**MPI Packages**

There are MPI packages which we have explored here.

* FastMPJ
* MPJexpress
* Open MPI

**FastMPJ**

FastMPJ, the fastest Java messaging library, enables Java for high performance solutions, especially suitable for the financial sector, particularly in High Frequency Trading (HFT), as well as for High Performance Computing (HPC) and performance critical environments in particular in IT, Energy and Defense/Space.

FastMPJ, outcome of a research project of the Computer Architecture Group of the University of Corunna, has as main objective the provision of a high performance, scalable, mature and stable solution with a friendly interface.

**Running MPJ Programs**

In this section the principal commands for running your programs in an MPJ environment are presented.

This guide starts explaining the **fmpjrun** command as the easiest way to run MPJ applications because it hides the complexity of the daemons management. Then, the rest of the daemons are explained.

**fmpjrun**

**fmpjrun** starts a FastMPJ parallel job just as the daemons involved.

**fmpjd**

**fmpjd** starts a FastMPJ daemon in local machine.

**fmpjdexit**

**fmpjdexit** stops a FastMPJ daemon in local machine

**fmpjdscan**

**fmpjdscan** shows FastMPJ daemon status (running/not running).

**fmpjdboot**

**fmpjdboot** starts a ring of FastMPJ daemons all at once.

**fmpjdallexit**

**fmpjdallexit** takes down all FastMPJ daemons in ring all at once.

**fmpjdtrace**

**fmpjdtrace** shows all FastMPJ daemons status (running/not running).

**fmpjexec**

**fmpjexec** starts a FastMPJ parallel job.

**fmpjdkilljobs**

**fmpjdkilljobs** kills all FastMPJ parallel jobs.

**Running Applications**

As in the Eclipse case, at the code creation time you might add the mpj.jar file to the project libraries. You could do this in the window that pops up at the **File/Project Properties** menu, going to the **Compile** tab at the **Libraries** tag. The image below illustrates this step.

Now, when you proceed to execute your application inside the IDE’s environment, you might go to the **Libraries** configuration tag again, but in this case to the **Run** tab. There, you must add the **mpj.jar** and **runtime.jar** JAR files (placed in the FastMPJ lib directory).

Then, in the **Run** tag, on the left, you should create a new configuration where the main class might be **runtime.FMPJRun**, being in the arguments box where you pass the **main class of your project** and the desired FastMPJ arguments. Moreover, be sure that the working directory contains the compiled classes.

**Links of Software and Guide**

<http://gac.udc.es/~rreye/fastmpj/download.html?fbclid=IwAR3ztvg91RSBZjsT-x1_7aWKi0LUeJ849GB0HuoQuaEcKnQT4QV8Fl0XKGY>

<http://gac.udc.es/~rreye/fastmpj/doc/UsersGuide.pdf?fbclid=IwAR19QqPAD4Vb7iZk8bWPlVfDGUSyVb3kWvMsDfokdxSz0SEMYN-G-FsWcos>

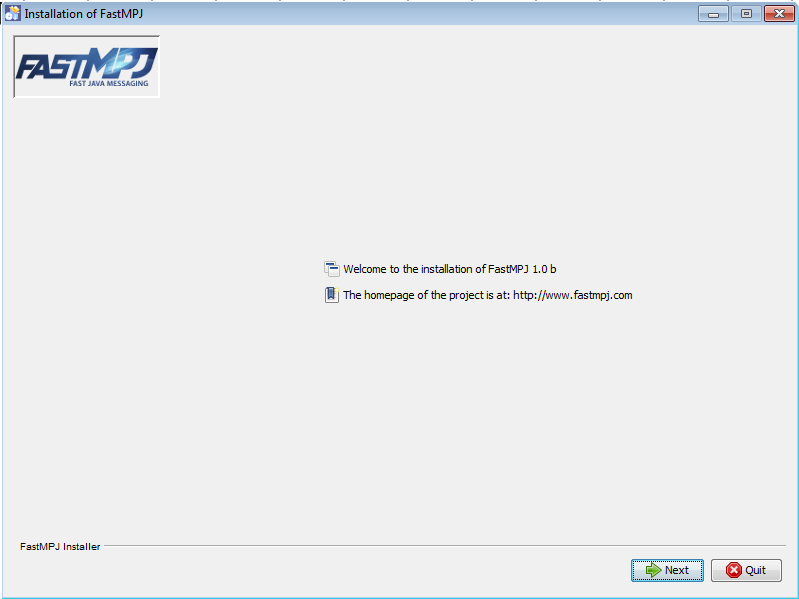
**Installation Steps**

Once you have run the installer, you enter the following installation steps:

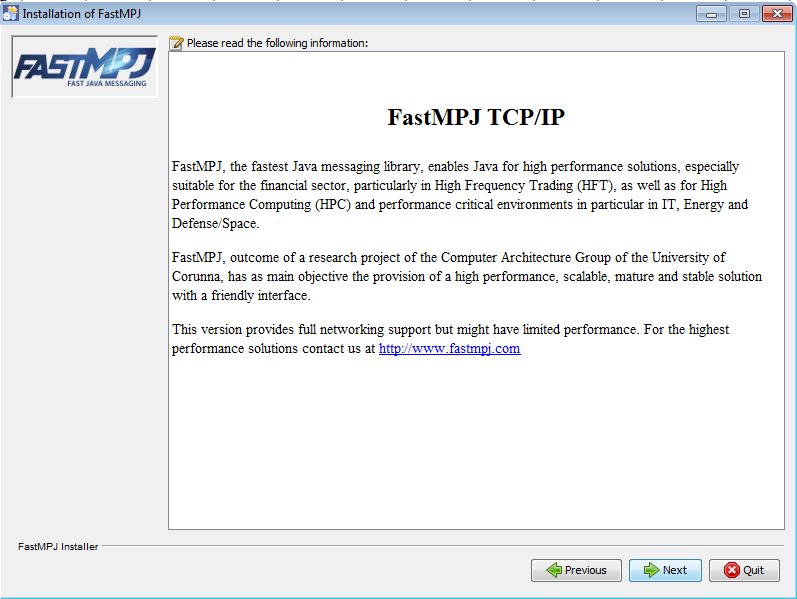
1. First, you can select the installation language in this window:



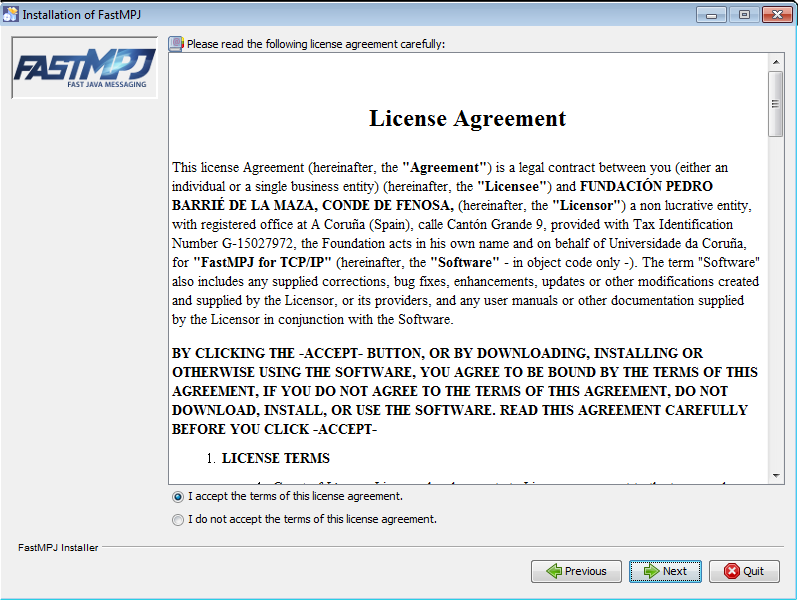
1. Next, the installer shows you the welcome window with basic information, click on next to continue.



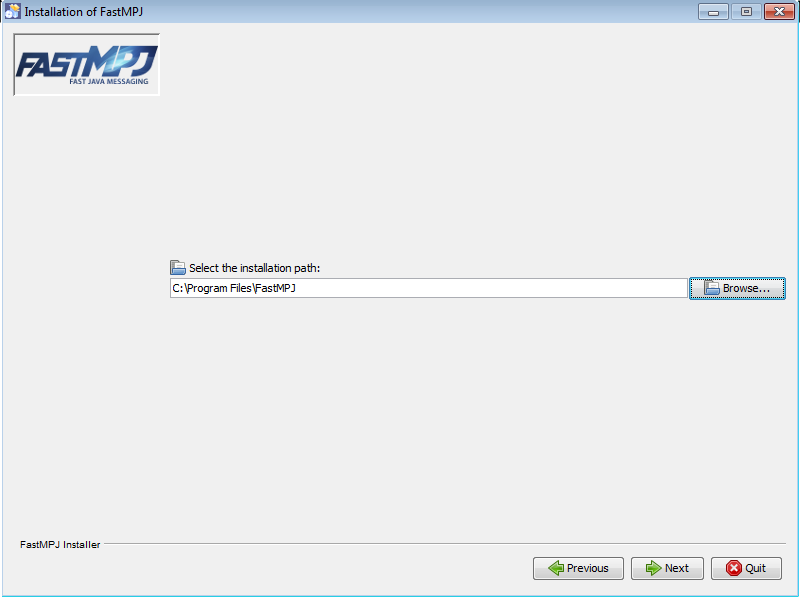
1. Now, you can read detailed information about the project, click on next again:



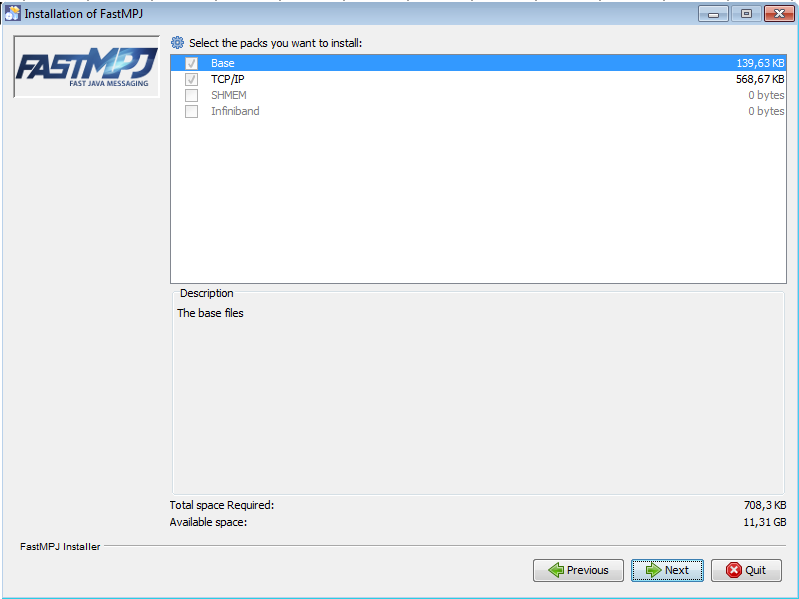
1. In this step, you must agree the license terms before continue, please read them carefully:



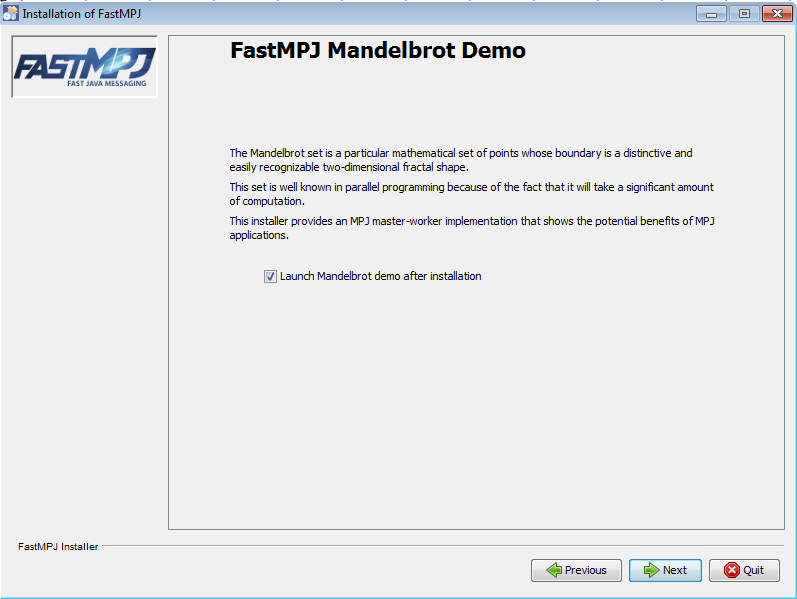
1. After that, you can choose the installation directory. Please notice that you are not allowed to choose an arbitrary location unless you are running the installer with administrator privileges.



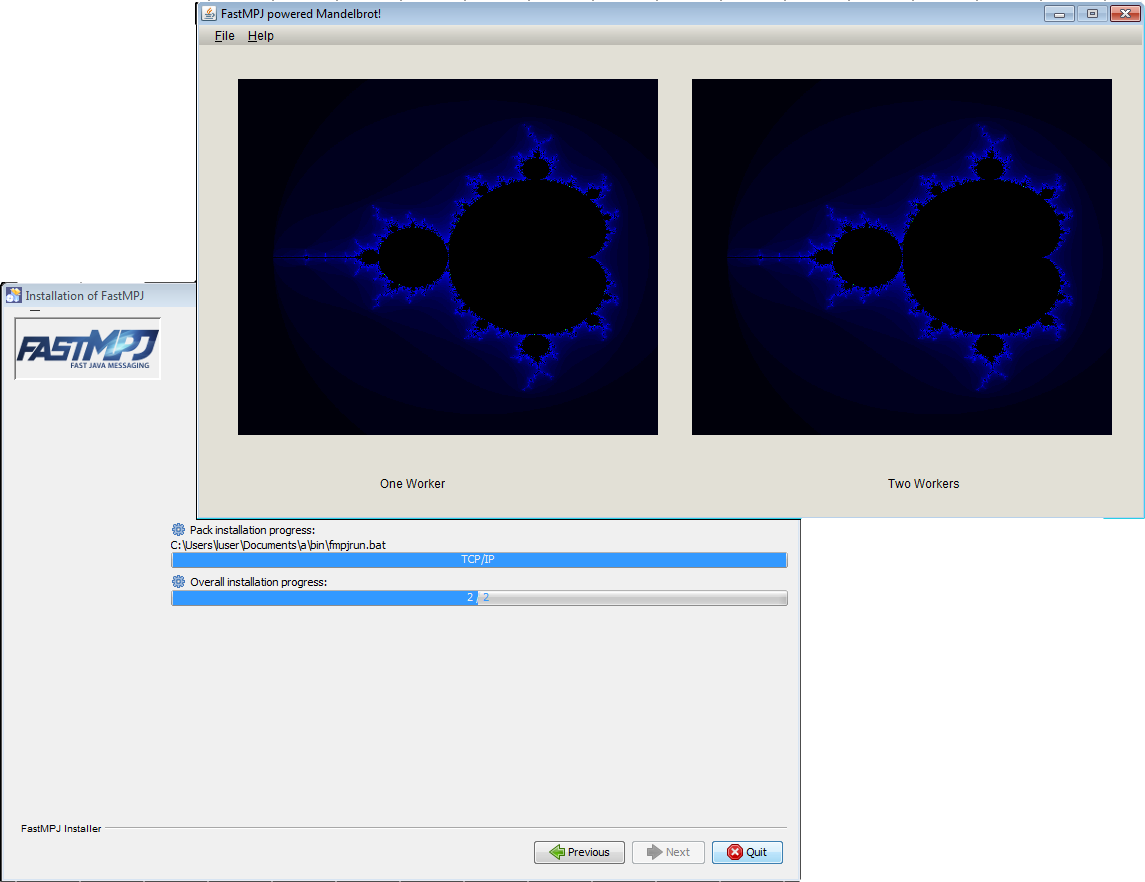
1. Depending on the installer version, in this moment you could select the packages to install:



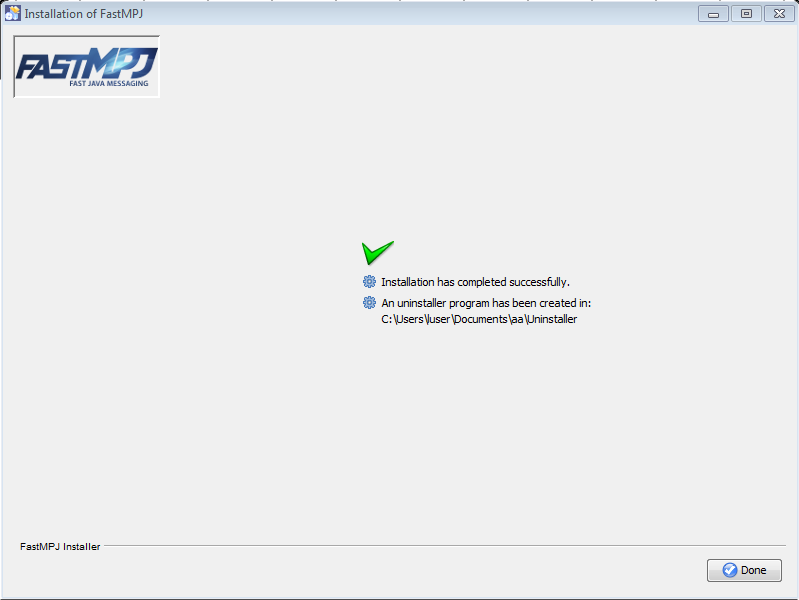
1. Again, according to the installer type, you could be asked to launch a demo at the end of the installation process. The FastMPJ TCP/IP version provides an MPJ master-worker implementation for the Mandelbrot set that shows the potential benefits of MPJ applications.



1. When you click next in the previous window, the installation process starts and, if you have choosed to launch the demo, you can see the installation progress and the demo, in this case the Mandelbrot demo.



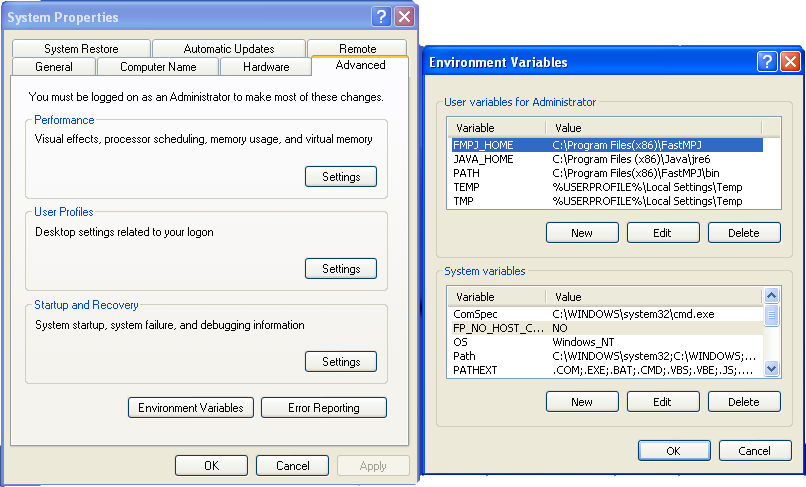
1. Closing the demo, the final windows appears and you can exit the installation process clicking on *Done*.



1. Finally, even though the installer must do it by itself, you should check if the **FMPJ\_HOME** and **JAVA\_HOME** variables were set, and if the *bin* directory (under *FMPJ\_HOME*) was added to the **PATH**.

Depending on the Windows operating system, you could proceed in a different way:

* 1. **Windows Vista**   
     Right click My Computer icon   
     Choose Properties from the context menu   
     Click Advanced tab (Advanced system settings link in Vista)
  2. **Windows 7**
     1. ***Administrator users***:   
        Select Computer from the Start menu   
        Choose System Properties from the context menu   
        Click Advanced system settings > Advanced tab
     2. ***Non administrator users***:   
        Select Control Panel from the Start menu   
        Choose User Accounts   
        On the left side click on Change environmental variables
  3. **Windows XP**   
     Right-click My Computer, and then click Properties.   
     Click the Advanced tab.   
     Click Environment variables.



For manual setup, please follow these instructions in a command window:

set (setx if available) FMPJ\_HOME=\path\to\fmpj\

set (setx if available) PATH=%PATH%;%FMPJ\_HOME%\bin

set (setx if available) JAVA\_HOME=\path\to\java

**Sample Code of MPI**

package examples;

import mpi.\*;

public class HelloWorld

{

public static void main(String args[]) throws Exception

{

int me,size;

args = MPI.Init(args);

me = MPI.COMM\_WORLD.Rank();

size = MPI.COMM\_WORLD.Size();

System.out.println(MPI.Get\_processor\_name()+": Hello World from "+me+" of "+size);

MPI.Finalize();

}

}

**Point to Point Communication**

import mpi.\* ;

class Hello {

static public void main(String[] args) {

MPI.init(args) ;

int myrank = MPI.COMM\_WORLD.rank() ;

if(myrank == 0) {

char [] message = "Hello, there".toCharArray() ;

MPI.COMM\_WORLD.send(message, 0, message.length, MPI.CHAR, 1, 99) ;

}

else {

char [] message = new char [20] ;

MPI.COMM\_WORLD.recv(message, 0, 20, MPI.CHAR, 0, 99) ;

System.out.println("received:" + new String(message) + ":") ;

}

MPI.finish();

}

}

**Collective** **Communication**

void my\_bcast(void\* data, int count, MPI\_Datatype datatype, int root,

MPI\_Comm communicator) {

int world\_rank;

MPI\_Comm\_rank(communicator, &world\_rank);

int world\_size;

MPI\_Comm\_size(communicator, &world\_size);

if (world\_rank == root) {

// If we are the root process, send our data to everyone

int i;

for (i = 0; i < world\_size; i++) {

if (i != world\_rank) {

MPI\_Send(data, count, datatype, i, 0, communicator);

}

}

} else {

// If we are a receiver process, receive the data from the root

MPI\_Recv(data, count, datatype, root, 0, communicator,

MPI\_STATUS\_IGNORE);

}

}

**MPJ Express**

MPJ Express is a reference implementation of the mpiJava 1.2 API, which is an MPI-like API for Java defined by the Java Grande forum. The mpiJava 1.2 API is the Java equivalent of the MPI

This release of the MPJ Express software contains the core library and the runtime infrastructure. The software also contains a comprehensive test suite that is meant to test the functionality of various communication functions.

MPJ Express is a message passing library that can be used by application developers to execute their parallel Java applications on compute clusters or network of computers. Compute clusters is a popular parallel platform, which is extensively used by the High Performance Computing (HPC) community for large scale computational work. MPJ Express is essentially a middleware that supports communication between individual processors of clusters.

**Installation Steps**

This section outlines steps to download and install MPJ Express software.

1. Download MPJ Express and unpack it

2. Assuming unpacked 'mpj express' is in 'c:\mpj', Right-click My

ComputerPropertiesAdvanced tabEnvironment Variables and export the following

system variables (user variables are not enough)

a. Set the value of variable MPJ\_HOME as c:\mpj [see Fig 4,Fig 5 and Fig 6]

b. Append the value of variable Path as c:\mpj\bin [see Fig 7]

See the snapshots below

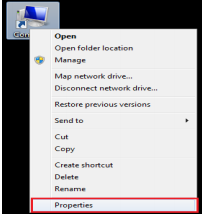
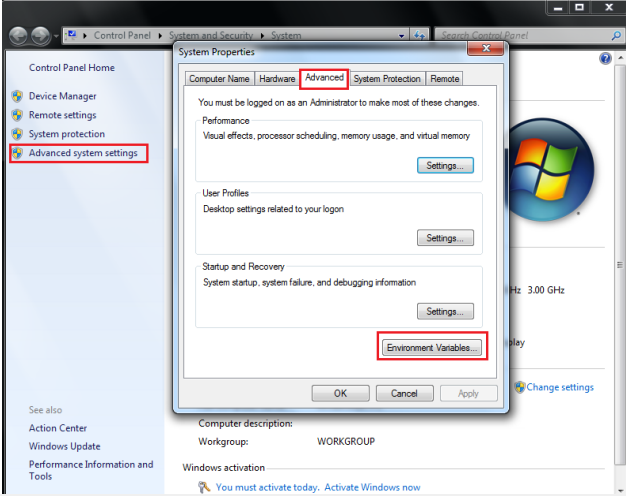
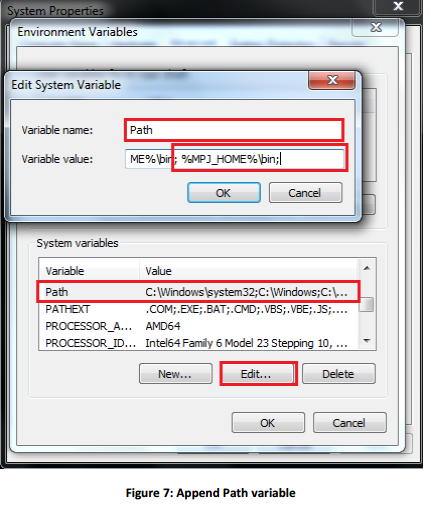
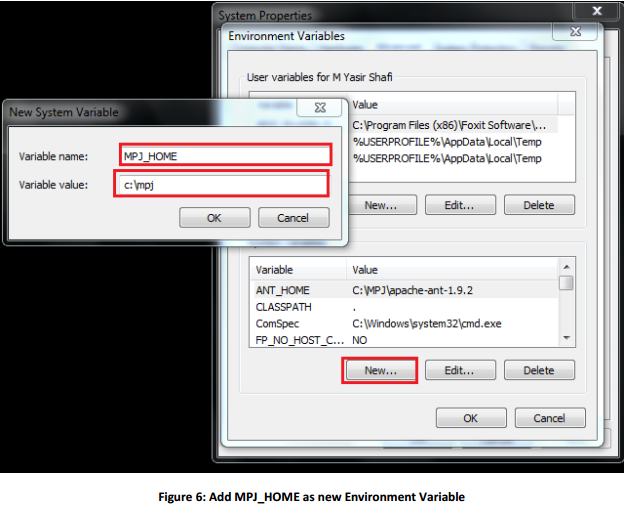


Figure 4: Right click on my computer and select Properties





For windows with Cygwin (assuming „mpj express‟ is in „c:\mpj‟) The recommended way to is to set variables as in Windows If you want to set variables in cygwin shell  
export MPJ\_HOME="c:\\mpj"  
export PATH=$PATH:"$MPJ\_HOME\\bin" 4. Create a new working directory for MPJ Express programs. This document assumes that the name of this directory is “mpj-user”. 5. Compile the MPJ Express library (Optional): cd %MPJ\_HOME%; ant

s

**Compilation**

Compile: javac -cp .;%MPJ\_HOME%/lib/mpj.jar HelloWorld.java

This section outlines steps to execute parallel Java programs in the multicore configuration. 1. Assuming the user has successfully carried out Section 2.2 and Section 2.3 2. Running HelloWorld Execute: mpjrun.bat -np 2 HelloWorld 3. Running test cases (Optional) [Test suite is provided with MPJ Express] a. Compile (Optional): cd %MPJ\_HOME%/test; ant b. Execute: mpjrun.bat -np 2 -jar %MPJ\_HOME%/lib/test.jar

**Links**

<http://mpj-express.org/docs/guides/windowsguide.pdf>

<http://mpj-express.org/download.php>

**Sample Code of MPI**

import mpi.\*;  
public class HelloWorld {  
public static void main(String args[]) throws Exception {  
MPI.Init(args);  
int me = MPI.COMM\_WORLD.Rank();  
int size = MPI.COMM\_WORLD.Size();  
System.out.println("Hi from <"+me+">");  
MPI.Finalize();  
}  
}

**Point to Point Communication**

} public class PingPongJava {

public static void main(String args[]) throws Exception {

int me;

int size;

int[] recv\_buf = { 0 };

int[] buf = {0};

int[] send\_buf = { 101 };

double tic = 0, toc = 0;

MPI.Init(args);

me = MPI.COMM\_WORLD.Rank();

size = MPI.COMM\_WORLD.Size();

tic = MPI.Wtime();

for (int i = 0; i < 50; i++) {

if (me == 0) {

MPI.COMM\_WORLD.Send(send\_buf, 0, 1, MPI.INT, 1, 17);

MPI.COMM\_WORLD.Recv(recv\_buf, 0, 1, MPI.INT, 1, 23);

} else {

MPI.COMM\_WORLD.Recv(buf, 0, 1, MPI.INT, 0, 17);

MPI.COMM\_WORLD.Send(buf, 0, 1, MPI.INT, 0, 23);

}

}

toc = MPI.Wtime();

if (me == 0) {

System.out.println("Time taken is " + (toc - tic) / 100);

}

MPI.Finalize();

}

}

**Collective** **Communication**

void my\_bcast(void\* data, int count, MPI\_Datatype datatype, int root,

MPI\_Comm communicator) {

int world\_rank;

MPI\_Comm\_rank(communicator, &world\_rank);

int world\_size;

MPI\_Comm\_size(communicator, &world\_size);

if (world\_rank == root) {

// If we are the root process, send our data to everyone

int i;

for (i = 0; i < world\_size; i++) {

if (i != world\_rank) {

MPI\_Send(data, count, datatype, i, 0, communicator);

}

}

} else {

// If we are a receiver process, receive the data from the root

MPI\_Recv(data, count, datatype, root, 0, communicator,

MPI\_STATUS\_IGNORE);

}

}

**Open MPI**

Open MPI is a new implementation of MPI, attempting to create the best MPI library available.

It uses resources and technology from existing or defunct MPI libraries such as FT-MPI, LAMPI,

LAM/MPI, and PACX-MPI, developing an all new implementation influenced by these codes.

Open MPI aims to provide support for the majority of established high performance interconnects, TCP/IP, shared memory. In addition the modular component system enables easy extension and addition, one such example is that of interconnect support.

The performance of Open MPI against established MPI implementations is not expected to be comparable. Open MPI has been tested against all our target architectures, and so should pose no real problem to compile and run the benchmarks.

MPI is used to describe the interaction (communication) within applications for distributed memory machines. It provides source level portability of parallel applications between different hardware platforms. Many concrete implementations of the MPI standard including Open MPI, MPICH, MVAPICH, Intel MPI, vendor libraries.

**Background**

1. Version 1.0 (1994): FORTRAN 77 and C bindings
2. Version 1.1 (1995): Minor corrections and clarifications
3. Version 1.2 (1997): Further corrections and clarifications
4. Version 2.0 (1997): Major enhancements
5. Version 2.1 (2008): Merger of MPI-1 and MPI-2
6. Version 2.2 (2009): Minor corrections and clarifications
7. Version 3.0 (2012): Major enhancements

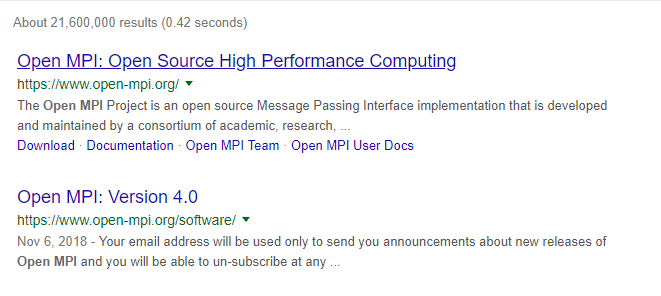
**Installation Steps**

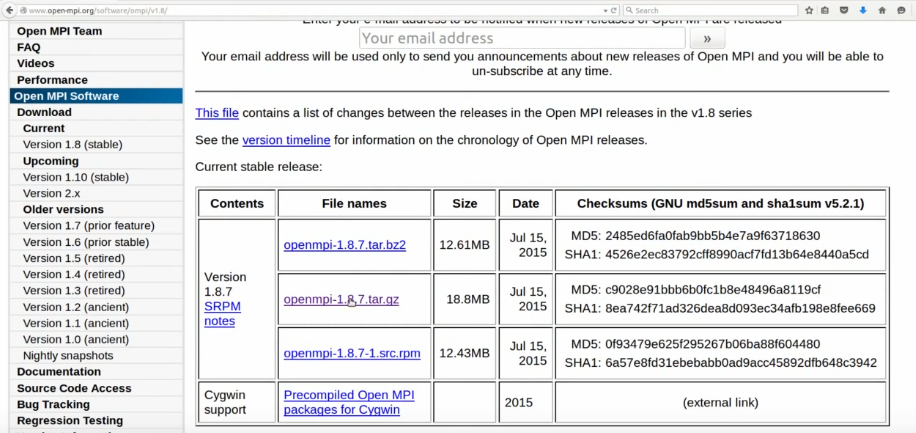
Open MPI is an open source MPI-2 implementation that is developed and maintained by a consortium of academic, research, and industry partners. Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available.

As **pre requisite**: It is necessary to have installed a C/C++ compiler before installing OpenMPI. Installing this on Ubuntu by package manager will automatically resolve this problem.\

**Step1:** Download the last version of OpenMPI from its official page:

<http://www.open-mpi.org/software/ompi>





**Step2:** If system supports .rpm file this is the package you should download. In case, you'll only have to install it without any of the following steps (usually double-clicking)

OR

If system does not support .rmp extension file, follow the next steps:

**Step3:** Decompress the downloaded file with file extension .tar

>> tar -xvf openmpi-\*

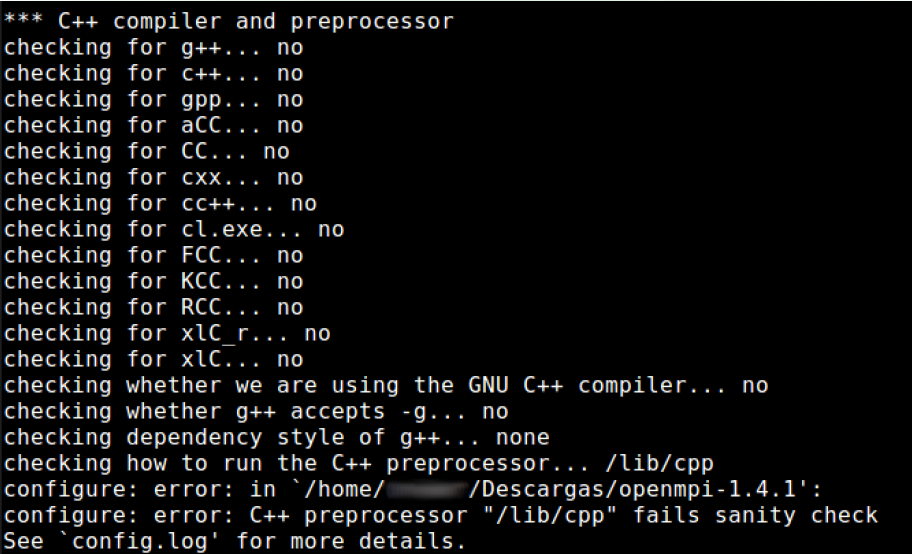
**Step4:** Go into the new folder created from the decompress and configure the installation file

>> cd openmpi-\*

**Step5:** The necessary step is to add prefix for the installation directory that we want to use for OpenMPI. Best practice is to select the next directory.

>> /configure --prefix="/home/USER/.openmpi"

In case if below error occurred, so it means that you haven’t installed any compiler



**Step6:** include the path to our path environment the path and to the library environment variable.

>> export PATH="$PATH:/home/$USER/.openmpi/bin"

>> export LD\_LIBRARY\_PATH="$LD\_LIBRARY\_PATH:/home/$USER/.openmpi/lib/"

**Sample Code**

#include <stdio.h>

#include <mpi.h>

int main(int argc, char\*\* argv)

{

int myrank, nprocs;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &nprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank);

printf("Hello from processor %d of %d\n", myrank, nprocs);

MPI\_Finalize();

return 0;

}

**Compilation**

$ mpicc MPI\_Hello.c -o MPI\_Hello 

**Execution**

$ mpiexec -n 5 MPI\_Hello

**Six Main Functions of OpenMPI**

MPI is very simple. These six functions allow you to write many programs:

* MPI Init
* MPI Finalize
* MPI Comm size
* MPI Comm rank
* MPI Send
* MPI Recv

**Point to Point Communication**

#include <mpi.h>

int main(int argc, char \*\*argv)

{

int numProcs, rank, data;

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD,

&numProcs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,

&rank);

if (rank == 0) MPI\_Recv(&data, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, &status);

else if (rank == 1)

MPI\_Send(&data, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Finalize();

return 0;

}

**Comparison**

