

5-17-01

---

**PERFORMANCE COMPARISON OF A MATRIX SOLVER  
ON A HETEROGENEOUS NETWORK  
USING TWO IMPLEMENTATIONS OF MPI:  
MPICH AND LAM**

---

BY:  
Jennifer Phillips

MENTOR:  
Jerrold M. Housner

RESEARCH AND TECHNOLOGY GROUP  
STRUCTURES DIVISION  
*Computational Structures Branch*

## ABSTRACT

---

Two of the current and most popular implementations of the Message-Passing Standard, MPI, were contrasted: MPICH by Argonne National Laboratory, and LAM by the Ohio Supercomputer Center at Ohio State University. A parallel skyline matrix solver was adapted to be run in a heterogeneous environment using MPI. The Message-Passing Interface Forum was held in May 1994 which lead to a specification of library functions that implement the message-passing model of parallel communication. LAM, which creates it's own environment, is more robust in a highly heterogeneous network. MPICH uses the environment native to the machine architecture. While neither of these free-ware implementations provides the performance of native message-passing or vendor's implementations, MPICH begins to approach that performance on the SP-2. The machines used in this study were: IBM RS6000, 3 Sun4, SGI, and the IBM SP-2. Each machine is unique and a few machines required specific modifications during the installation. When installed correctly, both implementations worked well with only minor problems.

## INTRODUCTION

---

With the current downsizing and new philosophy of "faster, better, cheaper" companies can no longer afford large powerful machines like the CRAY, yet they still require the computing power provided by such systems. The current direction of the computational sciences is to utilize the power of the machines already in use by engineers. Workstations have given engineers powerful and accurate tools with which to design aircraft such as the Boeing 777. These same machines are also giving engineers the ability to perform the large scale computations which were previously done on large machines.

A single workstation could spend days slowly grinding though a large computation, but with the use of parallel communications, many workstations can be linked together and provide the power of a single large computer. This concept has lead to many types communication software facilitating the production of massively parallel machines. The most common types of communication are MPL, the message passing library for the IBM machines, and Parallel Virtual Machine or PVM which was designed as a message passing library for heterogeneous networks. PVM has recently been used in a parallelized Navier-Stokes CFD code at McDonnell Douglas. With the success and development of such software, a decision was made to standardize the communication protocol.

In May 1994, the first conference was held to create a new standard for message passing. This forum produced the Message-Passing Interface Standard [Ref. 1] or MPI-1. The best attributes of the existing message passing libraries were assessed and combined. MPI-1 is currently a standard by which to create message passing libraries. It in itself is not software; it is left to the implementors to write the code. While this standard currently contains message passing information and control, I/O is being considered for MPI-2, and dynamic process control (e.g. MPI\_Spawn) is still under controversy. The beauty of the standard is that programs written in MPI can be used on any architecture and with any implementation.

There are currently about five implementations of MPI freely available (see resource list). The two most common and best supported are MPICH and LAM. These two implementations will be profiled and contrasted in the following. The machines used in this study were:

- IBM RS6000
- (3) Sun 4
- SGI Indigo
- IBM SP-2

These machines were accessed from a Macintosh IIfx using MacX.

## ◆ IMPLEMENTATIONS

---

Both implementations require the use of the remote shell command, 'rsh,' and thus require all machines in the available network, including the root machine, to appear in the .rhosts file of the user.

### ◆ LAM VERSION 5.2

The Ohio Supercomputer Center's implementation, Local Area Multicomputer or LAM (which is a subset of the greater Trollius system), creates a similar environment to PVM. LAM is a message passing library with it's own environment. LAM's implementation of MPI amounts to a separate set of libraries that interact with LAM's environment. To run an MPI program in LAM it needs to be compiled with a FORTRAN/C compile script provided by Ohio State. The environment must then be set up by "starting the engine" or running the LAM deamon on all the machines in the desired network.

### ◆ OPERATION

Once the engine is running LAM is fairly simple to use. However, for heterogeneous networks the executable must lie in the user's path. The program is run with a command line execution that includes the script executable (mpirun), options, the nodes which it will be run on, and the program executable. The program name is assumed to be the same on all machines. There is no way to easily deal with different executable names or paths. For a heterogeneous environment any I/O to the program must have a specified path name or be located in the directory from which LAM was booted. (For the root computer this will be the directory from which 'lamboot' was run. For other computers in the network, it is the user's home directory.)

### ◆ INSTALLATION

The installation is pretty straightforward as laid out in the Installation Guide [Ref. 2]. An environment variable must be set up to define the location of the executables and this location must then be placed in the user's path. This location is *different* from the location of the source code, since the source code may be eliminated once LAM is properly built. The last thing required before running the 'make' is linking the 'config' file to the proper architecture configuration file.

The only problem encountered during the installation was the necessary addition of the '-Bstatic' option to the C compiler (cc) and the FORTRAN compiler (f77) in the 'config' file on 'sunny.larc.nasa.gov' (Sun4) and 'csmsun.larc.nasa.gov' (Sun4). These machines require this option as a result of a problem with their shared libraries.

For a permanent network installation LAM should be installed in the path `/usr/local/lam` or `/usr/local/lam5.2` etc. In order to do this, the `TROLLIUSHOME` environment variable must be set to this path as well as the home variable in the `config` file. Once set, run:

**make install**

and the source directory may be deleted.

## ◆ MPICH VERSION 1.0.10

MPICH uses an environment already installed on the machine which gives rise to its chameleon nature and the CH in its name. MPICH relies on the environment already created on various machines, such as p4 on workstation and MPL/EUI on IBM machines. Argonne National Laboratory has created libraries to make MPI take advantage of the native message passing routines already on the computers. To run a MPI program with MPICH an appended makefile (created by MPICH) is required to initialize certain environment variables.

## ◆ OPERATION

MPICH is also fairly easy to run. It also uses a script executable (`mpirun`) followed by options and the program executable. MPICH does not require the listing of nodes/computers to be run on in the command line, however it does require the `-np` option at a minimum to specify the number of processes. When enlisted, MPICH looks for all the machines with the same architecture as the root computer (these are listed in a file in MPICH) and the processes are dispersed among them. In order to run on a heterogeneous network, the process file or procgroup (`PI####`) needs to be created by the user. This process file contains the computer's network name, the number of processes assigned to that computer and the full path name of the executable, as well as the user's login name on heterogeneous machines. Since the full path name of the program is specified, the program names and paths may vary on other machines without being placed in the user's path.

A generic procgroup file was created containing all of the machine names available; unused machines were then commented out as necessary for each machine.

## ◆ INSTALLATION

Installation of MPICH is also fairly simple; see the MPICH Installation Guide [Ref. 3]. MPICH supplies a configuration script, `configure`, that can be run with or without flags that specify options such as the architecture, device and C compiler used. This `configure` script creates the makefiles, which simply need to be built. The following variations were made for the workstations used. The IBM RS6000, borg-07, is a stand alone workstation and does not contain the MPL/EUI libraries, thus the device and communications protocol used were p4. On the SP-2, the `configure` was unable to successfully use the `mpCC` or C++ compiler, so the `-cc=mpcc` option was used to specify the C compiler. This had no effect to subsequent programs since MPICH is written in C and the sparse solver is written in FORTRAN. Finally, the two Sun4 workstations, sunny and csmsun, required a modification to their configuration files. MPICH requires use of the shared libraries on these Suns so the static or non-shared library option was not successful (i.e. `configure -cc="cc -Bstatic" -fc="f77 -Bstatic"`). An additional library reference is needed for the shared libraries. There is no specific flag to specify the libraries for the `configure` file, therefore the script was directly modified and the shared libraries were added to the primary list of parameters (i.e. `LIBS=-lc.1.8.1` and `-lc.1.9.1`, respectively for sunny and csmsun). In addition, subsequent makefiles required this option as well.

For a permanent network installation MPICH should be installed in the path /usr/local/mpi or /usr/local/mpi-1.0.10 etc. This installation is performed by the following command:

```
make install PREFIX=/usr/local/mpi-1.0.10
ln -s /usr/local/mpi-1.0.10 /usr/local/mpi
```



## THE SPARSE SOLVER

---

The matrix solver used for this study was a vectorized skyline sparse solver written by Majdi A. Baddourah. This particular solver has been tested in various forms including PVM and MPL on the SP-2. For previous benchmarks and performance see the Computational Structures Branch Web page (see resource list).

Since MPI's subroutines are similar to existing parallel communicators, few changes had to be made to the solver. For example, the standard send FORTRAN syntax is provided below for the native message-passing library of the IBM machines (MPL), Parallel Virtual Machine (PVM), and MPI.

```
MP_SEND( outgoing_message, msg_length(bytes), destination, data_type )
PVMFINITSEND( encoding, error_code )
PVMFPACK( outgoing_message, first_element#, msg_length(items), stride, error_code )
PVMFSEND( task_identifier_of_destination_process, msg_tag, error_code )
MPI_SEND( outgoing_message, msg_length(items), data_type, destination, msg_tag,
          comminucator, error_code )
```

MPI is a combination and reordering of the other two, allowing the simplicity of MPL with the heterogeneous capability of PVM. The main program was edited to reflect MPI's initialization and finalize routines; MPI\_Init()<sup>1</sup>, MPI\_Comm\_Rank()<sup>2</sup>, MPI\_Comm\_Size()<sup>3</sup>, and MPI\_Finalize()<sup>4</sup>. All other communication references were dealt with as subroutines. One other change was made to the factorization and back solving subroutines. MPL requests the number of bytes being sent while MPI requires the number of items, thus variables such as 'nbyt' were essentially divided by the number of bytes per item to reflect the number of items only.

The send subroutine was modified to use the standard MPI send, MPI\_Send()<sup>5</sup>. Other options for different types of sending routines were considered, but deemed unnecessary. One of these possibilities was a synchronous send, which will not complete until not only the message buffer is safe to be reused but also the matching receive has been posted. The standard send is a blocking send, where the send will not complete until the message buffer is able to be reused. This standard send was also the fastest send, especially for large messages. This modification was a little tricky since the send and receive subroutine were called for both integer and real data types. MPL accepts a range of numbers to specify a certain data type, e.g. the integers 1 to 100000 represent real numbers and larger integers represent integers. The original data type was used as the message tag and then changed to an MPI data type, e.g. MPI\_Real<sup>6</sup>. Using this message tag will also prevent mismatches in send and receive pairs.

---

<sup>1</sup> MPI Standard page 196 line 20

MPI\_INIT( ierror )

<sup>2</sup> MPI Standard page 142 line 5

MPI\_COMM\_RANK( comminucator, rank, ierror )

<sup>3</sup> MPI Standard page 141 line 28

MPI\_COMM\_SIZE( comminucator, size, ierror )

<sup>4</sup> MPI Standard page 196 line 34

MPI\_FINALIZE( ierror )

<sup>5</sup> MPI Standard page 16 line 26

MPI\_SEND( buffer, nelements, datatype, destination, tag, communicator )

<sup>6</sup> MPI Standard page 17 line 12

Fortran: REAL

The global sum subroutine was modified to make use of `MPI_Allreduce()`<sup>7</sup>, which not only performs a user defined function but also sends the result to all of the machines in the network. Here, the user defined function, vector addition, was slightly modified to the form expected by MPI. This modification only involved reordering the arguments.

The two communication ring subroutines, integer and real data types, were initially modified to use the `MPI_Sendrecv()`<sup>8</sup> routine. `MPI_Sendrecv()` will send and receive the same buffered information in either direction, which should prevent deadlock. Deadlock occurs when one machine is attempting to send and the receiving machine is also attempting to send, thus not allowing the completion of either send call. Unfortunately, this created problems and was abandoned.

Finally, the subroutine used for timing the program, or profiling, was changed from a C subroutine using `gettimeofday()` to `MPI_Wtime()`<sup>9</sup>. `MPI_Wtime()` returns wall-clock time in seconds similar to `gettimeofday()`.

The makefiles for the solver are generally straight forward, the header information for both LAM and MPICH was either copied or generated by the software. Initially, the makefiles were not finding the MPI library, this was temporary fixed by editing the environmental variable `LIB_PATH` to reflect the location of `libmpi.a`. This is not necessary, however, if the makefiles are created with the correct path for each MPI implementation. Also, LAM 5.2's include file for FORTRAN, `mpif.h`, was originally written in C, this facilitated the need for extra lines in the makefile, i.e. the preprocessor `cpp` was required of all files containing the include file. This was subsequently changed after an inquiry to the implementors of LAM. This change may now be included as a patch and will most likely appear in the next version to be released.



## PERFORMANCE TESTING

There have been previous studies in the performance of MPI as compared to other parallel libraries. "Performance Comparison of MPL, MPI, and PVMe" [Ref. 4] compared the performance of PVMe, MPICH, MPI-F, and MPL. As expected MPL performed the best, but IBM's vendor's version of MPI, MPI-F, practically matched MPL performance. The free-ware version of MPI, MPICH, was a close second followed by PVMe. Also, "Some early performance results with MPI on the IBM SP1" [Ref. 5] found MPI-F to closely match MPL and MPICH followed closely behind.

The solver was run for various combinations of processors using LAM and the p4 directories in MPICH. The SP-2 was used as the primary control machine, since it was the original source of the solver's program with MPL. For each matrix tested, the solver was run on the SP-2 using, MPL, MPICH, and LAM for two to six processors or nodes. In addition various other combinations of machines were used for each of the matrices up to the entire network of all six machines available. The matrices tested were a simple beam consisting of 24 equations, a test matrix of 840 equations, a plate with a hole in it with 1,802 equations and a model of the High Speed Civil Transport with 16,152 equations.

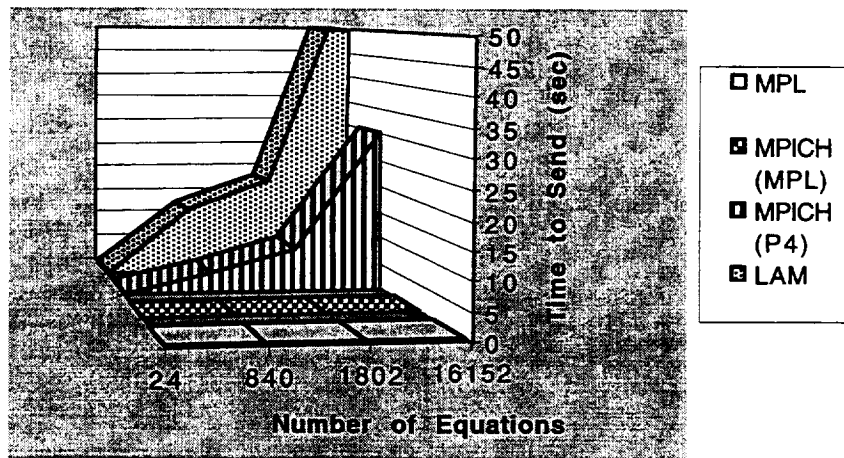
As a basis for a comparison, all four matrices were run on the SP-2 and compared in Figure 1. The send time in the factorization is compared for the solver running with MPL, MPICH using the native MPL, MPICH using p4, and LAM. The SP-2 was tested with p4 because, for this particular network p4 interacted the most consistently with the other machines. Figure 1 shows MPICH with MPL a close second to MPL itself.

<sup>7</sup> MPI Standard page 122 line 14     `MPI_ALLREDUCE( sendbuffer, recvbuffer, nelements, datatype, operation, communicator )`

<sup>8</sup> MPI Standard page 57 line 4     `MPI_SENDRECV( sendbuffer, sendnelements, sendtype, destination, sendtag, recvbuffer, recvnelements, recvtype, source, recvtags, comincator, status )`

<sup>9</sup> MPI Standard page 195 line 19     `FUNCTION MPI_WTIME()`

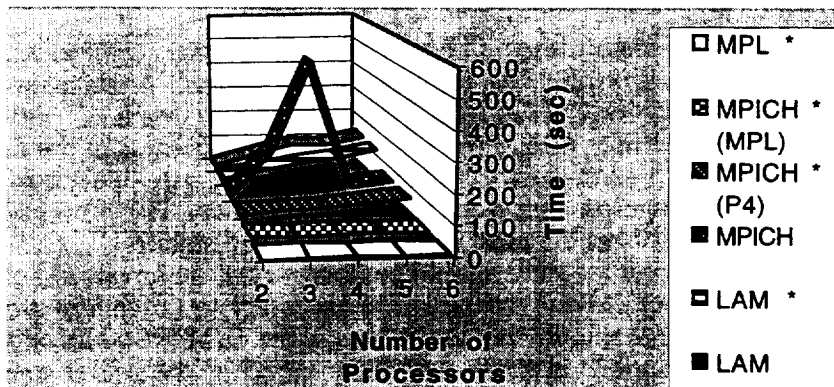
**Figure 1. Send Times on Poseidon**



Please note the graph is not to scale and is a bit skewed. Figure 1 does not yet suggest scalability as all graphs are climbing through the increase in number of equations, however, it has been shown that MPICH using native libraries will become as scalable as the native libraries themselves.

Figure 2 demonstrates the total factor time for the SP-2 as well as an average of many other heterogeneous configurations. This figure demonstrates the increase in communication time in the heterogeneous network. This is primarily due to the relative speeds and memory of each machine as the time varies with the number of processors and not with the degree of heterogeneity.

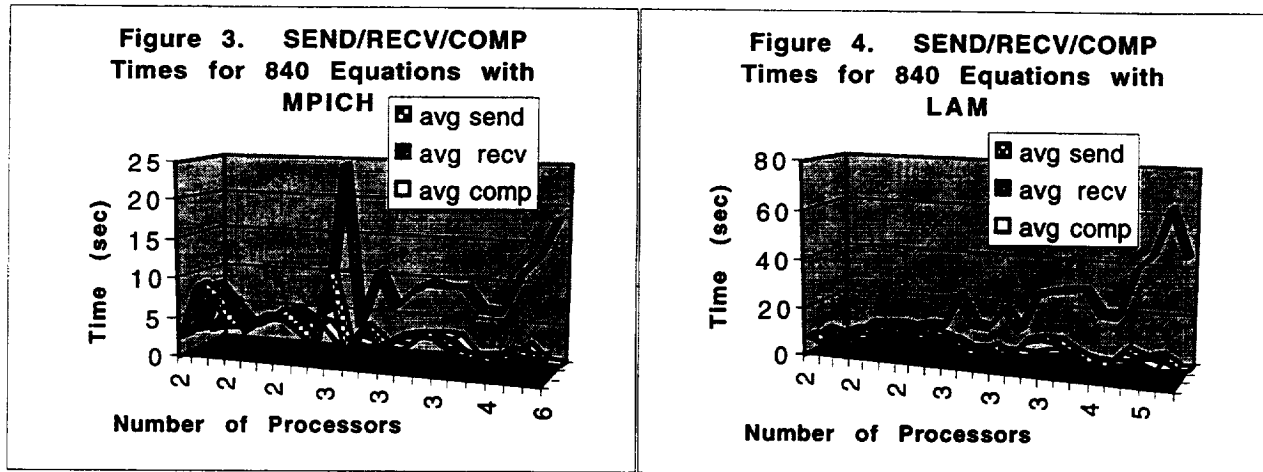
**Figure 2. Factor Times for 1802 Equations**



The largest peak stems from a largely heterogeneous network using MPICH and p4. This curve drops off quickly because this network could not handle larger matrices thus, the last data points are for the SP-2 alone and not an average of many networks.

As mentioned above, the time is dependent on the individual machine and thus the over all time can vary great depending on the machines in the network. Figure 3 shows the send, receive, and compute time for the 840 equations. The general trends are for send time to decrease with an increasing number of nodes which can be explained by the decreasing message size. The compute time also decreases with increasing nodes, which was expected since the relative work load is

decreasing. Finally, the receive time increases dramatically. This is mostly likely due to down time waiting for other machines in the network.



The large peaks represent very heterogeneous networks, i.e. at least three different architectures, and usually contain a Sun. It is also interesting to note that the order of networks configurations is the same in both Figure 3 and 4. Figure 4 lacks the large peaks found in Figure 3, but the overall time is much greater. The length of the send and receive times was also dependent on the degree on heterogeneity and the speed of the machine being sent to or receive from. The compute time remained relatively constant per machine, differences resulting from load on the machine. For large matrices, i.e. the high speed civil transport, LAM's performance begins to approach MPICH's performance using p4, however, neither approaches MPICH using native message-passing.

## ◆ CONCLUSIONS

The choice of implementation should be based on user preferences, i.e. what type of control the user is more comfortable with and the importance of time and performance. LAM is more robust in a heterogeneous environment than MPICH, however, MPICH provides much better performance than LAM. LAM may also allow dynamic process control in the future which is not available with MPICH. For a semi-homogeneous environment, e.g. cluster of IBM RS6000's, MPICH using the native message-passing would be the best choice. For a program developer on a heterogeneous network, LAM offers the most ease of control and stability.

Further testing with larger matrices should be done. This, however, should not be done on the local Suns, since they do not provide the hardware necessary to efficiently run them. A similar performance to that of the SP-2 should be attainable on the IBM RS6000 cluster. In addition, optimization can be done to further increase the performance of the solver using MPI. The X windows interfaces, e.g. MPE and Xmpi, provide an aide to this task.



## ◆ REFERENCES

- [1] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*. Computer Science Dept. Technical Report CS-94-230, University of Tennessee, Knoxville, Tennessee, 5 May 1994.
- [2] GDB. *LAM for Fortran Programmers. (LAM Installation Guide)* Ohio Supercomputer Center, The Ohio State University, July 20, 1994  
Ohio LAM version 5.2
- [3] Patrick Bridges, Nathan Doss, William Gropp, Edward Karrels, Ewing Lusk, Anthony Skjellum. *User's Guide to MPICH, a Portable Implementation of MPI. (Installation Guide to MPICH)* Argonne National Laboratory, 5 May 1995.  
MPICH version 1.0.10
- [4] Bill Saphir and Sam Fineberg. *Performance Comparison of MPL, MPI, and PVMe*. NAS Scientific Consulting Group, 21 October 1995.
- [5] William Gropp and Ewing Lusk. *Some early performance results with MPI on the IBM SP1 - EARLY DRAFT*. Argonne National Laboratory, 5 June 1995.
- [6] Ed Karrels and Ewing Lusk. *Performance Analysis of MPI Programs*. Argonne National Laboratory, (1995).
- [7] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1994.

## ◆ FTP SITES

MPICH source code [info.mcs.anl.gov/pub/mpi/mpich.tar.Z](http://info.mcs.anl.gov/pub/mpi/mpich.tar.Z)  
CHIMP source code [ftp.epcc.ed.ac.uk/pub/chimp/release/chimp.tar.Z](http://ftp.epcc.ed.ac.uk/pub/chimp/release/chimp.tar.Z)  
LAM source code [tbag.osc.edu/pub/lam/lam52.tar.Z](http://tbag.osc.edu/pub/lam/lam52.tar.Z)

Test Code repository [info.mcs.anl.gov/pub/mpi-test](http://info.mcs.anl.gov/pub/mpi-test)

## ◆ MAILING LISTS

<a href="mailto:mpi-bugs@mcs.anl.gov">mpi-bugs@mcs.anl.gov</a>	<i>MPICH bugs and information (William Gropp)</i>
<a href="mailto:trollius@tbag.osc.edu">trollius@tbag.osc.edu</a>	<i>LAM bugs and information (Raja Doaud)</i>
<a href="mailto:mpi-comm@cs.utk.edu">mpi-comm@cs.utk.edu</a>	<i>the MPI Forum discussion list</i>

## ◆ NEWS GROUP

comp.parallel.mpi

*MPI newsgroup*

## ◆ WWWEB PAGES

Message-Passing Interface

<http://WWW.ERC.MsState.Edu/mpi/>

*MS State's MPI resource web page*

The MPI Resource Center

<http://www.epm.ornl.gov/~walker/mpi/index.html>

*Oak Ridge National Laboratory's MPI resource web page*

Message Passing Interface

<http://www.mcs.anl.gov:80/mpi/>

*Argonne National Laboratory's MPI resource web page*

Papers Related to MPI

<http://www.epm.ornl.gov/~walker/mpi/papers.html>

*MPI Paper's from Oak Ridge's page*

LAM - MPI Network Parallel Computing

<http://www.osc.edu/lam.html#MPI>

*Ohio State's MPI/LAM information page*

The LAM companion to ``Using MPI..."

<ftp://cisr.anu.edu.au/pub/papers/meglicki/mpi/tutorial/mpi/mpi.html>

*Center for Information Science Research,  
Australia National Lab's MPI/LAM tutorial*

Getting Started with MPI on LAM

<http://www.osc.edu/Lam/lam/tutorial.html>

*Ohio State's MPI/LAM installation & boot-up tutorial*

MPICH Home Page

<http://www.mcs.anl.gov/mpi/mpich/index.htm>

*Argonne National Lab's MPICH information page*

Web pages for MPI and MPE

<http://www.mcs.anl.gov/Projects/mpi/www/index.html>

*Argonne National Lab's MPI (MPE) man pages*

Computational Structures Branch

<http://olaf.larc.nasa.gov/>

*Benchmarks for matrix solvers on various platforms*