Performance Evaluation of MPI and MPICH on the Cray T3E

Hans-Hermann Frese and *Harald Knipp*
Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)
Berlin, Germany
Email: frese@zib.de, knipp@zib.de

ABSTRACT: *MPI is a portable message passing interface for parallel applications on distributed memory machines. In this paper we present performance results for the native SGI/Cray MPI implementation and the portable ANL/MSU MPICH implementation on the Cray T3E.*

1 Introduction

The Message Passing Interface (MPI) provides a portable programming interface for the development of parallel applications on MIMD machines with distributed memory. MPI defines a core of library routines for point-to-point communication, broadcast, and collective operations (reduction algorithms). It also allows programmers to define their own data types, groups, contexts, and communicators. MPI includes useful features from other systems like Chameleon [25], P4 [5][6], PICL [19], PVM [14], Express [35], PARMACS [7], TCGMSG [26], and Zipcode [29].

The development for MPI began as early as 1992 [1][17][37]. MPI was designed by the Message Passing Interface Forum, a large group of application scientists, programmers, and computer vendors representing various organizations from universities, research laboratories, and industry [16]. A first proposal for MPI [13] was presented in 1993. The MPI Forum presented the first MPI standard [30] in 1994 which was updated for MPI 1.1 [31] in 1995 and MPI 2.0 [32] in 1997.

Overall the MPI Standard 1.1 [31] defines about 125 functions, but the basic set contains only 6 indispensable functions. The new MPI-2 Standard [32] provides extended functionality through one-sided communication and parallel MPI-IO.

MPI is available on wide range of platforms from PCs to Supercomputers. In addition to commercial implementations provided by computer vendors, public-domain implementations are available from a number of research centers.

Section 2 gives a short overview of the Cray T3E hardware and its communication network. Section 3 briefly describes MPI's overall functionality. Section 4 presents the implementations which were used for the performance results shown in section 5. In section 6 we discuss our measurements with previously published results [2][3][28].

2 Cray T3E

The Cray T3E [36] is a scalable distributed memory massively-parallel programming (MPP) system. It consists of up to 2,048 processing elements (PEs) connected through a bidirectional 3D torus. A full configured T3E system has an aggregated peak performance of 2.4 Tflop/s.

2.1 T3E Processing Nodes

Each processing node of the T3E consists of a DEC Alpha 21164 EV5 RISC processor, a system control chip, local memory of up to 2 GByte, and a network router. The system logic runs at 75 MHz.

T3E systems have been available with processors running at 300, 450, or 600 MHz. Each processor has two 8 KByte level-one data and instruction caches and a 96 KByte level-two cache. For consecutive data transfer from memory to the processor the cache can be bypassed through 6 stream buffers. As the DEC Alpha processor can perform on floating-point add and one floating-point multiply per clock period, a T3E node has a peak performance of 600, 900, or 1200 Mflop/s resp.

A block diagram of a T3E processing node is shown in appendix A.

2.2 T3E Communication Network

The T3E processor nodes are connected through a 3D torus interconnect network. The raw bandwidth of the 3D torus is 600 MByte/s per link in each direction with a nominal latency of 1 µs. This gives an overall network bandwidth of 3.6 GByte/s per node.

The physical layout of the interconnection network is shown in appendix B.

2.3 T3E Programming Models

The Cray T3E supports two distinctive programming models:

- implicit parallel programming (data parallel)
- explicit parallel programming through message passing.

Implicit data parallel programing is supported through Co-array Fortran (CF90), High-Performance Fortran (PGHPF) and Cray Adaptable Fortran (HPF_CRAFT) and a global address space.

Nevertheless, for performance reasons most of the parallel applications for the Cray T3E are developed through the use of an explicit parallel message passing library. The Cray Message Passing Toolkit (MPT) [10][11][12] provides 3 popular message passing libraries:

- Message Passing Interface (MPI)
- Parallel Virtual Machine (PVM)
- Cray Shared Memory Library (SHMEM).

2.4 T3E Configuration at ZIB

The performance benchmarks were run on a Cray T3E-900 with 256 application PEs each with 128 MByte local memory. The PEs were use in dedicated mode which means that a single application was executed on each PE.

The supercomputer configuration at ZIB which also includes a Cray J90 parallel vector machine, an SGI Origin 200 file server, and two STK 4400 ACS tape robot silos is shown in appendix C.

3 MPI

The Message Passing Interface (MPI) defines standardized and portable communication interface library for memory parallel programming systems. The MPI standard [30] which was published in 1994 is a leading standard for message-passing libraries for parallel computers. It defines a core library of routines for the following mechanisms:

- Point-to-point communication
- Collective operations
- Process groups
- Communication domains
- Process topologies
- Environmental management and inquiry functions
- Profiling interface
- Language bindings for Fortran and C.

The MPI-2 [32] extensions provide even more functionality:

- Dynamic process management
- Input/output
- One-sided communication
- Language binding for C++.

The books [15][20][24][34] provide a comprehensive presentation of MPI's overall functionality.

3.1 Point-to-Point Communication

The basic communication mechanism of MPI is the transfer of data between a pair of processes, one sender and one receiver. MPI provides a set of send and receive functions that allow the transfer of data of a specified type with an associated tag. The message tag allows the receiver to select from different incoming messages.

Basic MPI send and receive functions are built according to the following fragments:

- `send ( address , length , destination , tag )`
- `recv ( address , maxlen , source , tag , actlen )`

where

- `address` defines a memory location containing the data to be sent or a buffer to receive the data
- `length` and `maxlen` define the (maximum) length of the message in bytes
- `source` and `destination` define the process identifier of the sending or receiving process
- `tag` defines an arbitrary message tag (a non-negative `integer` value)
- `actlen` is set to the actual length in bytes of the message received.

The data typing helps to maintain the correct data representation even in a heterogeneous environment where different architectures are linked together. The message tag allows to select specific messages through their tag by the receiving task.

Different functions are provided for blocking (synchronous) and non-blocking (immediate) message transfers. A blocking send blocks the sending process until the buffer containing the data to be sent can safely be overridden. Similarly, a blocking receive blocks the receiving process until the data has arrived. Non-blocking send and receive allow the overlap of the message transfer with other message transfers or computations. Non-blocking send and receive consist of two parts: the posting function, which starts the requested operation; and the test-for-completion function, which allows the program to determine whether the requested operation has completed.

In section 5.1 we present the performance results for blocking, synchronous, and non-blocking point-to-point communication on the Cray T3E.

3.2 Collective Operations

Collective operations are provided to transmit data between all processes in a group or to synchronize processes. MPI provides the following collective operations:

- Barrier synchronization across all group members
- Broadcast from one member to all members of a group
- Gather data from all group members to one member or all members of the group (all-gather)

- Scatter data from one member to all members of a group
- Scatter/gather data from all members to all members of a group (all-to-all or complete exchange)
- Reduction operations like sum, max, min, or user-defined functions

The syntax and semantics of MPI collective operations ar consistent with MPI point-to-point communications. However, the collective operations are more restrictive.

In section 5.2 we present the performance results for MPI_Bcast and MPI_Reduce on the Cray T3E.

4 Implementations

For our performance measurements we investigated the following MPI implementations for the Cray T3E:

- Cray Message Passing Toolkit (MPT)
- ANL/MSU MPICH

4.1 Cray Message Passing Toolkit (MPT)

Cray's implementation of MPI for the Cray T3E is included in the Cray Message Passing Toolkit (MPT) [10][12]. It was derived from the Edinburgh Parallel Computing Centre's implementation of MPI for the Cray T3D [8][9]. MPT is also available for the Cray UNICOS and the SGI IRIX operating system.

For our performance measurements we used the Cray Message Passing Toolkit (MPT) Release 1.3 [10][12].

4.2 ANL/MSU MPICH

MPICH [17][22][23] is a portable implementation of MPI developed at Argonne National Laboratory and the Mississippi State University. MPICH uses an abstract device interface [21], thus providing portability of the implementation on a large range of machines. Whereas most MPICH implementations use P4 [5][6] or Nexus [18] as a low level device interface, on the Cray T3D and the Cray T3E the Cray Shared Memory Access Library (Shmem) is used for efficiency [4][27].

For our performance measurements we used the MPICH Release 1.1. The results show that this public-domain implementation runs very efficiently on the Cray T3E.

5 Performance Results

5.1 Point-to-point Communication

In a first approach we used a simple linear model to calculate the bandwidth and the latency: $T = t_s N + t_l$, where t_s denotes the time to transfer one byte, N the number of bytes transferred, t_l the latency, and T the total time. But as our test runs showed that the latency cannot be assumed as constant. Because different protocols are used for different message sizes, as stated in [4][8][27], the latency is strongly correlated to the size of a message transferred. Therefore we present the bandwidth as a function of the message size. For all investigated functions the transfer rate reached its maximum at about one megabyte for the message size.

To obtain a lower bound for the latency we measured the time to transfer a message of zero length. To get more detailed performance information for large messages sizes we measured the transfer time for messages between one and two megabyte in size. As the latency (and therefore the bandwidth) is nearly constant for messages of this size, the above stated equation holds and the latency can be directly obtained from the data by linear regression.

For point-to-point communications we investigated standard blocking, synchronous blocking, and standard non-blocking send/recv functions. For performance evaluation we used a simple round-trip (ping-pong) algorithm where the data is sent fourth and back several times for error correction, and the bandwidth is calculated from half the round-trip time. As an example the following C-code fragment shows the implementation for the timing of the standard blocking send operation:

```
if ( thisPe == masterPe ){
   /* Code for master PE */
   startTime = MPI_Wtime();
   for ( i=0 ; i<transferRepeats ; i++){
       MPI_Send( sendData, msgLen, MPI_BYTE,
                 slavePe, 1, MPI_COMM_WORLD);
       MPI_Recv( recvData, msgLen, MPI_BYTE,
                 slavePe, 1, MPI_COMM_WORLD,
                 &status);
       }
   endTime = MPI_Wtime();
   }
if ( thisPe == slavePe ){
   /* Code for slave PE */
   for ( i=0 ; i<transferRepeats ; i++){
       MPI_Recv( recvData, msgLen, MPI_BYTE,
                 masterPe, 1, MPI_COMM_WORLD,
                 &status);
       MPI_Send( sendData, msgLen, MPI_BYTE,
                   masterPe, 1,
MPI_COMM_WORLD);
       }
   }
time = (endTime - startTime)/
       transferRepeats/2;
```

5.1.1 Blocking Send and Receive

Figure 1 shows the performance for MPI_Send/Recv. The maximum bandwidth for MPICH was 342 MByte/s. The minimum latency was 7 µs and for message sizes larger than one megabyte was also 7 µs. Cray MPI only achieved 163 MByte/s transfer rate. The minimum latency was 8 µs and for messages larger than one megabyte 56 µs.

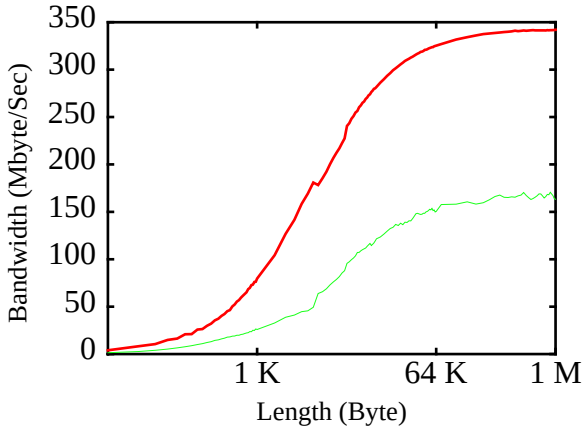


Figure 1: MPI_Send bandwidth for MPICH (upper line) and Cray MPI (lower line)

1.0.1 Synchronous Blocking Send and Receive

Figure 22 shows the performance for the synchronous blocking send/receive operation MPI_Ssend/Recv. In synchronous mode MPICH achieved a bandwidth of 334 MByte/s, a minimum latency of 8 µs, and for messages larger than one megabyte 10 µs. Cray MPI achieved 326 MByte/s bandwidth, a minimum latency of 29 s and for messages larger than one megabyte 126 µs.

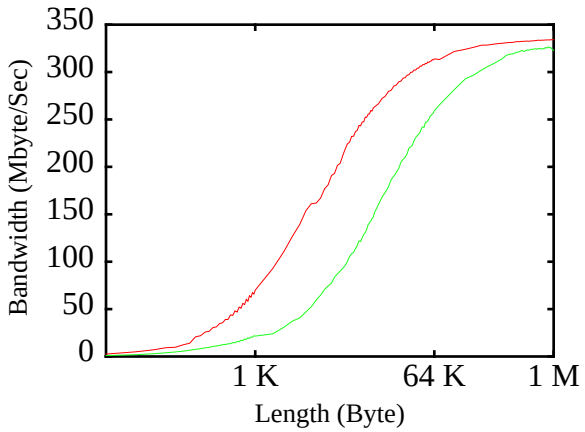


Figure 2: MPI_Ssend bandwidth for MPICH (upper line) and Cray MPI (lower line)

2.0.1 Non-blocking Send and Receive

The non-blocking send operation MPI_Isend achieved a bandwidth of 348 MByte/s for MPICH. The latency reached from 12 to 28 µs for zero length messages and was 13 µs for messages larger than one megabyte. Cray MPI achieved 167 MByte/s bandwidth, the latency for zero size messages reached from 24 to 34 µs, and the latency for messages larger than one megabyte was 55 µs (see fig. 3).

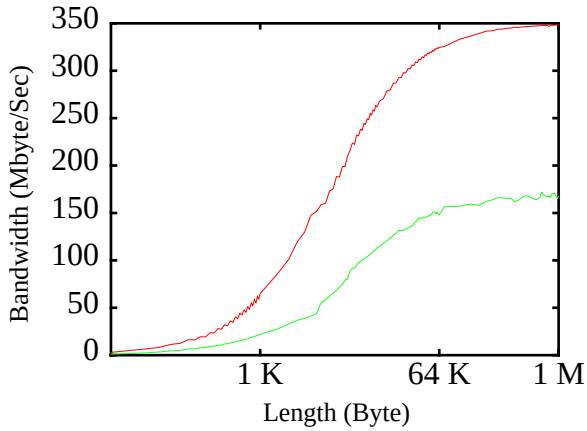


Figure 3: MPI_Isend bandwidth for MPICH (upper line) and Cray MPI (lower line)

3.1 Collective Operations

We investigated MPI_Bcast and MPI_Reduce, the latter with floating point sum as the reduction operation.

3.1.1 MPI_Bcast

To evaluate the performance of MPI_Bcast we used the following algorithm: First all processes are synchronized, then MPI_Wtime is called on the root process to determine the start time. After this, MPI_Bcast is called several times in a loop for error correction, and then MPI_Wtime is called on *all* processes, and finally the maximum of the last timer call of all processes is taken to determine the overall time. The following C-code fragment shows the actual implementation:

```
MPI_Barrier(MPI_COMM_WORLD);
startTime = MPI_Wtime();
for ( j=0 ; j<TREPEATS ; j++ ){
    MPI_Bcast( data, LEN, MPI_BYTE, root,
               MPI_COMM_WORLD );
    }
endTime = MPI_Wtime();
MPI_Barrier( MPI_COMM_WORLD);
MPI_Reduce( &endTime, &maxEndTime, 1,
            MPI_DOUBLE, MPI_MAX, root,
            MPI_COMM_WORLD );
if ( thisPe == root )
   time[i] = (maxEndTime - startTime)/
                              TREPEATS;
   }
```

For the test runs a message of 1024 bytes was distributed from a root process to 1, 3, 5, ... 127 processes. Figure 4 shows that Cray MPI performs and scales much better than MPICH. For Cray MPI the broadcast to 127 processes took 87 µs with an average of 11.2 MByte/s and accumulated 1.4 GByte/s bandwidth. With MPICH the same operation took 170 µs with an

average of 5.8 MByte/s and accumulated 730 MByte/s bandwidth

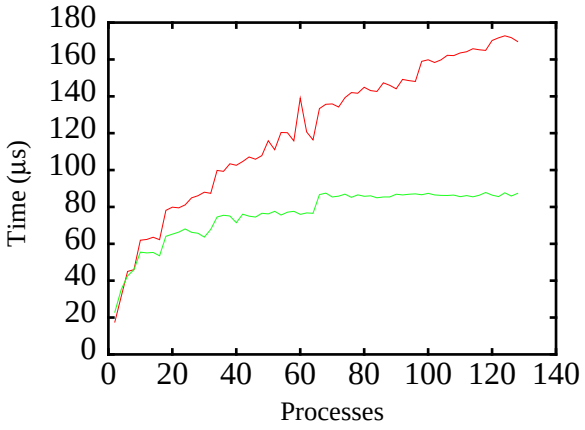


Figure 4: Total time for MPI_Bcast with MPICH (upper line) and Cray MPI

4.0.1 MPI_Reduce

As MPI_Reduce does not return until all values are gathered on the root process and the reduction operation has finished, the algorithm for performance measurement simply calls MPI_Reduce in a loop several times for error correction, measures the total time, and calculates the time per run.

We used MPI_Reduce to calculate the global sum of 128 real values (1024 bytes) per process over 2, 4, 6, ... 128 processes. Figure 5 shows the corresponding times. Cray MPI needs 1214 µs for a reduce operation over 128 processes whereas MPICH needed 1318 µs.

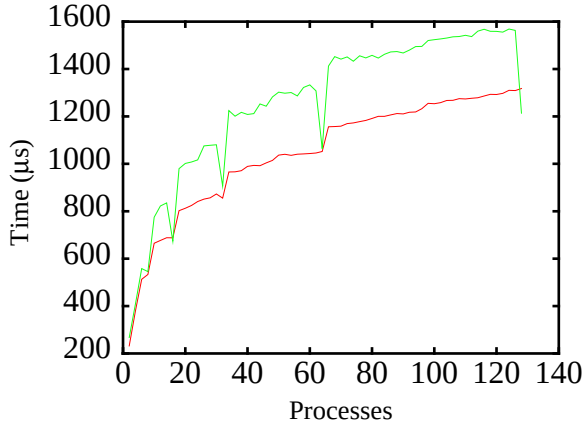


Figure 5: Total time for MPI_Reduce with DSUM as reduction operation for MPICH (lower line) and Cray MPI

6 Discussion of the Results

Overall, our measurements for standard blocking MPI_Send/Recv correspond with the results published in [28]. For synchronous MPI_Ssend/Recv we obtained a slightly better performance than Berger et al. in [2] and [3].

For MPI_Bcast Cray MPI scales and performs better than MPICH. One reason may be that in MPICH MPI_Bcast is implemented on top of the point-to-point communication primitives.

For MPI_Reduce MPICH scales and performs better than Cray MPI. Only if the total number of processes equals a power of two, the performance of both implementations is nearly the same. The only exception is for 128 processes where Cray MPI is faster than MPICH.

Credits

We would like to thank the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) for granting access to their Cray T3E to run the performance benchmarks.

References

[1] P. Bangalore, Nathan Doss and Anthony Skjellum. Writing libraries in MPI. In *Proc. Scalable Parallel Libraries Conf.*, pages 166-173. IEEE Computer Society, 1993.

[2] Holger Berger, Thomas Bönisch, Rolf Rabenseifner, and Michael Resch. *Performance of MPI on the CRAY T3E-512*. Benutzerinformation des Rechenzentrums, Universität Stuttgart, BI 5+6/1997

[3] Holger Berger, Thomas Bönisch, Michael Resch, and Dirk Sihling. Performance of MPI on a CRAY T3E-512. Proc. *Third European CRAY-SGI MPP Workshop*, PARIS (France), Sept. 11 and 12, 1997

[4] Ron Brightwell and Anthony Skjellum. MPICH on the T3D: A Case Study of High Performance Message Passing. Technical Report, Mississippi State University.
`http://www.cs.sandia.gov/~bright/mpi/t3dpaper/t3dpaper.html`

[5] Ralph Butler and Ewing Lusk. User's guide to the P4 parallel programming system. Technical Report ANL-92/17, Argonne National Laboratory, October 1992

[6] Ralph Butler and Ewing Lusk. Monitors, messages, and clusters: The P4 parallel programming system. *Parallel Computing*, 20(4):547-564, April 1994.

[7] R. Calkin, Rolf Hempel, H. Hoppe, and P. Wypior. Portable programming with the PARMACS Message-Passing Library. *Parallel Computing*, 20(4):615-632, April 1994

[8] Kenneth Cameron, Lyndon J. Clarke, and A. Gordon Smith. CRI/EPCC MPI for CRAY T3D. In *Proc. 1st European Cray T3D Workshop*. EPFL, September 1995.

[9] Kenneth Cameron, Lyndon J. Clarke, A. Gordon Smith, and Klaas Jan Wierenga. Using MPI on the Cray T3D. Technical Report, University of Edinburgh, June 1997.
`http://www.epcc.ed.ac.uk/t3dmpi/Product/Docs/user/ug-plain.html`

[10] Cray Research. *Message Passing Toolkit: MPI Programmer's Manual*. SR-2197 1.3.

[11] Cray Research. *Message Passing Toolkit: PVM Programmer's Manual*. SR-2196 1.3.

[12] Cray Research. *Message Passing Toolkit: Release Notes*. RN-5290 1.3.

[13] Jack Dongarra, Rolf Hempel, Anthony Hey, David Walker. A proposal for a user-level, message passing interface in a distributed memory environment. Technical Report TM-12231, Oak Ridge National Laboratory, February 1993.

[14] Jack Dongarra, Alan Geist, R. Manchek, and V. Sunderam. Integrated PVM framework supports heterogeneous network computing. *Comput. Phys.*, 7(2):166-175, April 1993.

[15] Jack Dongarra, Steven Huss-Lederman, Steve Otto, Marc Snir, and David Walker. *MPI: The Complete Reference*, Vol. 1. MIT Press, 2nd edition, 1998.

[16] Jack Dongarra, Steve Otto, Marc Snir, and David Walker. A Message Passing Standard for MPP and Workstations. *Comm. ACM*, 39(7):84-90, July 1996.

[17] Nathan Doss, William Gropp, Ewing Lusk, and Anthony Skjellum. A high-performance, portable implementation of the MPI message-passing interface standard. *Parallel Computing*, 22:789-828, 1996.

[18]Ian Foster, Carl Kesselman, and Steven Tuecke. The Nexus task-parallel runtime system. In *Proc. 1st Intl. Workshop on Parallel Processing*, pages 467-462. McGraw Hill, 1994.
[19]Alan Geist, Michael Heath, B. Peyton, and P. Worley. PICL: A portable instrumented communications library. Technical Report TM-11130. Oak Ridge National Laboratory, July 1990.
[20]William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI: The Complete Reference - The MPI-2 Extensions*, Vol. 2, MIT Press, 2nd edition, 1998.
[21]William Gropp and Ewing Lusk. An abstract device definition to support the implementation of a high-level message-passing interface. Technical Report MCS-P342-1193, Argonne National Laboratory, 1993
[22]William Gropp and Ewing Lusk. Installation Guide to `mpich`, a Portable Implementation of MPI. Technical Report ANL/MCS-TM-ANL-96/5, Argonne National Laboratory, 1996.
`http://www.mcs.anl.gov/mpi/mpiinstall/paper.html`
[23]William Gropp and Ewing Lusk. User's Guide for `mpich`, a Portable Implementation of MPI. Technical Report ANL/MCS-TM-ANL-96/6, Argonne National Laboratory, 1996.
`http://www.mcs.anl.gov/mpi/mpiuserguide/paper.html`
[24]William Gropp, Ewing Lusk, and Anthony Skjellum. *USING MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1994.
[25]William Gropp and Barry Smith. Chameleon parallel programming tools users manual. Technical Report ANL-93/23, Argonne National Laboratory, March 1993.
[26]Robert Harrison. Portable tools and applications for parallel computers. *Int. J. Quantum Chemistry*, 40, 1991.
[27].Shane Hebert, Walter Seefeld, and Anthony Skjellum. MPICH on the Cray T3E. Technical Report, Mississippi State University, March 1998.
`http://www.cs.sandia.gov.mpi/cray-t3e/mpich-t3e.html`
[28]Andrea Hudson and Alex Wang. Performance and Portability of the Message Passing Toolkit on CRAY J90 and CRAY T3E Systems. Proc. *Thirty-Ninth Semi-Annual Cray User Group Meeting*, San Jose, California, May 5-9, 1997.
[29]Alvin Leung, Manfred Morari, Anthony Skjellum, Steven Smith, and Charles Still. The Zipcode message-passing system. In Geoffrey Fox (editor), *Parallel Computing Works!* Morgan Kaufman, 1994.
[30]Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, April 1994.
[31]Message Passing Interface Forum. *MPI-1.1: A Message-Passing Interface Standard*, June 1995.
`http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html`
[32]Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*. July 1997.
`http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html`
[33]Wilfried Oed. Cray Research Massiv-paralleles Prozessorsystem CRAY T3E. Technical Report, Cray Research GmbH, November 1996
`http://www.zib.de/zibdoc/zib/mpp/t3e/T3E_OED.html`
[34]Peter S. Pacheco. *Parallel Programming With MPI*. Morgan Kaufmann, 1996.
[35]Parasoft Corp., Monrovia, CA. *Express User's Guide*, Release 3.2.5, 1992.
[36]Silicon Graphics, Inc. Performance of the Cray T3E Multiprocessor, 1998.
`http://www.sgi.com/t3e/performance.html`
[37]David Walker. Standards for message passing in a distributed memory environment. Technical Report, Oak Ridge National Laboratory, August 1992.

Authors Biographies

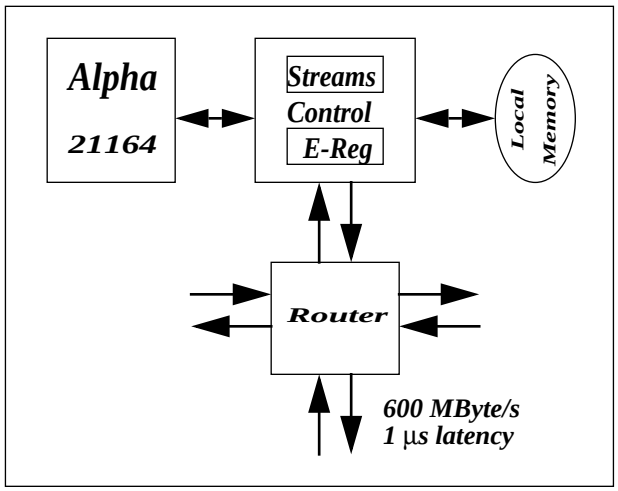
Hans-Hermann Frese is working as a senior system analyst and software engineer in the Computer Science Division of the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) in Berlin, Germany. He has been a member of the CUG Advisory Council since 1994 and the CUG Software Tools SIG Chair from 1995 through 1998. He is now the CUG Compilers and Libraries Focus Chair. Hans-Hermann's research areas include:

- High-performance parallel and distributed computing
- Programming languages and environments for supercomputers
- Software tools for parallel and distributed computing
- Performance analysis and prediction for high-performance computing.

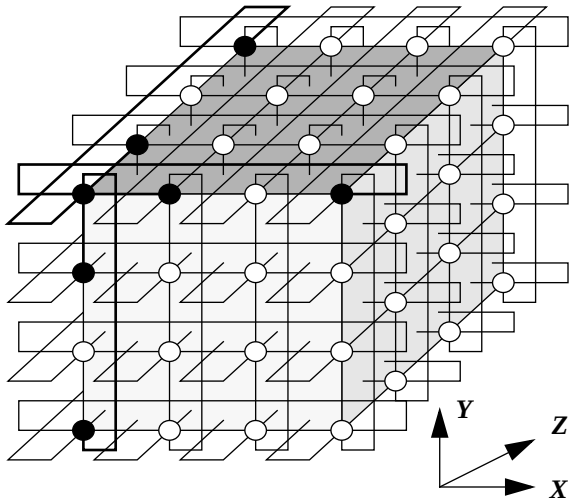
Harald Knipp is a student member at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB). He studies Physics at the Freie Universität Berlin. His research areas include message passing environments for distributed and high-performance parallel computing environments.

Appendix

A. Block Diagram of a Cray T3E Node



B. Cray T3E Interconnect Network



Source: Cray T3E Interprocessor Network in [33]

C. Supercomputer Configuration at ZIB