A platform independent communication library for distributed computing

Groen, D.; Rieder, S.; Grosso, P.; de Laat, C.; Portegies Zwart, S.

[Link to publication](#)

International Conference on Computational Science, ICCS 2010

A platform independent communication library for distributed computing

Derek Groen[a,b], Steven Rieder[a,b], Paola Grosso[b], Cees de Laat[b], Simon Portegies Zwart[a]

[a]*Leiden Observatory, Leiden University, Leiden, the Netherlands*
[b]*University of Amsterdam, Amsterdam, the Netherlands*

Abstract

We present MPWide, a platform independent communication library for performing message passing between supercomputers. Our library couples several local MPI applications through a long distance network using, for example, optical links. The implementation is deliberately kept light-weight, platform independent and the library can be installed and used without administrative privileges. The only requirements are a C++ compiler and at least one open port to a wide area network on each site. In this paper we present the library, describe the user interface, present performance tests and apply MPWide in a large scale cosmological N-body simulation on a network of two computers, one in Amsterdam and the other in Tokyo. © 2010 Published by Elsevier Ltd.

1. Introduction

A parallel application can run concurrently on multiple supercomputers provided one is able to coordinate the tasks between them and limit the performance overhead of the wide area communications. The advantage of using a distributed infrastructure lies in the enormous amounts of storage, RAM and computing performance it makes available. Distributed computing therefore allows us to solve large scale scientific problems [1]. Starting from the coupling of Intel Paragons over an ATM network [2] in the early 1990s, distributed parallel applications have become very popular.

An efficient method to program a parallel application is the Message Passing Interface (MPI [3]), a language-independent communication protocol that coordinates the computing tasks in parallel programs. MPI is often used for intra-site parallelization, but it can also be used for message passing in a distributed infrastructure. Prior efforts in the use of MPI on distributed infrastructures are abundant [4, 5]. With respect to N-body simulations Gualandris et. al. [6] have demonstrated that it is possible to use grid-enabled clusters of PCs connected via regular internet, grid middleware and MPICH-G2 [7]. However, the vast majority of MPI implementations requires all participating nodes to have a public IP address, which is generally undesirable for supercomputer environments for security reasons. Furthermore these implementations do not have built-in optimizations to fully exploit dedicated network circuits between supercomputers.

The lack of flexibility in deployment and link-specific optimizations of grid-oriented MPI implementations in distributed supercomputer environments led us to develop MPWide, a light-weight socket library specially aimed for

---

high-performance wide-area message passing between supercomputers. In this paper we present several performance results and apply MPWide to parallelize a large-scale cosmological N-body simulation across two supercomputers.

2. Related Work

Several grid message-passing libraries and frameworks have been developed with the intent to make distributed computing possible between sites that have restrictive firewall policies. PACX-MPI [8] is specifically geared for parallelization across sites and does not require compute nodes to have a public IP adress. Instead, it forwards inter-site communications through two forwarding demon processes on each site. Such a setup works reasonably well for applications that have been parallelized over multiple supercomputers using regular internet [9], but the two communication process restriction is less optimal when using multiple sites in a dedicated network environment. The Interoperable MPI (IMPI) [10] standard has also been designed to specifically facilitate execution across sites, but at the time of writing very few of the vendor-tuned implementations on supercomputers support IMPI. Also, IMPI requires the installation of a centralized and globally accessible server and does not support path-specific optimizations.

NetIbis [11] and PadicoTM [12] are two communication frameworks which are able to establich connections using bootstrap links, thus not requiring public IP addresses. However, PadicoTM also requires the use of a centralized rendez-vous node for bootstrapping, and thereby some means of centralized connectivity. Both Ibis [13] and NETIbis are sufficiently flexible to use in a restricted supercomputer environment, but introduce a communication overhead compared to regular socket communications. These libraries are therefore less suitable for high-performance message passing over dedicated inter-supercomputer networks.

3. Architecture of MPWide

3.1. Design

MPWide is a light-weight communication library which connects multiple applications on different supercomputers, each of them running with the locally recommended MPI implementation. It can be installed by a local user without administrative privileges, has a very limited set of software requirements, and the application programmer interface is similar to that of MPI.

MPWide has been designed to facilitate message passing between supercomputers and construct/modify custom communication topologies. The MPWide library is linked to the application at compile time and requires only the presence of UNIX sockets and a C++ compiler. MPWide provides an abstraction layer on top of regular sockets with methods to construct a communication topology, to adjust the parameters of individual communication paths and to perform message passing and forwarding across the topology. MPWide does not link against local MPI implementations, but can be used to combine multiple programs parallelized with MPI. Maintaining separate implementations for intra- and inter-site message passing makes it easier to specifically optimize and debug long-distance communication paths while relying on well-tested and vendor-tuned software for optimal intra-site communication performance.

During the development of MPWide, we have chosen to support multiple streams with a TCP-based protocol. This is a well-known and proven technique to improve network performance in the WAN [14], and provided the best wide area communication performance in our preliminary tests. In MPWide, each TCP stream is represented as a *channel*. A channel therefore provides a bidirectional connection between two ports on two hosts. On network paths where the use of parallel TCP streams provides a performance benefit, it is possible to use multiple channels concurrently on the same path. The message passing and forwarding functions in MPWide are designed to operate concurrently on multiple channels when needed.

Channels are locally defined at initialization and may be closed, modified and reopened at any time during execution. This allows us to alter the communication topology at run-time, e.g. to restart or migrate part of the MPWide-enabled application.

Once one or more communication channels have been established, the user can transfer data using the communication calls in the MPWide API. The user provides a listing of the channels used for each communication call by supplying a list of channel indices to the MPWide communication functions. Two example function signatures are shown in Fig. 1, and Table 1 provides an overview of the MPWide functionality. MPWide also supports several functions to modify properties of individual channels, such as the tcp buffer sizes, transmission chunk sizes and software packet pacing.

```
void MPW_SendRecv(char* SendBuf, long long int SendSize, char* RecvBuf,
      long long int RecvSize, int* Channels, int NumChannels);
void MPW_Cycle(char* SendBuf, long long int SendSize, char* RecvBuf,
      long long int RecvSize, int* SendChannels, int NumSendChannels,
      int* RecvChannels, int NumRecvChannels);
```

Figure 1: Full signatures of MPW_Cycle and MPW_SendRecv.

command name	functionality
MPW_Barrier()	Synchronize between two ends of the network.
MPW_Cycle()	Send buffer over one set of channels, receive from other.
MPW_DSendRecv()	Send/receive buffers of unknown size using caching.
MPW_Init()	Set up channels and initialize MPWide.
MPW_Finalize()	Close channels and delete MPWide buffers.
MPW_Recv()	Receive a single buffer (merging the incoming data).
MPW_Relay()	Forward all traffic between two channels.
MPW_Send()	Send a single buffer (splitted evenly over the channels).
MPW_SendRecv()	Send/receive a single buffer.

Table 1: List of MPWide function calls. In addition to this list, each function has a variant call with a prefix 'P' which operates on one send and/or recv buffer per channel.

Since message passing can be performed over multiple channels in parallel, it is possible to communicate with multiple hosts simultaneously. For example, the user can scatter data across multiple processes with a single `MPW_Send()` call or gather data from multiple hosts with a single `MPW_Recv()`. Each function has a variant call with a prefix 'P' (e.g. `MPW_PSend()`) which takes an array of buffer pointers instead of one buffer pointer. These functions use one pointer for each channel, and the size of each seperate buffer can be explicitly specified. Consequently, `MPW_PSend()` or `MPW_PRecv()` functions can be used to respectively scatter and gather data which is not equally distributed across the hosts.

3.2. Implementation

We implemented MPWide using C++ in combination with GNU C sockets and POSIX threads [15]. MPWide creates and destroys threads on the fly whenever a communication call is made. With modern kernels, the overhead of creating and destroying threads is very small, and using MPWide we were able to reach nearly 10Gbps with message passing tests over local networks. For longer network paths, the high latency results in an even smaller relative overhead for thread creation/destruction. We have considered creating threads only at startup and managing them at runtime, but these modifications would increase the code's complexity and only offer a limited performance benefit, as threading overhead plays a marginal role in wide area communication performance.

Aside from the ability to hardwire each communication, the library also supports a number of customizable parameters:

- number of concurrent streams for each communication call

- data feeding pace of sending and receiving.

- TCP window size for each individual socket

The maximum number of streams and the TCP window size may be restricted by local system policies. However, we were able to use up to 128 streams on most systems without requiring administrative rights. The code has been packaged and is publicly available at
`http://castle.strw.leidenuniv.nl/software/mpwide.html`.

4. Benchmarking MPWide

We performed a series of tests on the Dutch ASCI Supercomputer 3 (DAS-3 [1]) to measure the performance of MPWide between two sites, one at the University of Amsterdam and one at the Delft University of Technology. Both sites are connected to regular internet with a 1 Gbps duplex interface. A detailed specification can be found in columns 2 and 3 of Table 2. We performed the tests using the system default TCP window sizes (16 kB send and 85 kB recv).

Each run consists of 100 two-way message exchanges, where we record the average throughput and the standard error. First we performed 8 different runs using messages of 8 MB and respectively 1, 2, 4, 8, 16, 32, 64 and 128 TCP streams in parallel. We then repeated the same series of runs with message sizes of 64 and 512 MB.

	DAS-3 Ams	DAS-3 Delft	Huygens	Cray
Architecture	AMD Opteron	AMD Opteron	IBM Power6	Cray XT4
Number of nodes	41	68	104	740
Cores per node	4	2	32	4
CPU frequency [GHz]	2.2	2.4	4.7	2.2
Memory per core [GB]	1	2	4/8	2

Table 2: Specifications of the two DAS-3 sites used in the MPWide experiments, and the Huygens supercomputer in Amsterdam and the Cray XT4 supercomputer in Tokyo which are used for experiments in Section 5.

4.1. Results

The results of our tests on the DAS-3 are found in Fig. 2. Although the tests were performed over regular internet, the fluctuations in our measurements are limited. When exchanging messages of 8 MB size, we obtain the best performance using a single stream, as the use of additional streams results in a lower average performance as well as an increased fluctuation in performance. This is caused by the fact that message passing performance over multiple streams is limited by the slowest streams. For larger message sizes, however, using a single stream does not result in an optimal performance. Instead, we find that the best results are obtained using 8 streams (for 64 MB) to 32 streams (for 512 MB). Although a high peak performance is obtained when using 64 or more streams, the sustained performance is lower because the excess streams can cause network congestion.

5. Testing Performance in a Production Environment

We originally developed MPWide to manage the long-distance message passing in the CosmoGrid project [16]. CosmoGrid is a large-scale cosmological project which aims to perform a dark matter simulation of a cube with sides of 30 Mpc using supercomputers on two continents. In this simulation, we use the cosmological Λ Cold Dark Matter model [17] which defines a constant fraction of the overall energy density for dark energy to model the accelerating expansion of the universe. We apply this model to simulate the dark matter particles with a parallel tree/particle-mesh N-body integrator, GreeM [18]. This integrator can be run either as a single MPI application, or as multiple MPI applications on different supercomputers. In the latter case, the wide area communications are performed using MPWide. We use GreeM to calculate the dynamical evolution of 2048^3 (~ 8.590 billion) particles over a period of time from redshift $z = 65.35$ to $z = 0$. More information about the parameters used and the scientific rationale can be found in [16].

Before the simulation is launched, the initial condition is decomposed in slices for each site, and in blocks within that slice for each process. Each block contains an equal number of particles but may vary in volume. A simulation process loads one block during startup, and calculates tree and particle mesh force interactions at every step. These force calculations require the exchange of particles with neighbouring processes (and sites, see Fig.3) as well as the exchange of mesh cells. In addition, a number of smaller communications are performed to balance the load across all processes.

We have used GreeM together with MPWide in two different experiments. One experiments consists of a full-length simulation of a limited scale (256^3 particles), and one run consists of a limited part of the production simulation described earlier.

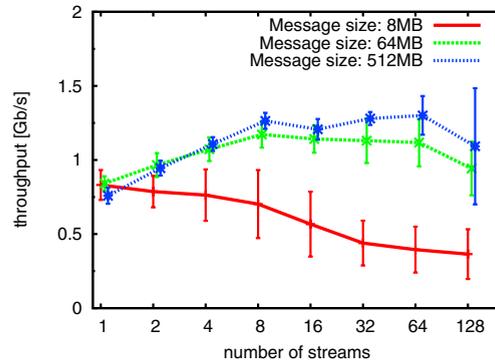
[1]DAS-3: http://www.cs.vu.nl/das3/

Figure 2: Measured throughput in Gbit per second as a function of the number of communications streams used between the DAS-3 site in Amsterdam and the DAS-3 site in Delft. The throughput is given for runs with 1 to 128 threads and message sizes of resp. 8, 64 and 512 MB. This test was performed over regular internet, and the theoretical bandwidth limit of the network interfaces is 2 Gbit per second.
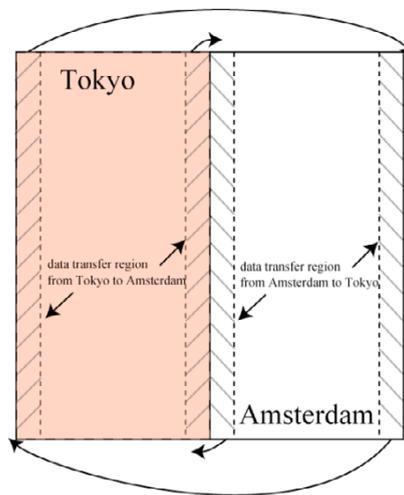

Figure 3: Data decomposition overview of the CosmoGrid simulation when run on two supercomputers [16].

5.1. Test Experiment

We have performed two test runs, of which each one uses a different infrastructure. Both runs were performed over two sites, with 30 calculation processes and one communication process per site. The run was performed using the DAS-3 sites in Amsterdam and Delft, of which the specification can be found in Table 2.

5.1.1. Test Results

The performance results of our test simulation on the DAS-3 can be found in Fig. 4. Here we find that the simulation performance is dominated by calculation, with a communication overhead less than 20 percent of the overall wallclock time throughout the run. As we used regular internet for the wide area communication, our simulation performance is subject to the influence of background network traffic. The two performance dips which can be found around step 1300 and 1350 are most likely caused by incidental increases in background traffic.
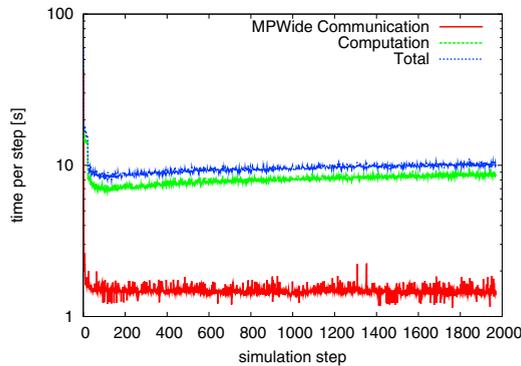
Figure 4: Measured wall-clock time spent (in log-scale) on each simulation step for a 256^3 particle test run on the DAS-3 between Amsterdam and Delft. The full-length run was performed using 62 cores, with 30 cores residing on each supercomputer and 2 cores used for communication only. The top dotted line indicates total time spent, the dashed line indicates time spent on calculation and the bottom solid line represents time spent on communication with MPWide.
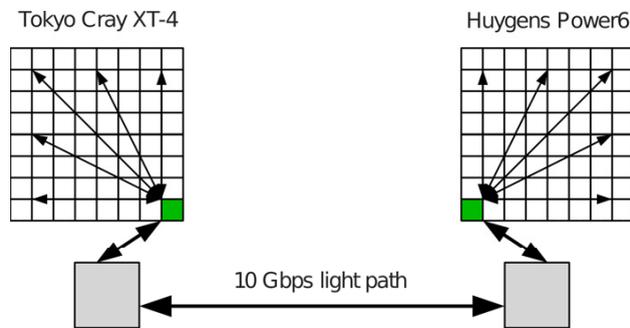

Figure 5: Network topology example of the CosmoGrid simulation when run on two supercomputers, one in Amsterdam, the Netherlands and one in Tokyo, Japan. Data transfers within the local supercomputer are performed using MPI (thin arrows), whereas other communications are performed using MPWide (thick arrows). The communication nodes (indicated by the gray boxes) reside outside of the MPI domains and contain user-space port forwarders. Before the data is transferred to the communication node, it is gathered on a central process on the local supercomputer (indicated by the green boxes).

5.2. Production

We have performed a large-scale calculation between supercomputers in Amsterdam and Tokyo to measure the performance of the code when it is used for production. We have used the IBM Power6-based Huygens supercomputer at SARA, Amsterdam, the Netherlands and the Cray XT4 supercomputer at the National Astronomical Observatory of Japan in Tokyo, Japan. The technical specifications for both supercomputers can be found in Table 2.

To exchange data between the supercomputers in Amsterdam and Tokyo we reserved and used a 10 Gbps dedicated light path in the GLIF network[19], which has a round trip time of 273 milliseconds. The run between Huygens and the Tokyo Cray was performed using 64 concurrent TCP streams. A detailed overview of the communication topology during the simulation is given in Fig.5. Each of the supercomputers was equipped with one specialized communication node. These nodes are each connected to the local supercomputer network and are linked together by the 10 Gbps light path. MPWide is used to transfer the locally gathered data to the communication node, forward it to the other site using the light path, and finally to deliver the data to the remote MPI simulation.

The production-sized run was performed on 752 cores in total. The topology of this run was asymmetric, using 500 cores on Huygens and 250 cores on the Cray for calculation. The full run lasted just under 12 hours, during
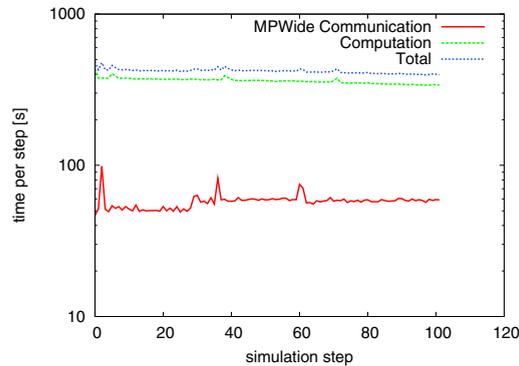
Figure 6: Measured wall-clock time (in log-scale) for each simulation step for a partial 2048^3 particle run. The run, which uses some adjusted TCP settings, was performed using 750 cores, with 500 cores used on Huygens and 250 cores used on the Tokyo Cray. An explanation of the lines can be found in the caption of Fig. 4. The time spent on MPWide communication also includes local communication overheads for mesh and interaction tree exchanges.

which we performed 102 simulation steps. The performance results of this run can be found in Fig. 6. In this full-scale run, the calculation time dominated the overall performance, and was slightly higher at startup and during steps where snapshots were written. The communication performance is generally constant with three performance dips throughout the run. These dips are caused by periods of packet loss on the light path.

Also, the communication time increases slightly after step 30 in the simulation. The increase is most likely caused by a change in TCP buffering sizes by the local system, although we did not track this directly. Overall, the total communication time per step was between 50 and 60 seconds for most of the simulation, and constituted about one eighth of the total execution time.

6. Conclusions and Future Work

We present MPWide, a communication library to perform message passing between supercomputers. MPWide provides message passing that is intrinsically parallelized, and can be used for performing high-performance computing across multiple supercomputers. The library allows for customization of individual connections and has a light-weight design, which makes it well-suited for connecting different supercomputer platforms. We have shown performance results of MPWide between two sites and applied MPWide to combine two MPI applications into a very large parallel simulation across a wide area compute infrastructure. During our tests, we were able to obtain communication speeds in excess of 1Gbps between two DAS-3 sites. In addition, we were able to perform an N-body simulation across two continents with 2048^3 particles. During this simulation, about one eighth of the execution time was spent on communications.

Given that the parallel application is sufficiently scalable (which is the case for the N-body integrator used in this work), MPWide can be used to efficiently parallelize production applications across multiple supercomputers.

Acknowledgments

[1] A. Hoekstra, S. Portegies Zwart, M. Bubak, P. Sloot, Towards distributed petascale computing, Petascale computing: Algorithms and applications, Vol. 1, D.A. Baker, 2007, Ch. 8, pp. 147–164.

[2] T. Pratt, L. Martinez, M. Vahle, et al., Sandia's network for supercomputer '96: Linking supercomputers in a wide area asynchronous transfer mode (atm) network, Tech. rep., Sandia National Labs., Albuquerque, NM (United States) (1997).

[3] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, S. Huss-Lederman, MPI: The Complete Reference, MIT Press, Cambridge, MA, USA, 1995.

[4] R. W. Hockney, The communication challenge for mpp: Intel paragon and meiko cs-2, Parallel Computing 20 (3) (1994) 389 – 398.

[5] S. Manos, M. Mazzeo, O. Kenway, P. V. Coveney, N. T. Karonis, B. R. Toonen, Distributed mpi cross-site run performance using mpig, in: HPDC, 2008, pp. 229–230.

[6] A. Gualandris, S. Portegies Zwart, A. Tirado-Ramos, Performance analysis of direct n-body algorithms for astrophysical simulations on distributed systems., Parallel Computing 33 (3) (2007) 159–173.

[7] N. T. Karonis, T. B, I. Foster, Mpich-g2: A grid-enabled implementation of the message passing interface, Journal of Parallel and Distributed Computing 63 (5) (2003) 551 – 563, special Issue on Computational Grids.

[8] E. Gabriel, M. M. Resch, T. Beisel, R. Keller, Distributed computing in a heterogeneous computing environment, Vol. 1497 of Lecture Notes in Computer Science, Springer, 1998.

[9] C. A. Stewart, R. Keller, R. Repasky, M. Hess, D. Hart, M. Muller, R. Sheppard, U. Wossner, M. Aumuller, H. Li, D. K. Berry, J. Colbourne, A global grid for analysis of arthropod evolution, in: GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, IEEE Computer Society, Washington, DC, USA, 2004, pp. 328–337.

[10] W. George, Impi: Making mpi interoperable, Journal of Research - National Institute of Standards and Technology 105 (2000) 343–428.

[11] O. Aumage, R. Hofman, H. Bal, Netibis: an efficient and dynamic communication system for heterogeneous grids, in: CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2, IEEE Computer Society, Washington, DC, USA, 2005, pp. 1101–1108.

[12] A. Denis, C. Perez, T. Priol, PadicoTM: An open integration framework for communication middleware and runtimes, Future Generation Computer Systems 19 (4) (2003) 575–585.

[13] R. V. van Nieuwpoort, J. Maassen, G. Wrzesinska, R. Hofman, C. Jacobs, T. Kielmann, H. E. Bal, Ibis: a flexible and efficient Java based grid programming environment, Concurrency and Computation: Practice and Experience 17 (7-8) (2005) 1079–1107.

[14] L. Qiu, Y. Zhang, S. Keshav, On individual and aggregate tcp performance, in: Network Protocols, 1999. (ICNP '99) Proceedings. Seventh International Conference on, 2002, pp. 203–212.

[15] F. Mueller, A library implementation of posix threads under unix, in: In Proceedings of the USENIX Conference, 1993, pp. 29–41.

[16] S. Portegies Zwart, T. Ishiyama, D. Groen, K. Nitadori, J. Makino, C. de Laat, S. McMillan, K. Hiraki, S. Harfst, P. Grosso, Simulating the universe on an intercontinental grid of supercomputers, IEEE Computer (accepted)arXiv:1001.0773.

[17] A. H. Guth, Inflationary universe: A possible solution to the horizon and flatness problems, Physical Review D (Particles and Fields) 23 (1981) 347–356.

[18] T. Ishiyama, T. Fukushige, J. Makino, GreeM : Massively Parallel TreePM Code for Large Cosmological N-body Simulations, accepted by Publications of the Astronomical Society of JapanarXiv:0910.0121.

[19] L. Smarr, T. A. DeFanti, M. D. Brown, C. de Laat, Special section: igrid 2005: The global lambda integrated facility, Future Generation Computer Systems 22 (8) (2006) 849 – 851.