCI)*, and developed FCI conform communication hardware and software. Two different communication networks are currently available: EasyNet and T-NET. Both use almost the same communication soft-

[1] About 5 man-years for the communication software and 7 man-years for the hardware.

ware. EasyNet is an inexpensive bus-based low-latency network for up to 8 nodes. T-NET is a switch-based high-bandwidth, low-latency network designed for large and complex network topologies. The topology can be modified on-the-fly and the hardware supports uni- and multicast routing with automatic error correction. Peak bandwidth of the bidirectional links is 100 Mbyte/s in each direction. Swiss-T0 and Swiss-T0 (Dual) use EasyNet, Baby T1, Swiss-T1, and probably Swiss-T2 will use T-NET.

FCI offers two programming paradigms, the widely used message passing and *Remote Store* (see 2.1). To be compatible with other machines, the standardized Message Passing Interface (MPI) [7] is available. The MPI library is written entirely on top of FCI by using the message passing functionality of FCI. Low-latency and high-bandwidth data transfers are achieved by a sophisticated overall concept, zero-copy user-level send and receive operations, small protocol overhead, and intelligent communication hardware. All is implemented without operating system stability and security violations.

2 FCI – The Fast Communication Interface

The *Fast Communication Interface (FCI)* is a new communication architecture for commodity supercomputing. The highlights of FCI are low-latency, high-bandwidth, and portability, while system stability and security is maintained.

Low-latency and high-bandwidth are mainly achieved by a sophisticated overall concept which includes:

- Zero-copy send and receive.
- User-space communication to avoid expensive operating system calls [5].
- Small protocol overhead.
- A well designed hardware/software interface and intelligent communication hardware.

Small protocol overhead is important for a communication network. For example, a system using the TCP/IP protocol [17, 18] to transport data between processes over Ethernet, normally wastes a lot of CPU time, because checksums have to be calculated in software and data must be copied and split up in small network packets. This overhead is mostly hidden in benchmarks measuring only latency and bandwidth but shows up in real applications that need CPU time for calculations.

Designing the hardware for the software and not vice versa is also a major concern. A dedicated and optimized hardware/software interface for commodity supercomputing simplifies the communication software and reduces its overhead.

In modern systems is guaranteed that a process cannot interfere with other processes or even the operating system. System stability and security must be guaranteed in commodity supercomputers and should not be subordinated to the performance.

Portability means that the communication hardware and software is mainly platform independent and that it can be ported easily to new platforms. This is in fact a big advantage because we can reuse most of the software for future machines to reduce costs and development time. We separate the software into two parts: the platform dependent part and the platform independent part. Most code is absolutely hardware and operating system independent. Currently, we use PCs and Compaq AlphaServers with Compaq Tru64 UNIX, Linux, and Microsoft Windows NT [2].

FCI offers the programming paradigms message passing and Remote Store, a lock management (used to restrict access to shared resources) and barriers. The basic functionality of Remote Store and the message passing mechanism are presented in 2.1 and 2.2. Lock management and barriers are not further discussed in this paper.

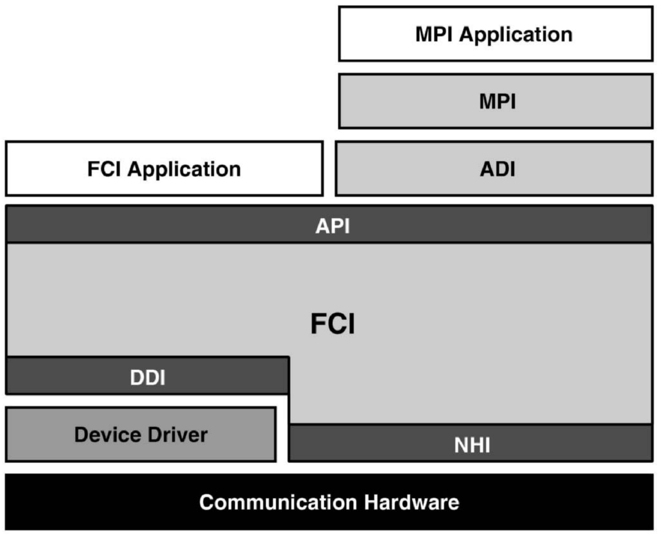

Fig. 1. FCI and its integration in the environment

See figure 1 for an overview of FCI and its environment. MPI is the widely used Message Passing Interface [7]. It is built entirely on top of FCI. The *Abstract Device Interface (ADI)* does a part of the memory management for MPI and handles MPI data types.

[2] All installed Swiss-Tx machines are Compaq AlphaServers with Compaq Tru64 UNIX and Microsoft Windows NT. At ETHZ, we use only PCs with Linux for Swiss-Tx hardware and software development.

FCI has the following three interfaces to the environment:

– *Application Programming Interface (API)*: The interface to the FCI application. An application using the FCI API can be a pure FCI application or for example a higher-level interface such as MPI.
– *Device Driver Interface (DDI)*: The interface to the device driver that is necessary to control the communication hardware. FCI needs a device driver only for start up, close down, and maintenance of an application. It is normally easy to implement it.
– *Network Hardware Interface (NHI)*: The interface to the communication hardware. The communication hardware has a set of mandatory and optional functionality. The optional functionality is used to increase the flexibility and to improve the performance of the network.

The FCI API offers the following routines:

– Management routines (start up, close down, environmental queries).
– Blocking and non-blocking message passing sends, receives, and probes (point-to-point routines).
– Barriers and message passing broadcasts (collective routines).
– Remote Store routines.
– Lock management routines.

2.1 Remote Store

Remote Store is some sort of distributed shared memory with remote writes but no remote reads. It is similar to Reflective Memory [6]. All processes of a parallel application see a virtual *Global Communication Space (GCS)*. They can write to this space and can also receive from it. A process can define several address ranges in the GCS where it wants to receive data. These address ranges are called windows. They can be arranged in any possible variation while overlapping is allowed. Each process that has an open window in the GCS has a corresponding window of the same size in its memory [3]. See figure 2 for an example with three processes named P_x, P_y, and P_z. Each has a local process memory and one open window in the GCS. Window w_x in the GCS belongs to window w'_x in the memory of process P_x, w_y belongs to w'_y in the memory of P_y, and w_z, which is equal to w_y, belongs to w'_z in the memory of P_z. Assume that process P_x wants to write data D to the process memory of P_y and P_z. It knows that P_y has opened window w_y and that P_z has opened window w_z for the same area in the GCS. It writes D to the right location in the GCS (1). D will be transported automatically to the corresponding locations in the memories of processes P_y (2) and P_z (3).

[3] The number of available windows depends on the used communication hardware. The boundaries of such windows must be aligned to a hardware-dependent number of bytes (4 bytes for EasyNet).

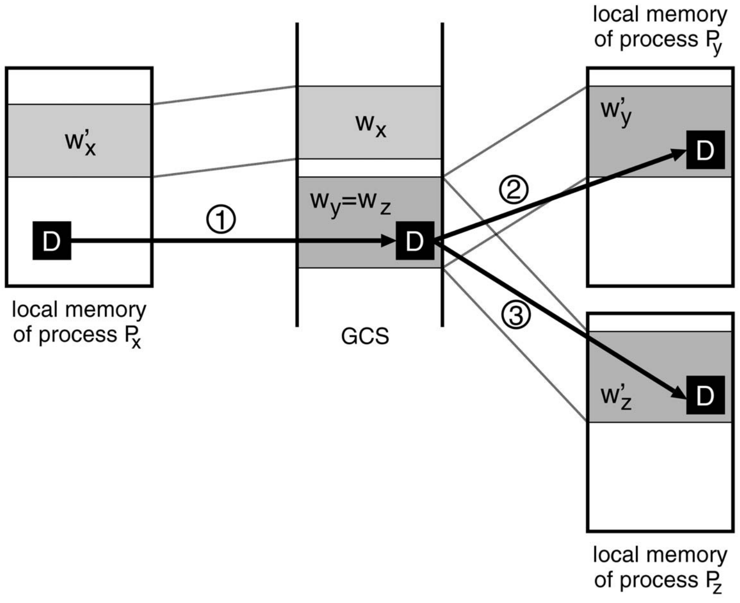

Fig. 2. A part of the *Global Communication Space (GCS)* with three processes P_x, P_y, and P_z. Each process has opened one window

Remote Store avoids two problems of conventional distributed shared memory systems:

- To allow caching of distributed data, distributed shared memory systems need a synchronization mechanism to guarantee data consistency. This makes hardware more complicated and more expensive.
- A read instruction takes two steps. Data has to be requested and afterwards delivered by the owner. A write instruction is faster because the same instance requests the operation and delivers the data. Normally, efficient read instructions supporting bursts are expensive to implement. Remote Store does not support distributed read instructions at all. The necessary hardware is easy to implement.

2.2 FCI Message Passing

The message passing functionality of FCI is a minimum subset necessary to implement MPI. No direct message passing support is included in the FCI NHI, which makes hardware easy to design but demands sophisticated software: The efficient message handling of the *Intelligent Sender Concept* is based on local and distributed tables that are maintained by Remote Store accesses and by the lock management [4].

[4] The lock management is currently implemented in software by use of Remote Store. Lock management hardware support is a optional functionality of the NHI.

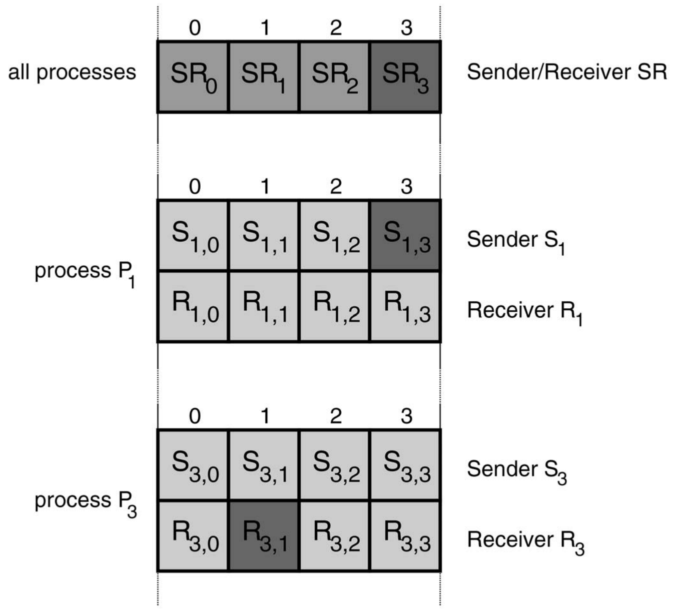

Fig. 3. A part of the tables that hold the receive requests for an application with four participating processes P_0 to P_3. The tables for P_0 and P_2 are omitted

Intelligent Sender means that the complete message transfer can be done by the sending process without any software interaction at the receiving process. The message is directly sent to the right location in main memory of the receiving process. The tables mentioned hold the receive requests that contain the necessary information to execute the transfer. A receive request is set up by a process that wants to receive a message. The request must be visible by the process or processes that are possible senders. To maintain such requests, it is also necessary that the receiving process keeps a local copy of it. Figure 3 shows a part of the local and distributed tables that hold the receive requests for an application with four participating processes. Each square can hold a fixed number of requests. The request tables are named $S_{i,j}$, $R_{i,j}$, and SR_j $(0 \leq i, j \leq 3)$. The uppermost line containing SR_0 to SR_3 is used to maintain receive requests that are set up by a process that wants to receive a certain message from a set of processes, the other lines are involved when the sender is explicitly given. SR_0 to SR_3 are stored on all participating processes, $S_{i,0}$ to $S_{i,3}$, and $R_{i,0}$ to $R_{i,3}$ on process P_i. All processes can write to all tables and read from SR_j. $R_{i,j}$ and $S_{i,j}$ are only readable for P_i. Assume that process P_3 wants to receive a message from process P_1 and that process P_1 wants to send a message to process P_3. Process P_3 inserts a receive requests in $R_{3,1}$ (local) and also in $S_{1,3}$ (remote in process P_1). Process P_1 waits for the matching receive request in $S_{1,3}$, finds it, takes the necessary information out of the request and transfers the message at the right location in the memory of process P_3. Then, process P_1 marks the receive request in $S_{1,3}$ (local) and also in $R_{3,1}$ (remote in process P_3) inactive. Later, process P_3 can clean up the tables and remove the inactive receive requests from $R_{3,1}$ and $S_{1,3}$.

Now, assume that process P_3 wants to receive a message that is allowed to be sent by any process of the message passing application and that process P_1 will be the sender. Process P_3 inserts a receive request in SR_3 which is available for all processes. Process P_1 waits for the matching receive request in SR_3 and finds it. To guarantee that no other process will send a message for the found request, process P_1 has to lock SR_3 exclusively. When it has acquired the lock, it has to rescan SR_3 in order to guarantee that the receive request is still pending (it could be marked inactive or even removed if another process was faster and has attained the lock first). If the request is still available, it sends the message to process P_3 and marks the receive request in SR_3 inactive. In any case, it has to release the lock. Later, process P_3 can remove the inactive request handle.

The Intelligent Sender Concept has the following advantages:

– The tables are easy to handle. No complex and time consuming software is necessary. Using Remote Store makes it fast.
– There is only one way messages are transferred and there are no exceptions like unexpected messages [10]. When a message is sent, the receiver has always set up a matching receive request.
– No code must run on the receiving processing element to enable the reception. In state-of-the-art systems, it is normally expensive to call an interrupt handler or to wake up a kernel thread. In the Intelligent Sender Concept, all code that has to run on a receiving processing element is executed when the receive request is set up (e.g. MPI_Recv, MPI_Irecv) or when the completion of the receive is checked (e.g. MPI_Wait) by the message passing application.

All this helps to keep the communication libraries small and fast. The main drawback of the Intelligent Sender Concept is that a receive of a message, which can be sent by any process of the application, needs a locking mechanism increasing the latency. Additionally, the necessary entries in the distributed tables must be sent to all (or at least a set of [5]) processes, which consumes network bandwidth. A solution to overcome these problems is to avoid such receive requests in the application software whenever possible.

[5] This is a possible improvement of the Intelligent Sender Concept: Such receive requests need to be accessible only for the group of processes that is allowed to send the message. FCI knows and handles process groups. Process groups are for example defined by MPI communicators.

3 Concept of the Communication Network T-NET

T-NET is a high-bandwidth, low-latency System Area Network (SAN) designed for the Swiss-T1 and Swiss-T2 commodity supercomputers. T-NET consists of communication adapters, 12×12 crossbar routers, electrical or optical bidirectional links, and a service network with service workstation. Figure 4 shows an example of a T-NET network, where 4 crossbar routers are connected in a ring. Each crossbar router has 10 communication adapters attached. A communication adapter connects one host [6] to the network. A standard Ethernet LAN is used as service network. It connects the 4 crossbar routers to the service workstation.

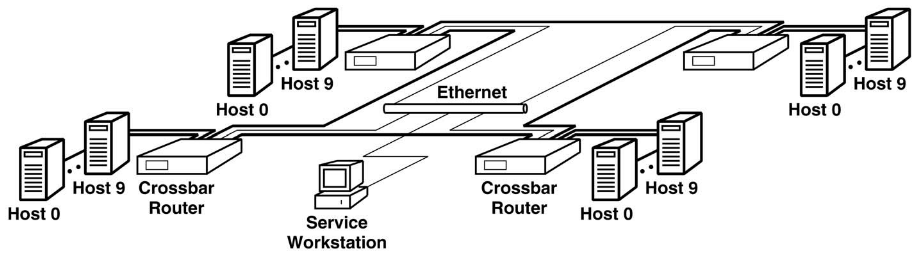

Fig. 4. Example of a T-NET network with 4 crossbar routers and 40 hosts

T-NET is compliant to the *Network Hardware Interface (NHI) Specification*. The NHI specifies the functionality which has to be implemented in the communication hardware to be used with the Fast Communication Interface (FCI). T-NET provides a large set of features. Some of these features are mandatory, others are optional. The most important features are described in the following sections.

3.1 Flexible Network Architecture

T-NET is very flexible regarding topology, dimensions, and size of the network. Any topology can be realized, e.g. 2D-Mesh, 3D-Torus, Hypercube, Multistage, Fat Tree. The dimensions of such a network are restricted by the maximum link length which is 25 m for electrical and 1 km for optical links. An off-the-shelf Media Interface Adapter (MIA) can be used to interface between the two medias. The maximum number of addressable communication adapters in the network is limited to 2^{16}.

3.2 Fast and Efficient Communication

The fast and efficient communication in T-NET is based on the Remote Store Concept (see 2.1). Since reading data remotely is far more expensive than reading

[6] PC, Workstation, or Server with one or several processing elements

it locally, Remote Store allows writes as well as reads in the local host, but only writes and no reads in the remote host. The concept demands that the transferred data is directly stored at the the correct location in the memory of the remote host. T-NET accomplishes this with an address translation mechanism in the communication adapter. No obligations are made by the concept about how data has to be transferred from the host to the network and vice versa. T-NET provides in both directions a Direct Memory Access (DMA) engine located in the communication adapter. To transfer data from the host to the network, the DMA is set up by the host and executed by the communication adapter. To transfer data from the network to the host, the DMA is set up and executed directly by the communication adapter without using any CPU time. Because the DMA initialization costs for small data transfers to the network are higher than for long data transfers, the T-NET communication adapter can also be accessed in Programmed I/O (PIO) mode. This allows the usage of either PIO or DMA for transferring data to the network, whatever is more efficient.

3.3 Reliable Communication

T-NET transfers data from one host to another in packets. The packet flow is controlled by an acknowledge/retransmission protocol on a per-link basis. To detect transmission errors, each packet is tailed by a CRC. This CRC is generated once in the source communication adapter and checked throughout the network in every crossbar router with a final check in the destination communication adapter. In the crossbar router, cut-through routing techniques are used. In cut-through routing, packets are, whenever possible, immediately routed further without prior buffering. This reduces latency but prohibits the crossbar router from removing erroneous packets. When packets cannot be routed immediately because the desired output link is busy, they have to be buffered temporarily in a so-called receive buffer located in the crossbar router. This of course adds additional latency, but allows the crossbar router to discard erroneous packets. In contrast to the crossbar routers, the communication adapters use store-and-forward flow control. In this type of flow control, the entire packet is stored temporarily in a receive buffer before it is forwarded to the host. This allows to detect any transmission errors in time, but in turn, incurs one store-and-forward delay [7]. An additional store-and-forward delay is incurred by the source communication adapter. The communication adapter, in order to reduce the protocol overhead added by the insertion of a packet header, is designed to transmit whenever possible packets that are filled with the maximum payload. Because data for the payload of the packet normally does not arrive from the host in one single burst, the communication adapter has to buffer the packet until enough data has been obtained. To be able to perform a packet retransmission on a per-link basis, each crossbar router and communication adapter stores temporarily all packets with outstanding acknowledgment in a so-called retransmit buffer. This allows to retransmit packets when necessary. The T-NET acknowledge/retransmission

[7] the delay incurred by storing the entire packet before forwarding it

protocol, together with additional error detection on the physical layer of the link, reduces the error rate per link to less than 10^{-20}.

3.4 Multi-Channel Support

T-NET is designed to support several communication channels per communication adapter. This allows concurrent processes of a parallel application on the same host. Each process can use an own private channel. For performance reasons, it is only interesting to run several processes on a single host when each process can also run on a private processing element. Thus, the T-NET multi-channel support is important on Symmetric Multi Processor (SMP) hosts, where several processes running on different processing elements share one communication adapter. In the near future, T-NET will offer an additional channel that is reserved for the operating system. This channel can be used to transfer system data that is currently transported over Ethernet. This will speed up applications using sockets, such as PVM [11] and MPICH [9].

3.5 Protected Communication

T-NET provides several mechanisms to guarantee a protected communication. In a machine running several different parallel applications, where each application runs on a private set of hosts, it is necessary to prevent hosts from one application from being accessed by a host of another application. This prohibits erroneous applications from disturbing other applications. In T-NET, each packet carries an identification number in its header. According to this ID, packets are routed over the network to their destination host. Each communication adapter stores an ID validation bit for all of these IDs in a so-called ID validation table. Before transmitting a packet, the communication adapter checks if the appropriate ID validation bit is set. Packets for which the validation bit is not set are automatically discarded by the communication adapter and a notification is sent to the host. An additional protection mechanism prevents an application from accessing memory in a remote host that does not belong to the application. The communication adapter checks for accesses to unauthorized memory locations and discards them when necessary. To protect the T-NET system channel (see 3.4) from being used by erroneous applications, the communication adapter offers a private set of system channel communication control registers. These registers are located in a separate address range so that they can only be accessed by the system.

3.6 Multicast Capability

T-NET offers the ability to perform multicasts directly in hardware. A multicast packet carries a multicast group ID [8] in its header. The crossbar router supports multicasts with its ability to concurrently route a multicast packet from one

[8] an unique ID for each group of processes participating in the same multicast

input link to all required output links, according to the multicast group ID. The crossbar router obtains the information required to route the packet from the routing table located in the crossbar router. The destination communication adapter supports multicasts by translating the multicast group ID into a process-specific virtual address, making use of an index-to-address translation table, for which the multicast group ID serves as index. Because the virtual address is derived in the destination communication adapter by use of a table, this type of communication is called *table-mapped.* This in contrast to *direct-mapped* communication used for unicasts. In direct-mapped communication, the virtual address is directly sent along with each packet to the destination adapter. The index-to-address translation table is located in the communication adapter and is designed to store more than only one virtual address per ID. This feature, together with the ability of the communication adapter to replicate received packets, allows to expand the multicast support to several processes on the same host.

3.7 Address Translation Mechanism

T-NET provides, in addition to the address translation mechanism offered by direct-mapped and table-mapped communication (see 3.6), the ability to remap data from the network before forwarding it to the host memory. For remapping, a page table located in the communication adapter is used. This feature allows the main memory allocated in the host for communication purposes to be built out of non-contiguous memory blocks. No specialized memory manager which allocates the necessary space in one single contiguous block is necessary. Independent of contiguous or non-contiguous memory allocation is the fact that the allocated memory is not allowed to be swappable. This is no real drawback, since for performance reasons, a Swiss-Tx machine should not swap.

3.8 Smart Packet Routing

T-NET offers a set of routing-related features, allowing smart packet routing. Features concerning the crossbar router include the ability to connect the 12 different input links of the crossbar router independently to one or several of the 12 output links, with the only restriction that the desired output link is not busy. The information to decide how to connect an input to an output link is stored in tables located in the crossbar routers. These tables can be configured either over the service network or over the T-NET network. The configuration can take place whenever required, making on-the-fly reconfiguration possible. An additional feature of the T-NET network is the ability to perform static as well as adaptive packet routing. Packets can be assigned to be routed in a static or adaptive way, whatever is preferred. To avoid deadlocks, the crossbar routers provide deadlock detection and multi-level deadlock resolving mechanisms directly in hardware. Information about potential deadlocks as well as other network related status information can be obtained over the service network and viewed on the service workstation.

4 Implementation of the Communication Network T-NET

4.1 Communication Adapter

The T-NET communication adapter is a 32 bit PCI adapter board with DMA capabilities, offering one bidirectional link to the T-NET network with a peak bandwidth of 100 Mbyte/s in each direction. The adapter board is functionally partitioned into four subsystems: the PCI bridge, the communication controller, the link controller and the link transceiver. FIFO buffers, one for data to the network (TX-FIFO) and one for data from the network (RX-FIFO), decouple the communication and the link controller. Additional on-board memory is used to store communication related tables, i.e. the ID validation table, the page table, and the index-to-address translation table (see figure 5).

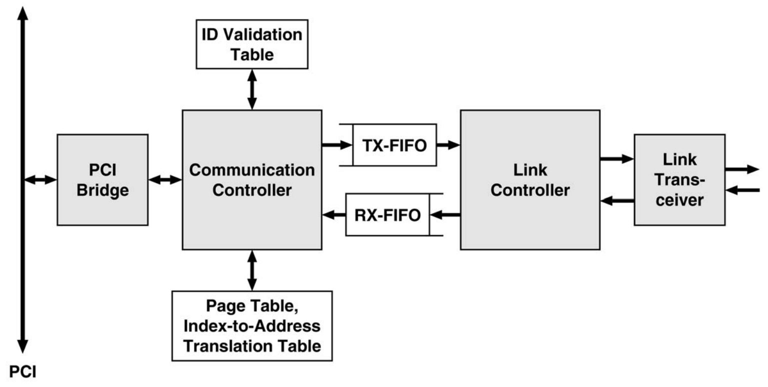

Fig. 5. T-NET PCI adapter block diagram

The communication controller is responsible for packet generation, packet extraction, communication protection and address translation. The link controller is responsible for the acknowledge/retransmission protocol. It accepts data to be transmitted from the TX-FIFO and stores it temporarily in its transmission buffer. As soon as at least one complete packet is in the transmit buffer, the link controller starts forwarding the packet to the link transceiver. In the other direction, the link controller accepts data from the link transceiver, computes the CRC and stores it temporarily in its receive buffer. After the reception of a complete packet, the link controller compares the received with the locally computed CRC and determines whether the packet has to be discarded or can be written to the RX-FIFO. The link controller, as already described in 3.3, incurs two store-and-forward delays, one on the outgoing and one on the incoming packet. The link transceiver is a 1.25 Gbit/s Fibre Channel [16] transceiver. It encodes the data before transmitting it serially over the link, allowing clock recovery and low-level error detection in the receiver.

4.2 Crossbar Router

The T-NET crossbar router has 12 bidirectional ports with a peak bandwidth of 100 Mbyte/s per port and direction. It can achieve a maximum throughput of 1.2 Gbyte/s, while adding a maximum latency of less that 0.5 μs. The design is functionally partitioned into 12 identical link transceiver/link controller pairs, a crossbar, a routing controller, a service controller and a performance collector. Additional on-board memory is used to store routing related tables (see figure 6).

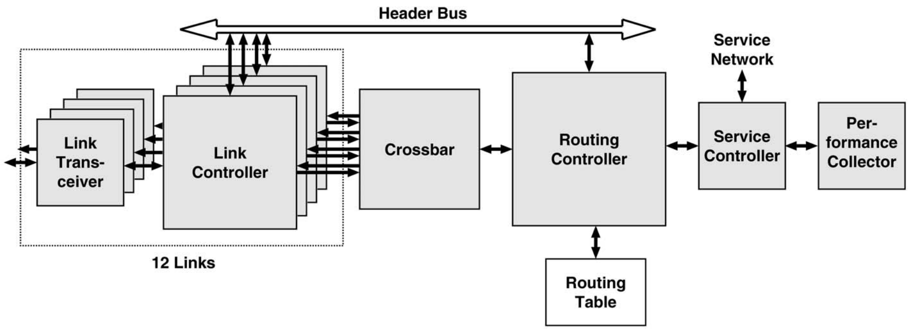

Fig. 6. T-NET router block diagram

The crossbar features 12 input and 12 output ports as well as one bidirectional port. The 12 input and output ports connect to the 12 link transceiver/link controller pairs whereas the bidirectional port connects to the routing controller (see figure 6). Packets can be routed from any of the 12 input ports to any of the 12 output ports as well as from any of the 12 input ports to the bidirectional port or from the bidirectional port to any of the 12 output ports. The bidirectional port is used to temporarily store packets in the routing controller for deadlock resolve purposes and to configure the routing table directly over the T-NET network. The routing controller is connected to the 12 link controllers over the so-called header bus. From the link controllers, the routing controller can obtain over the header bus the headers of all packets which have to be routed. Based on the header information and the information stored in the routing table, the routing controller determines the output links to which a specific packet has to be routed to and establishes the appropriate crossbar connections whenever possible. Routes which cannot be established immediately are set back temporarily for the benefit of another connection. Reasons preventing an immediate establishment of a connection are the ownership of an output link by another input link or the lack of space in the transmit buffer of an output link controller. The link controller is responsible for the acknowledge/retransmission protocol. It accepts data to be transmitted from the crossbar and stores it temporarily in its transmit buffer. As soon as there is data in the transmission buffer, the link controller starts to forward it to the link transceiver. In the other direction, the link controller accepts data from the link transceiver, computes the CRC and stores it temporarily in its receive buffer. As soon as at least the header is stored in the receive buffer, the link controller signals the routing controller the availability of a new packet. Therefore, the link controller in the router, unlike the link controller on the communication adapter, does not incur any additional store-and-forward delays to a packet. The link transceiver is identical to its counterpart on the communication adapter as described in 4.1. The performance collector collects performance data from the 12 link controllers. This includes the total time a link controller spends waiting for the establishment of a crossbar connection, the total number of communicated packets, the total number of words communicated within these packets, the number of packet retransmissions performed, and the number of link errors detected. The service controller connects the routing controller and the performance collector to the service network, allowing for routing table configuration as well as performance and status data exchange between the router and the service workstation.

5 Conclusions and Future Work

Swiss-Tx is the first research project in Switzerland in the field of commodity supercomputing. The communication hardware and software has been developed with reasonable manpower in only three years. Therefore, a sophisticated overall concept is necessary to make the project possible. The main goals of the communication network are to reach high-bandwidth, low-latency, and portability while system stability and security should be still maintained. Hardware and software dovetail well. We believe that the Intelligent Sender Concept is a good way to provide fast message passing. It is fully designed on top of the efficient and easy to implement programming paradigm Remote Store. The communication hardware is flexible and supports multicast sends, automatic routing, and a set of communication channels. A machine using the T-NET network can be partitioned to run several applications concurrently and protected from others.

By the end of July 1999, the T-NET-based Swiss-T1 is still under construction and useful benchmark results cannot be presented for this machine. Measurements on a development machine, consisting of eight standard 233 MHz Intel Pentium PCs connected by the bus-based EasyNet network, proved that our communication architecture enables low-latency and high-bandwidth data transfers. In this machine, EasyNet transmits 32 bits in parallel and runs at 12 MHz. The maximum possible bandwidth is 48 Mbyte/s. The average one-way latency is about 10 μs for a Swiss-Tx MPI message without payload, and 5 μs for a Remote Store 4-byte-packet. The highest bandwidth measured is approximately 46 Mbyte/s in both cases.

In the near future, an enhanced version of the T-NET communication adapter with 64 bit 66 MHz PCI interface and two independent bidirectional T-NET links will be available. First samples of such boards are already under test. The crossbar router will also be improved so that adaptive routing will be possible.

This new communication hardware will offer a higher overall communication performance. It will be probably used in the Swiss-T2 tera-flop supercomputer.

A lot of work has still to be done. This includes further improvements and optimizations of the hardware and software, the implementation of parts of MPI-2 [8], as well as the necessity to provide an efficient resource management (e.g. a user-friendly runtime environment and parallel debugging capabilities) and useful programming tools (e.g. compilers and performance tools).

References

1. Swiss-Tx Architecture. Swiss Federal Institute of Technology Lausanne (EPFL), `http://capawww.epfl.ch/swiss-tx/index.html`
2. Gruber, R., Gunzinger, A.: The Swiss-Tx Supercomputer Project. EPFL Supercomputing Review, **9** (1997) 21–23
3. Gruber, R., Dubois-Pèlerin, Y., Swiss-Tx Group: Swiss-Tx: First Experiences on the T0 System. EPFL Supercomputing Review, **10** (1998) 19–23
4. Brauss, S., Nemecek, J.: The FCI Reference Manual. Swiss Federal Institute of Technology Zurich (ETHZ), `http://www.ife.ee.ethz.ch/hpc/fci`
5. Araki, S., Bilas, A., Dubnicki, C., Edler, J., Konishi, K., Philbin, J.: User-Space Communication: A Quantitative Study. `http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Bilas820/index.htm`
6. Protić, J., Tomašević, M., Milutinović, V.: Distributed Shared Memory: Concepts and Systems. University of Belgrade, IEEE Parallel and Distributed Technology (Summer 1996)
7. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. University of Tennessee (1995), `http://www.mpi-forum.org/docs/mpi-11.ps`
8. Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface. University of Tennessee (1997), `http://www.mpi-forum.org/docs/mpi-20.ps`
9. MPICH - A Portable Implementation of MPI. University of Tennessee, `http://www-unix.mcs.anl.gov/mpi/mpich/`
10. Gropp, W., Lusk, E.: The implementation of the second generation MPICH ADI. University of Chicago, `http://www.mcs.anl.gov/mpi/mpich/workingnote/adi2impl/note.html`
11. PVM: Parallel Virtual Machine. Oak Ridge National Laboratory, `http://www.epm.ornl.gov/pvm/pvm_home.html`
12. Boden, N., Cohen, D., Felderman, R., Kulawik, A., Seitz, C., Seizovic, J., Su, W.: Myrinet: A Gigabit-per-Second Local Area Network. IEEE Micro Vol. **15**, No. **1** (1995)
13. Pakin, S., Lauria, M., Chien, A.: High Performance Messaging on Workstations: Illinois Fast Messages (FM) on Myrinet. `http://www-csag.ucsd.edu/papers/csag/external/HPVMFM-p.html`
14. Chun, B., Mainwaring, A., Culler, D.: Virtual Network Transport Protocols for Myrinet. IEEE Micro, Vol. **18**, No. **1** (1998)
15. Fillo, M., Gillett, R.: Architecture and Implementation of MEMORY CHANNEL2. Compaq Computer Corporation, `http://www.digital.com/info/DTJP03/DTJP03HM.HTM`

16. Fibre Channel Working Set. American National Standards Institute. ISBN **1-57053-009-2** (1994)
17. Postel, J.: Internet Protocol. University of Southern California, RFC791, `http://www.cis.ohio-state.edu/htbin/rfc/rfc791.html`
18. Postel, J.: Transmission Control Protocol. University of Southern California, RFC793, `http://www.cis.ohio-state.edu/htbin/rfc/rfc793.html`