MPJ Express: Eclipse Plugin for MPJ Express Debugger

User Guide

7th February, 2014

Document Revision Track

Version	Updates	By
1.0	Initial version document	Rizwan Hanif, Amjad Aziz, Aamir Shafi
1.1	Complete step details, Steps for running and debugging in hybdev configuration	Aleem Akhtar, Aamir Shafi, Mohsan Jameel

Table of Contents

1. Introduction....6
   1.1 Configurations....6
      1.1.1 Multicore configuration....6
      1.1.2 Cluster configuration....7
      1.1.3 Hybrid Configuration....8
2. Debugging Tool....9
   2.1 Sequential Debuggers....9
   2.2 Parallel/Distributed Debuggers....9
      2.2.1 Remote Debugger....9
3. MPJ Express Debugger –Eclipse Plugin....10
   3.1 Local Debugging....10
   3.2 Remote Debugging....10
4. Getting Started with MPJ Express....11
   4.1 Pre-requisites....11
   4.2 Installation of MPJ-Express Debugger....11
   4.3 MPJ Express New Application....11
   4.4 Running MPJ Express in the Multi-core Configuration....15
   4.5 Running MPJ Express in the Cluster Configuration....15
   4.6 Running MPJ Express in the Hybrid Configuration....17
5. MPJ Express Application Debugging....19
   5.1 Local Debugging....19
   5.2 Remote Debugging....21
6. Known Issues and Limitations....24

7. Contact and Support 25

8. Appendices.....26

Appendix A: Setting Environment Variables in Windows.....26

Appendix B: Setting Environment Variables in Linux.....28

List of Figures

Figure 1: Shared Memory Communication Architecture 7
Figure 2: Distributed Memory Communication Architecture 8
Figure 3: Cluster of Shared Memory Architecture Machines 8
4: MPJ Express Application option in Run Configurations 11
Figure 5: Create a new java project 12
Figure 6: Add External Libraries 13
Figure 7: Click finish to add project 13
Figure 8: Run Configurations 14
Figure 9: MPJ Parameters for multicore Configuration 15
Figure 10: MPJ Parameters for niodev Configuration 17
Figure 11: MPJ Parameters for hybdev Configuration 18
Figure 12: MPJ Parameters for Local Debugging 20
Figure 13: Debug Perspective for MPJ Express Debugging 20
Figure 14: MPJ Parameters for Remote Debugging 21
Figure 15: Connect Parameters for Remote Multicore Configuration 22
Figure 16: Connect Parameters for Remote Distributed Debugging 23
Figure 17: State partition error 24
Figure 18: Solution of State partition error 24
Figure 19: Right click on my computer and select Properties 26
Figure 20: Select Environment Variables to Add/Edit variables 26
Figure 21: Add MPJ_HOME as new Environment Variable 27
Figure 22: Append Path variable 27

1. Introduction

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 1.1 specification document (http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html).

MPJ Express is a message passing library that can be used by application developers to execute their parallel Java applications on compute cluster or network of computers. Compute cluster 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 cluster. The programming model followed by MPJ Express is Single Program Multiple Data (SPMD).

The MPJ Express software can be configured in three ways. The first configuration—known as the Multicore Configuration—is used to execute MPJ Express user programs on laptops and desktops. The second configuration—known as the Cluster Configuration—is used to execute MPJ Express user programs on cluster or network of computers. The third configuration—known as Hybrid Configuration—is used to execute MPJ Express user programs in cluster of multicore.

1.1 Configurations

The MPJ Express software can be configured to work on cluster (network of computers), on laptops/desktops (multicore processors) or on cluster of multicore processors.

1.1.1 Multicore configuration

The multicore configuration is meant for users who plan to write and execute parallel Java applications using MPJ Express on their desktops or laptops—typically such hardware contains shared memory and multicore processors. In this configuration, users can write their message passing parallel application using MPJ Express and it will be ported automatically on multicore processors. We envisage that users can first develop applications on their laptops and desktops using multicore configuration, and then take the same code to distributed memory platforms including cluster. Also this configuration is preferred for teaching purposes since students can execute message passing code on their personal laptops and desktops. It might be noted that user applications stay the same when executing the code in multicore or cluster configuration.

Under the hoods, the MPJ Express library starts a single thread to represent MPI process. The multicore communication device uses efficient inter-thread mechanism.

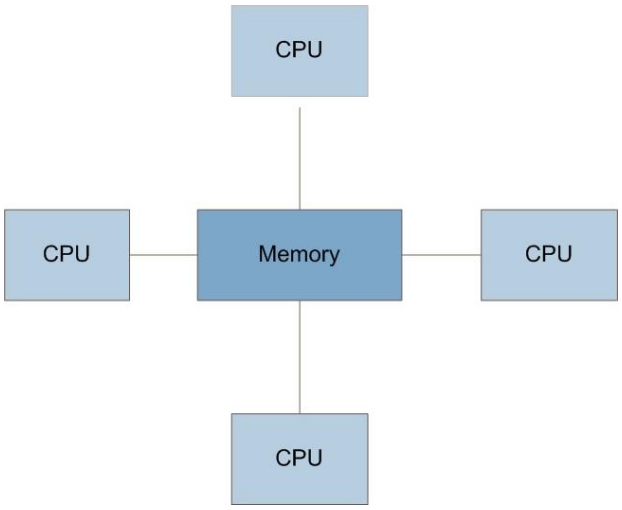


Figure 1: Shared Memory Communication Architecture

1.1.2 Cluster configuration

The cluster configuration is meant for users who plan to execute their parallel Java applications on distributed memory platforms including cluster or network of computers. Application developers can opt to use either of the two communication devices in the cluster configuration: the communication devices including Java New I/O (NIO) device or Myrinet device.

1. Java New I/O (NIO) device driver known as niodev
2. Myrinet device driver known as mxdev

The Java NIO device driver (also known as niodev) can be used to execute MPJ Express programs on cluster or network of computers. The niodev device driver uses Ethernet-based interconnect for message passing. On the other hand, many clusters today are equipped with high-performance low-latency networks like Myrinet. MPJ Express also provides a communication device for message passing using Myrinet interconnect—this device is known as mxdev and is implemented using the Myrinet eXpress (MX) library by Myricom. These communication drivers can be selected using command line switches.

As an example, consider a cluster or network of computers shown in Figure 2 that shows eight compute nodes connected to each other via private interconnect. The MPJ Express cluster configuration will start one MPJ Express process per node, which communicates to each other using message passing

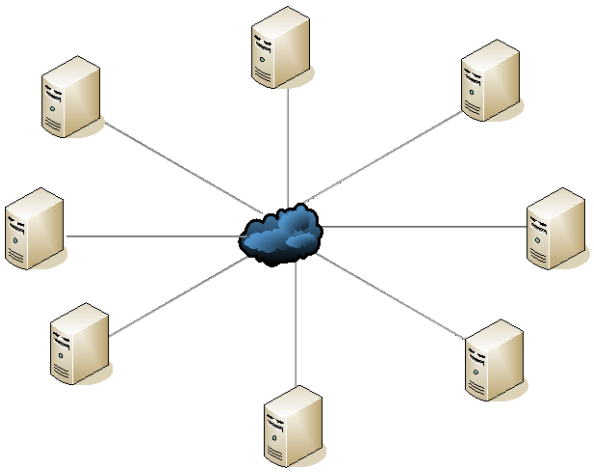


Figure 2: Distributed Memory Communication Architecture

1.1.3 Hybrid Configuration

The hybrid configuration is meant for users who plan to execute their parallel Java applications on cluster of multicore machines. Hybrid configuration enclosed both multicore configuration and cluster configuration and uses one of them for each message. Hybrid configuration uses multicore configuration for intra-node communication and cluster configuration (NIO device only) for inter-node communication.

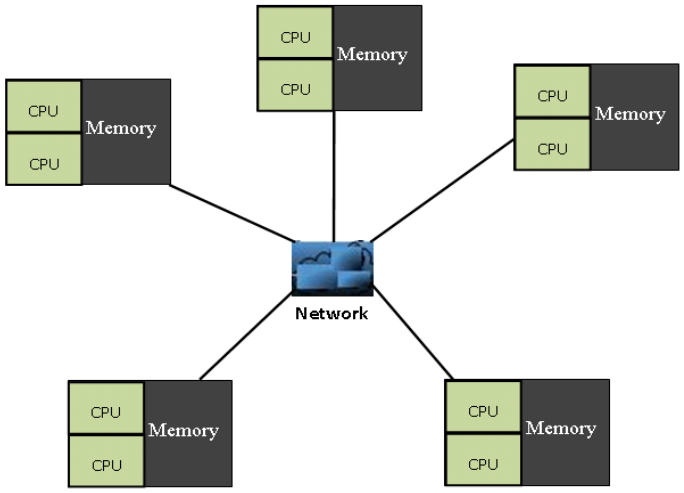


Figure 3: Cluster of Shared Memory Architecture Machines

2. Debugging Tool

Debugging is a systematic process of locating and reducing the number of bugs, or defects in computer software. This makes the program operate as desired. However, debugging sometimes tends to be harder in software product development owing to tightly coupled subsystems. When there are changes made to a particular subsystem, it causes bugs to emerge in another.

2.1 Sequential Debuggers

Debugging of single process single thread application is known as Sequential Debugging. To know which function/block the program enters or what are the values of variables at particular line in program. One can insert print statements in program that report which portion of program is executing and what are values of different variables at current stage. **gdb** and **dbx** are common examples of Sequential Debuggers.

2.2 Parallel/Distributed Debuggers

Distributed software development must address many complex issues generally not encountered in sequential program, such as synchronization, concurrency control, communication delays and non-determinism, to name a few. Similarly, debugging distributed programs is much more difficult than debugging sequential program. A distributed program, in contrast, consists of multiple processes with multiple instruction threads running on remote processors each with their own separate memories. The possibility of remote execution combined with communication delays makes it difficult to debug a distributed program and to implement a distributed program debugger. Furthermore, distributed programs, especially faulty ones, may not be deterministic due to concurrent execution.

2.2.1 Remote Debugger

Since we are running MPJ Express processes in different JVMs, so we need remote debugger which connects JVMs. gdb (The Java Debugger) provide option to debug application on remote machine i.e. if application is running on remote machine then user can debug that application from local machine. The Java Debug Wire Protocol (JDWP) is the protocol used for communication between MPJ Express debugger and the Java virtual machine (VM) which it debugs. The existence of JDWP allows MPJ Express Debugger to work on remote computers.

3. MPJ Express Debugger –Eclipse Plugin

The MPJ Express Debugger is based on Eclipse debugger. Client/server design of Eclipse debugger allows user to debug programs running locally on your workstation as well as programs running remotely on other systems in the network. In case of local debugging, program will be launched on your workstation or in case of remote debugging; program will be launched on a system that is accessible through a network where you want to launch your job. Current version of MPJ Express Debugger supports both local and remote Debugging modes.

3.1 Local Debugging

In local debugging, launched program and debugger client runs on the same workstation which makes local debugging simplest and most common kind of debugging. You can then use breakpoints, step into, step out, or step return features to debug your program. In local debugger both MPJ Express process and MPJ Express Debugger should be on same machine.

3.2 Remote Debugging

In remote debugging, you launch MPJ Express program on your workstation and MPJ Express debugger client runs on some other system on the same network. Remote debugging comes handy when you are developing program for a device that cannot host the development platform. It is also useful when debugging MPJ Express programs running on cluster. For Multicore mode of MPJ Express Remote Debugging user specify Host name and Port of running application while for Cluster mode user specify `mpjdev.conf` file path.

Note: To use remote debugging, you must be using a Java VM that supports this feature i.e. JRE 1.6 or higher.

4. Getting Started with MPJ Express

This section shows how MPJ Express programs can be executed in the multicore, cluster and hybrid configuration using MPJ Express Debugger.

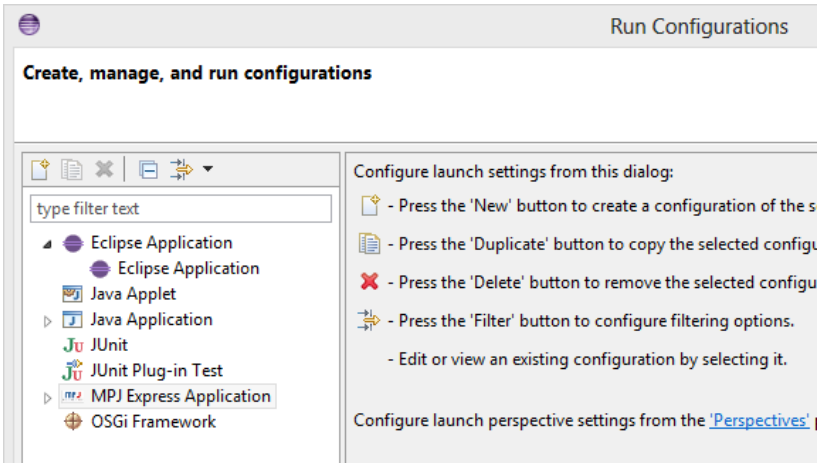
Complete video guide can be found at http://vimeo.com/86049819

4.1 Pre-requisites

- Java 1.6 (stable) or higher
- MPJ-Express (version 0.40 or higher)
- Eclipse IDE (version 3.6 or higher)
- Apache ant 1.6.2 or higher (For those who are interested in Compiling source code)
- Perl (Optional): MPJ Express needs Perl for compiling source code because some of the Java code is generated from Perl templates. The build file will generate Java files from Perl templates if it detects Perl on the machine. It is a good idea to install Perl if you want to do some development with MPJ Express.

4.2 Installation of MPJ-Express Debugger

1. Download MPJ-Express Debugger from MPJ Express official website and unpack it
2. Copy MPJ_Express_Debugger.jar file and paste in Eclipse plugins directory
3. Run eclipse as an Administrator
4. Go to Run→Run Configurations menu and check if there is MPJ-Express Application option added or not



4: MPJ Express Application option in Run Configurations

4.3 MPJ Express New Application

MPJ Express can be run in Eclipse using three configurations namely multicore, cluster and hybrid configuration. Steps for installing MPJ Express in all configurations are same but differ where we provide device field for all the configurations.

1. Download MPJ-Express and unpack it.
2. Set Environment Variables (MPJ_HOME, PATH). You can follow **Annex A** for Windows and **Annex B** for Linux.
3. Create a new Java Project. Name it as "MPJEclipseTest" and Click on Next (Do not click on finish at this point).

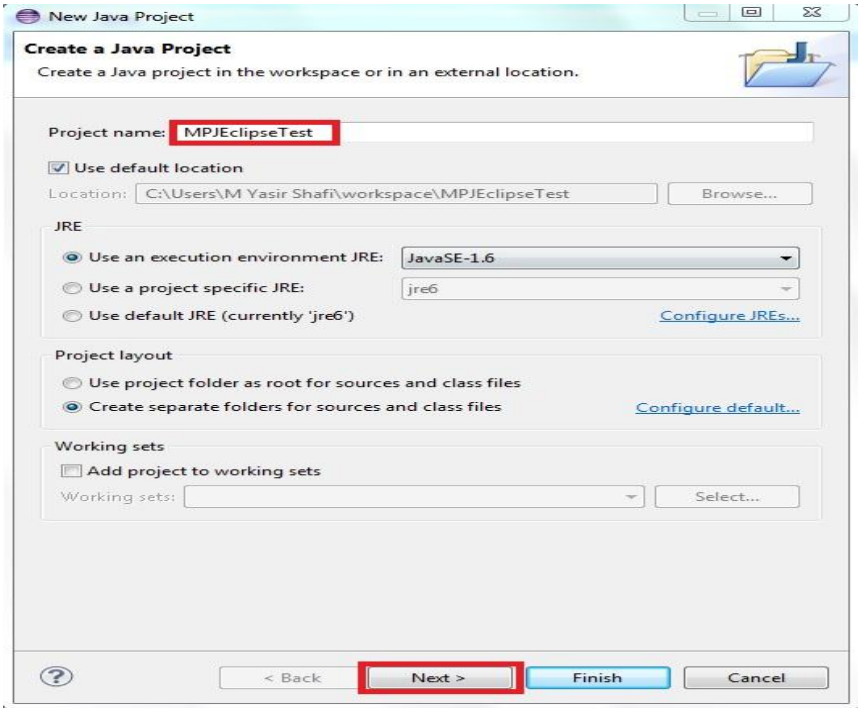


Figure 5: Create a new java project

4. On Java setting page for this project, click on Libraries tab and select "Add External JARs". Look for mpj.jar in lib folder of "MPJ Express" root directory. Click finish

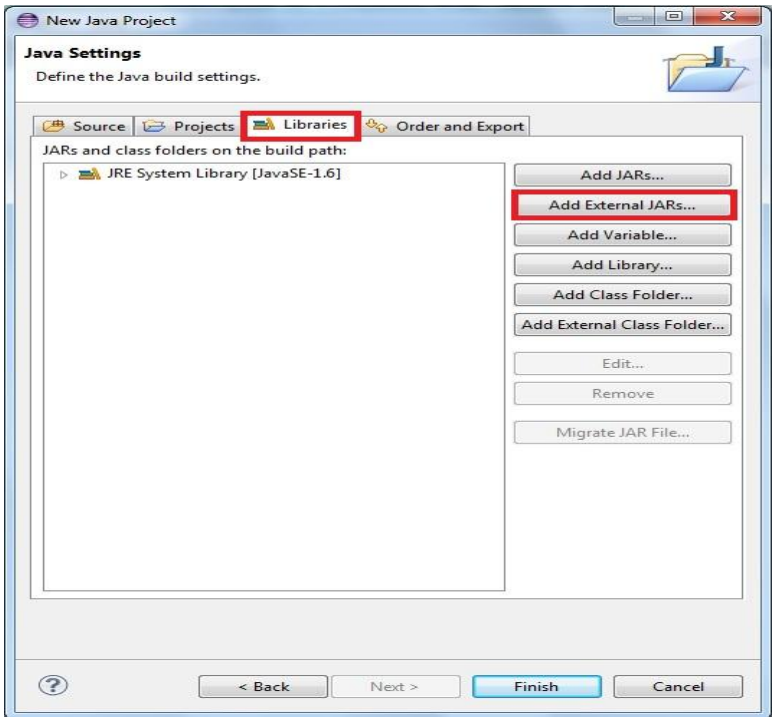


Figure 6: Add External Libraries

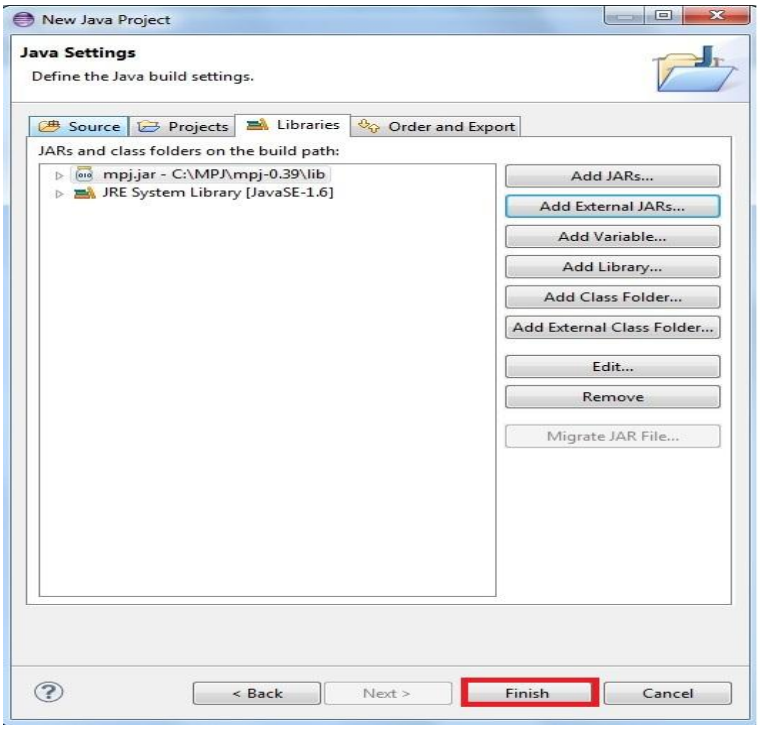


Figure 7: Click finish to add project

5. Add a new Java class and write Hello World MPJ Express program in it

```
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();
	}
}
```

6. To compile this program in Eclipse, Right click on Java Class and go to Run as → Run Configurations

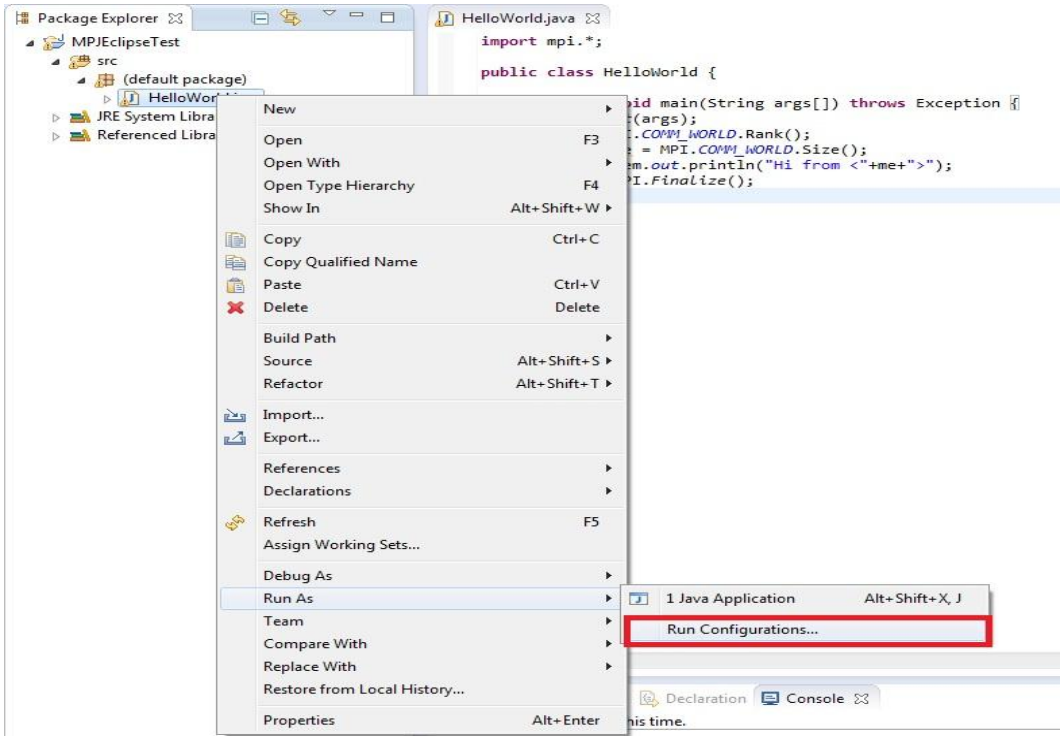


Figure 8: Run Configurations

7. Create, manage and run configurations will be opened. Double click on **MPJ Express Application**. A new run configuration named "HelloWorld" will be added with different tabs.

8. Select MPJ Parameters tab. Here you can enter number of processes you want to want run (np), device you want to use (device [multicore, niodev, hybdev]) and additional arguments if required.

9. Browse for MPJ-Express root directory and set it for MPJ_HOME field

10. Enter number of processes (np). See figure-9
11. Click Apply

4.4 Running MPJ Express in the Multi-core Configuration

This section outlines steps to execute parallel Java programs in the multicore configuration.

1. Follow steps from 1 to 11 of above section 4.3
2. Enter multicore in device field of MPJ Parameters tab in Run Configurations
3. Click apply and Run
4. MPJ Express Application will start running in multicore mode

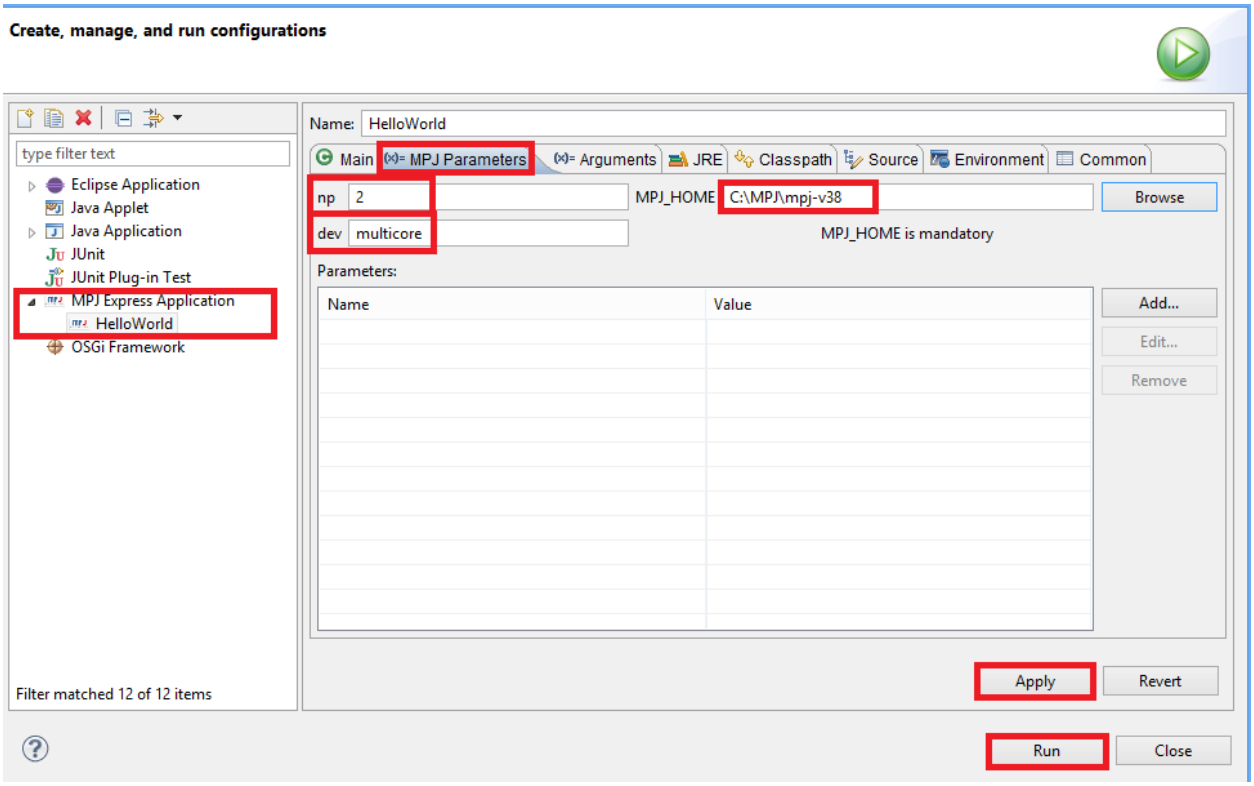


Figure 9: MPJ Parameters for multicore Configuration

4.5 Running MPJ Express in the Cluster Configuration

This section outlines steps to execute parallel Java programs in the cluster configuration with niodev communication device driver.

1. Write a machines file stating machine name, IP addresses, or aliases of the nodes where you wish to execute MPJ Express processes. Save this file as 'machines' in mpj-user directory. This file is used by scripts like mpjboot, mpjhalt, mpjrun.bat and mpjrun.sh to find out which machines to contact. For detail configuration steps you can follow section-2.3 of Linux/Windows guide of MPJ Express.

   Suppose you want to run a process each on 'machine1' and 'machine2', then your machines file would be as follows

   ```
   machine1
   machine2
   ```

   Note that in real world, 'machine1' and 'machine2' would be fully qualified names, IP addresses or aliases of your machine

2. Start the daemons

   - Windows users

     a. Run %MPJ_HOME%/bin/installmpjd-windows.bat. Users of Windows Vista and 7 needs to "Run as Administrator". Also, for running properly, users of Windows Vista and 7 might need to turn off their firewall.

     b. Go to Control-Panel→Administrative Tools→Services→MPJ Daemon and start the service.

   - Linux users

     a. Open Terminal and move to current project directory where machines file is placed. Run mpjboot machines command. Daemons will start running

3. Follow steps from 1 to 11 of section 4.3
4. Enter niodev in device field of MPJ Parameters tab in Run Configuration
5. Click apply and Run

   MPJ Express Application will start running in niodev mode

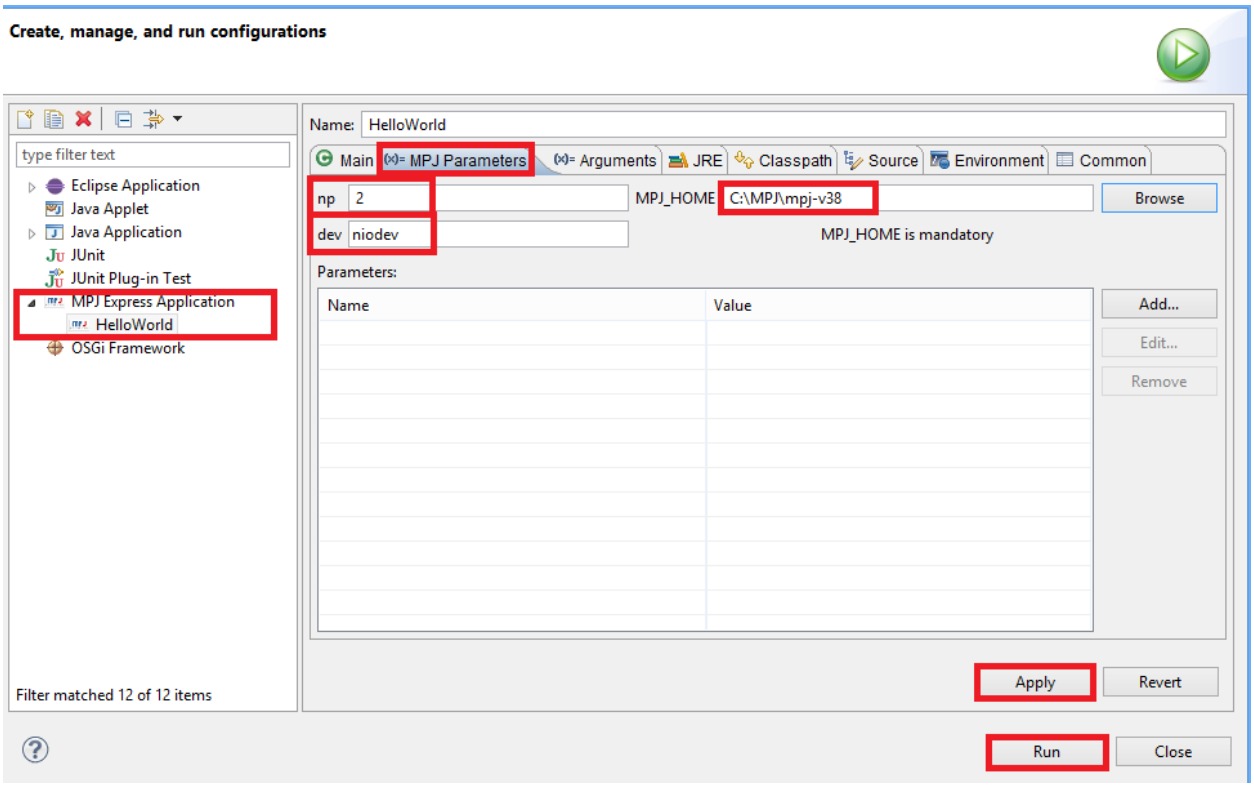


Figure 10: MPJ Parameters for niodev Configuration

4.6 Running MPJ Express in the Hybrid Configuration

This section outlines steps to execute parallel Java programs in the hybrid configuration with hybdev communication device driver.

1. Follow steps from 1 to 3 of section 4.5
2. Enter hybdev in device field of MPJ Parameters tab in Run Configuration
3. Click apply and Run

   MPJ Express Application will start running in hybdev mode

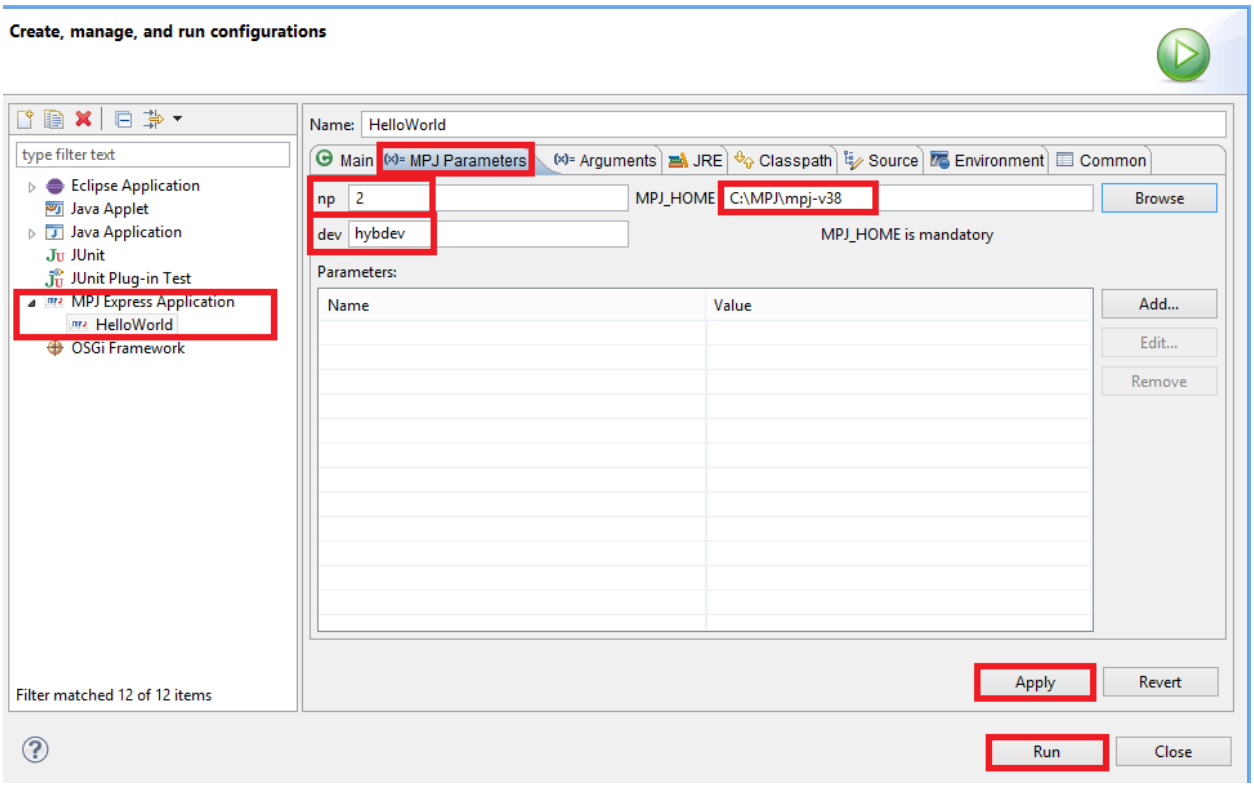


Figure 11: MPJ Parameters for hybdev Configuration

5. MPJ Express Application Debugging

Users can debug MPJ Express applications using MPJ-Express Debugger for all three configurations modes [multicore, niodev, hybdev]. Steps for all three modes are same. There are two types of debugging that current MPJ-Express Debugger supports

- Local Debugging
- Remote Debugging

Detail steps can be followed below

5.1 Local Debugging

1. Create a new MPJ-Express Application in eclipse. You can follow steps from 1 to 11 of section 4.3
2. Introduce breakpoints where you want to stop your MPJ Express Application.
3. Click on Run→ Debug Configurations. Create, manage, and run configuration window will open. Here you can see two options for MPJ-Express [MPJ Express Application and Remote MPJ Express Application]. Since we are debugging in local mode so double click on MPJ Express Application option. A new configuration with "HelloWorld" will be added.
    a. Browse for MPJ Express root directory and set field MPJ_HOME
    b. Enter number of processes in np field
    c. Enter Device name [multicore, niodev, or hybdev] in device field.
    d. Add a new parameter with **Name = debug** and **value = 8000**. Note 8000 is port number, you can use whatever port you want provided the port you chose if available.
4. Click Apply and Debug

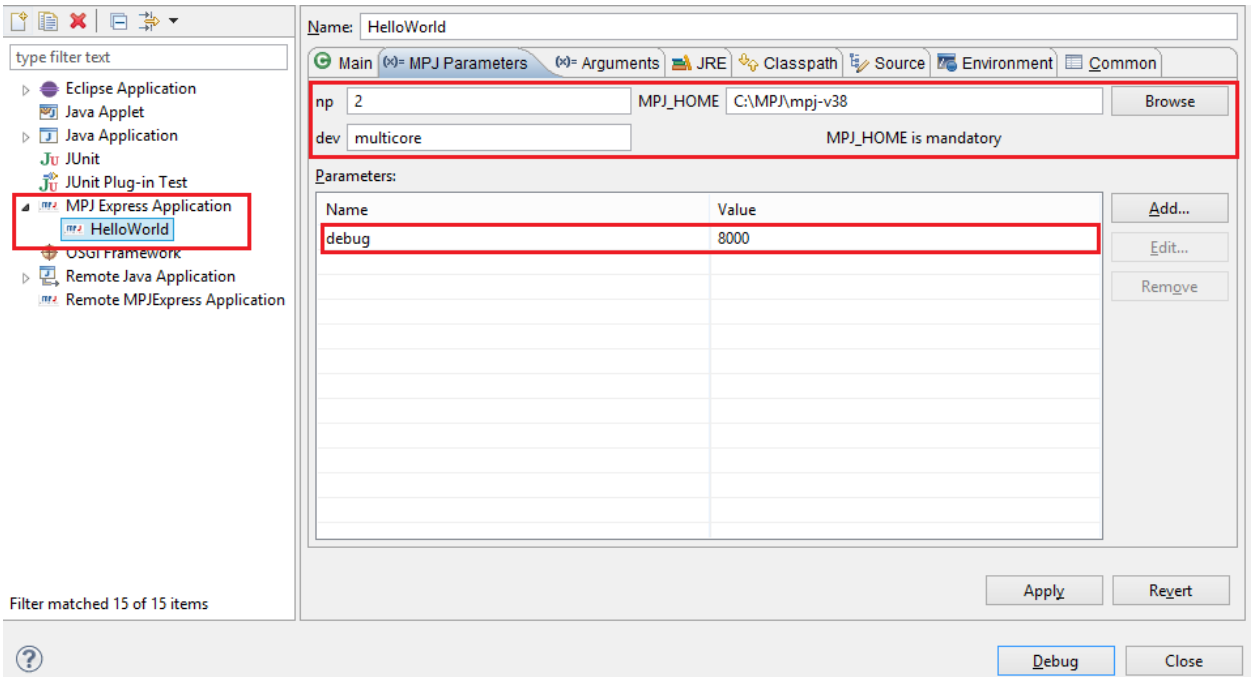


Figure 12: MPJ Parameters for Local Debugging

5. Application will open in Debug Perspective with all processes suspended on breakpoint.

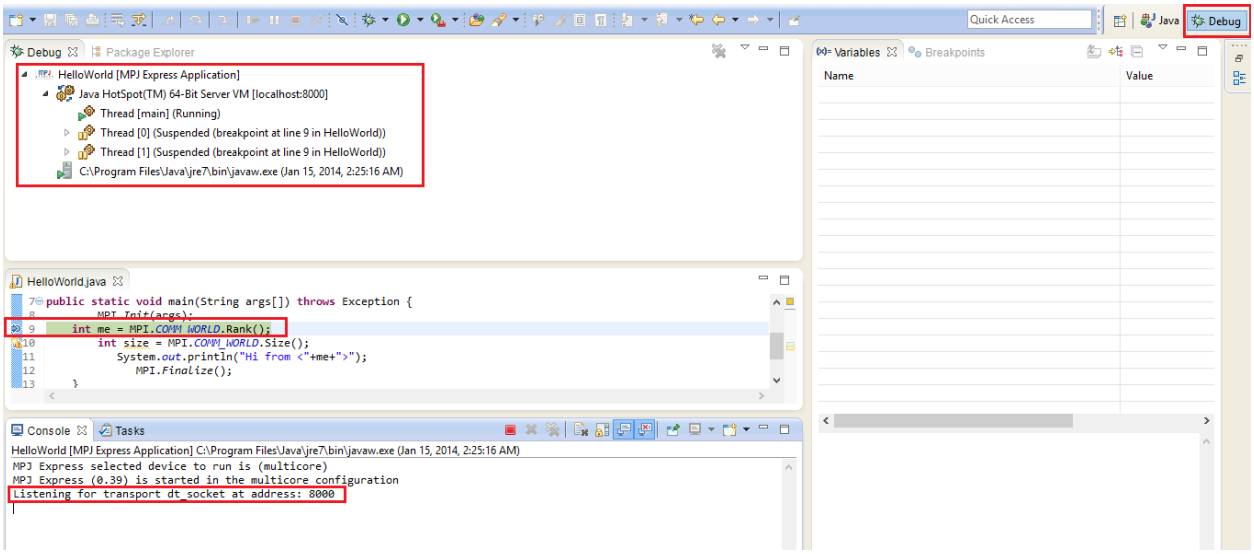


Figure 13: Debug Perspective for MPJ Express Debugging

6. You can now select each thread and use Step Into, Step Over and Step Return functionality to debug your application.

Note: For niodev and hybdev you will need to write machines files and start daemons to work it properly. To get help on how to start daemons follow step 1 and Step 2 of section 4.5

5.2 Remote Debugging

There are two types of Remote Debugging that current MPJ-Express Debugger supports

- Multicore Remote Debugging
- Distributed Remote Debugging

Steps for both types are almost same.

1. Create a new MPJ-Express Application in eclipse. You can follow steps from 1 to 11 of section 4.3
2. Introduce breakpoints where you want to stop you MPJ Express Application.
3. Click on Run →Run Configurations. Create, manage, and run configuration window will open. Double click on MPJ Express Application option. A new configuration with "HelloWorld" will be added.
    a. Browse for MPJ Express root directory and set field MPJ_HOME
    b. Enter number of processes in np field
    c. Enter Device name [multicore, niodev, hybdev] in device field.
    d. Add a new parameter with **Name = debug** and **value = 8000**. Note 8000 is port number, you can use whatever port you want provided port you chose if available.
4. Click Apply and Run

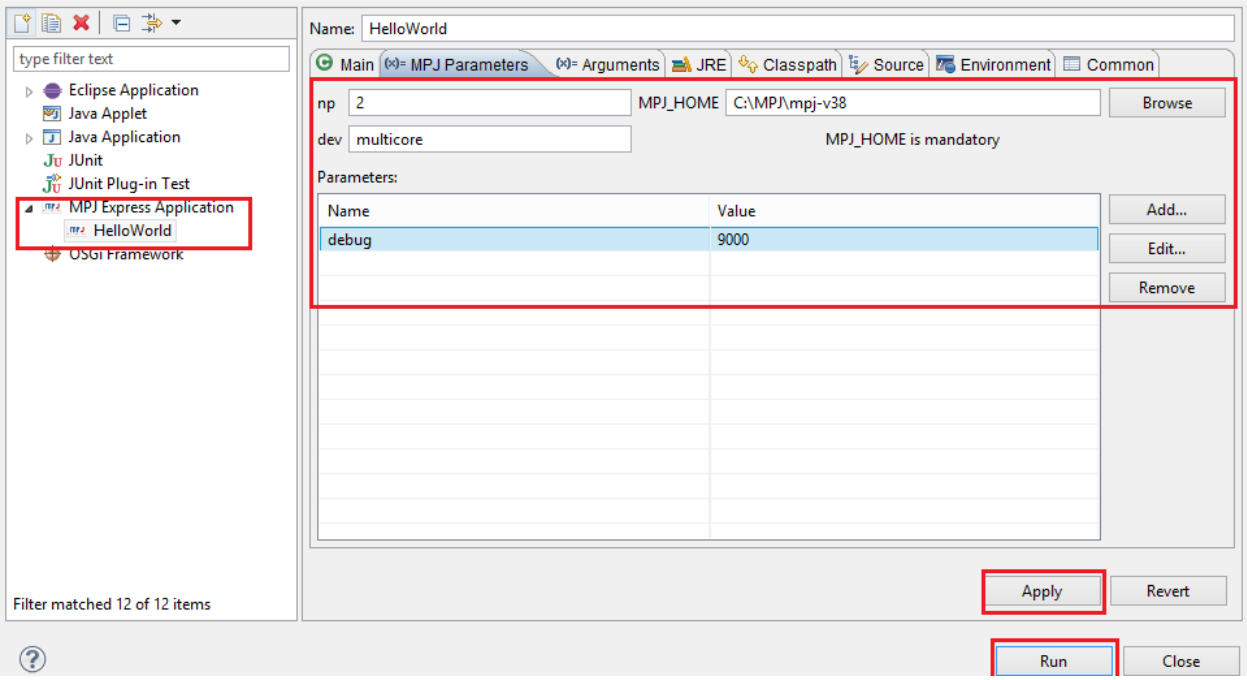


Figure 14: MPJ Parameters for Remote Debugging

5. Application will open in Debug perspective with application listening to your port defined in debug parameter.
6. Now go to Run → Debug Configuration and select Remote MPJ Express Application and double click on it. A new configuration will be added

- **For Multicore Remote Debugging**
    a. Open connect tab and select multicore as Connection type
    b. Enter Host value, localhost or IP address of system where MPJ-Express application is running
    c. Enter Port value. This should be same what you defined in your debug parameter of MPJ Express Parameters
    d. Click Apply and Debug

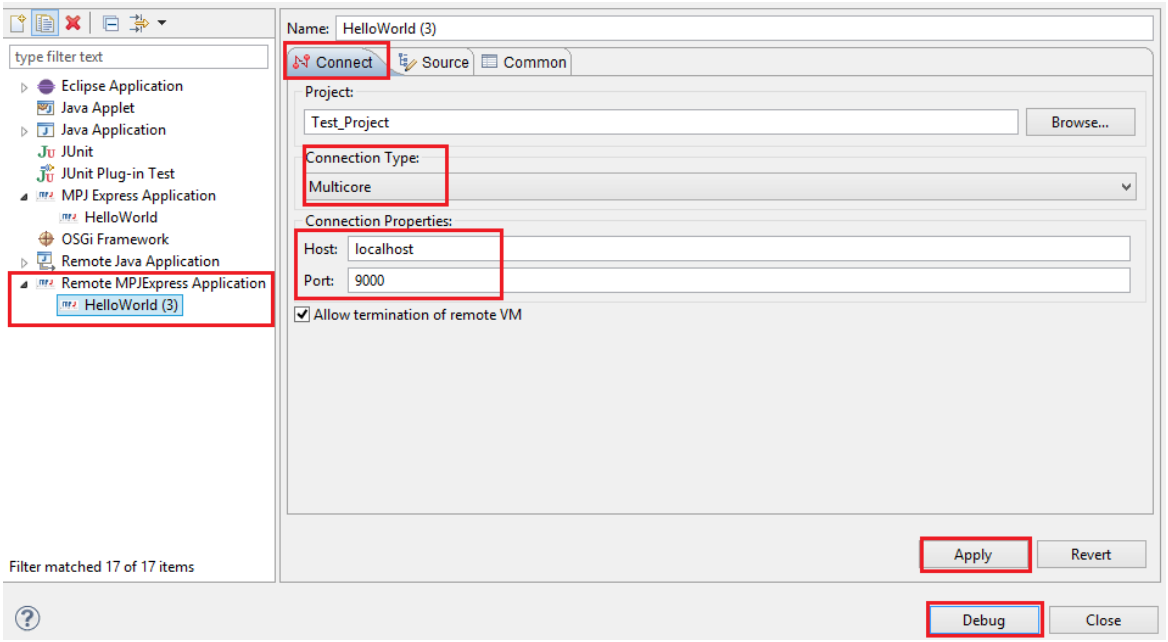


Figure 15: Connect Parameters for Remote Multicore Configuration

MPJ Express Application will start remotely debugging in multicore mode

- **For Distributed (niodev/hybdev) Remote Debugging**
    a. Open connect tab and select Distributed Memory as Connection type
    b. Browse for Configuration file (mpjdev.conf). By default path will be set to user directory where it is placed. mpjdev.conf must be accessible to workstation where your debug client is running.
    c. Click Apply and Debug

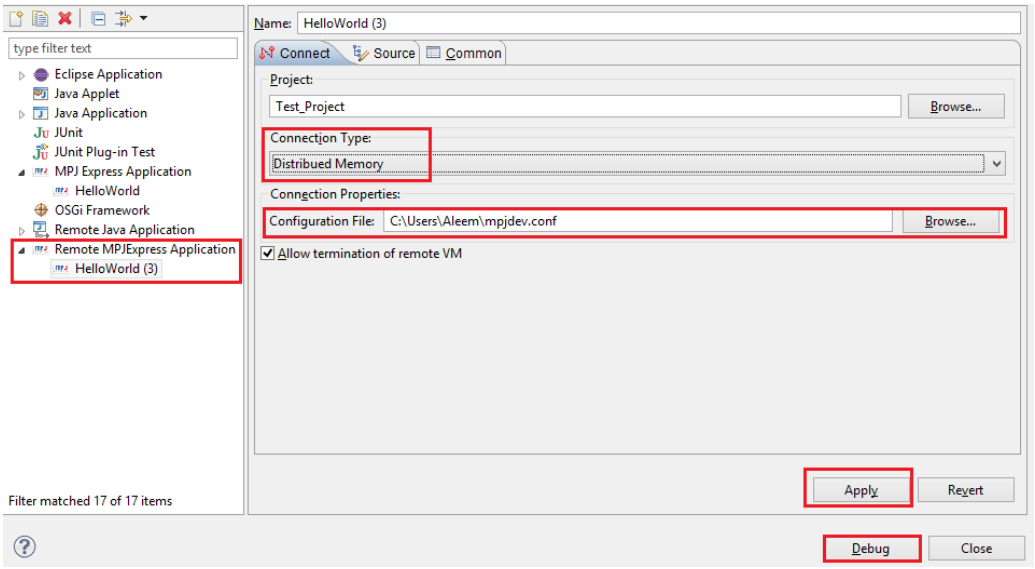


Figure 16: Connect Parameters for Remote Distributed Debugging

MPJ-Express application will start running in niodev/hybdev configuration debug mode. You can debug your application using Step Into, Step out, Step Return functionalities.

Note: For niodev and hybdev you will need to write machines files and start daemons to work it properly. To get help on how to start daemons follow step 1 and Step 2 of section 4.5

6. Known Issues and Limitations

A list of known issues and limitations of the MPJ Express Debugger are listed below.

1. Users of the Windows Vista and 7 might find install mpjd-windows.bat script not working directly. In such case, try executing with "Run as Administrator". If the issue still persists, try executing the script mpjdaemon.bat directly or turning off the firewall.

2. If you face trouble with Remote VM Connection failed. Run Eclipse as Administrator. Linux user can start Eclipse as Administrator through command line.

   Failed to connect to remote VM. Connection refused.
   Connection refused

3. If MPJ Express application fail to run and get stuck at this line: MPJ Express (0.38) is started in the cluster configuration
   Then just restart daemons and your application will start running fine.
4. If you are running your applications on Rocks cluster then you might get below mentioned error. To solve this just add a new parameter in MPJ Parameters Tab with **name=wdir and value=absolute path to bin directory of your project**.

```
java.io.IOException: Cannot run program "java" (in directory "/state/partition1/home2/MPJ_Work/MPJ_Test"): error=2,
        at java.lang.ProcessBuilder.start(ProcessBuilder.java:1029)
        at runtime.daemon.HybridDaemon.startNewProcess(HybridDaemon.java:323)
        at runtime.daemon.HybridDaemon.<init>(HybridDaemon.java:124)
        at runtime.daemon.MPJDaemon.<init>(MPJDaemon.java:272)
        at runtime.daemon.MPJDaemon.main(MPJDaemon.java:1242)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:601)
        at runtime.daemon.WrapperSimpleApp.run(WrapperSimpleApp.java:187)
        at java.lang.Thread.run(Thread.java:722)
Caused by: java.io.IOException: error=2, No such file or directory
        at java.lang.UNIXProcess.forkAndExec(Native Method)
        at java.lang.UNIXProcess.<init>(UNIXProcess.java:135)
        at java.lang.ProcessImpl.start(ProcessImpl.java:130)
        at java.lang.ProcessBuilder.start(ProcessBuilder.java:1021)
        ... 10 more
```

Figure 17: State partition error

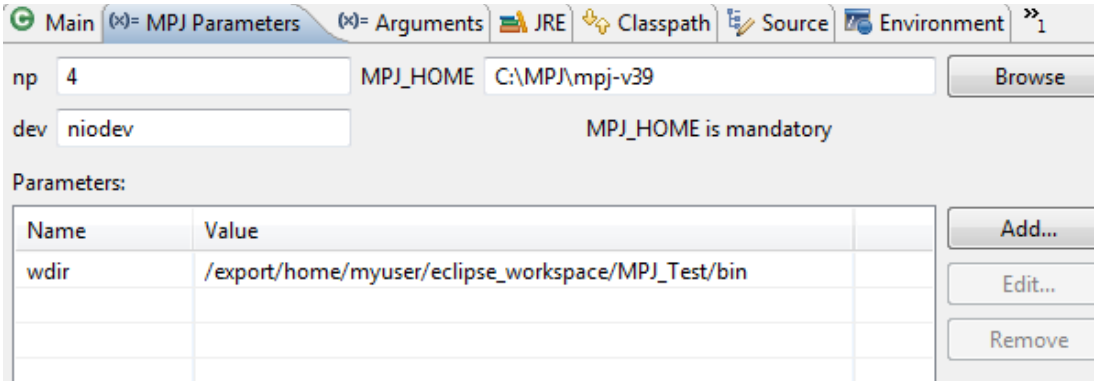


Figure 18: Solution of State partition error

7. Contact and Support

For help and support, join and post on the MPJ Express mailing list (mpjexpress-users@lists.sourceforge.net). Alternatively, you may also contact us directly:

1 Aamir Shafi (aamir.shafi@seecs.edu.pk)

2 Mohsan Jameel (mohsan.jameel@seecs.edu.pk)

3 Bryan Carpenter (bryan.carpenter@port.ac.uk)

4 Mark Baker (http://acet.rdg.ac.uk/~mab)

5 Guillermo Lopez Taboada (http://www.des.udc.es/~gltaboada)

8. Appendices

Appendix A: Setting Environment Variables in Windows

1. 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 4,5 and 6]
    b. Append the value of variable Path as c:\mpj\bin [Fig 7]

    See the snapshots below

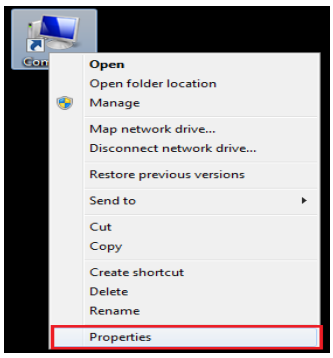


Figure 19: Right click on my computer and select Properties

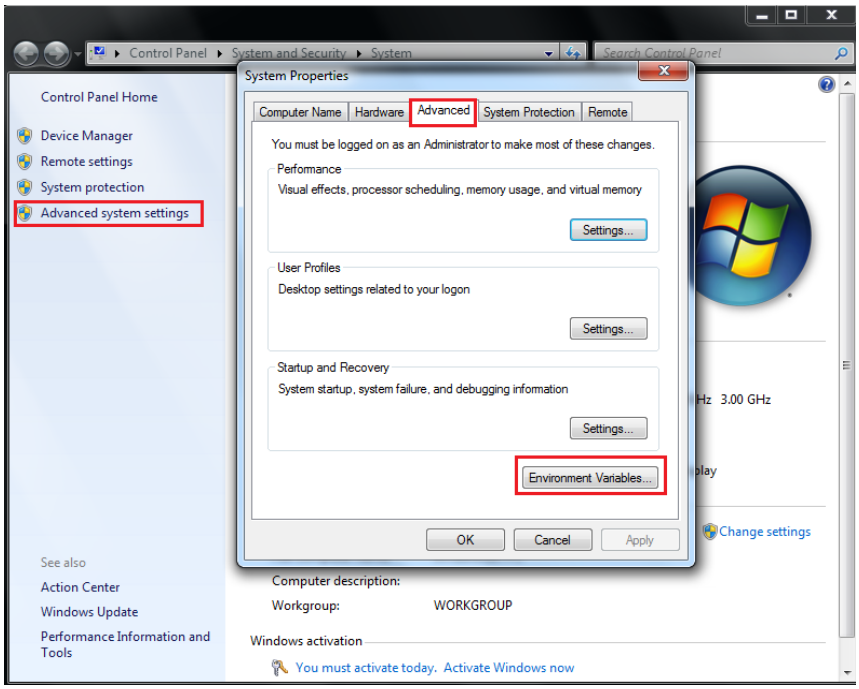


Figure 20: Select Environment Variables to Add/Edit variables

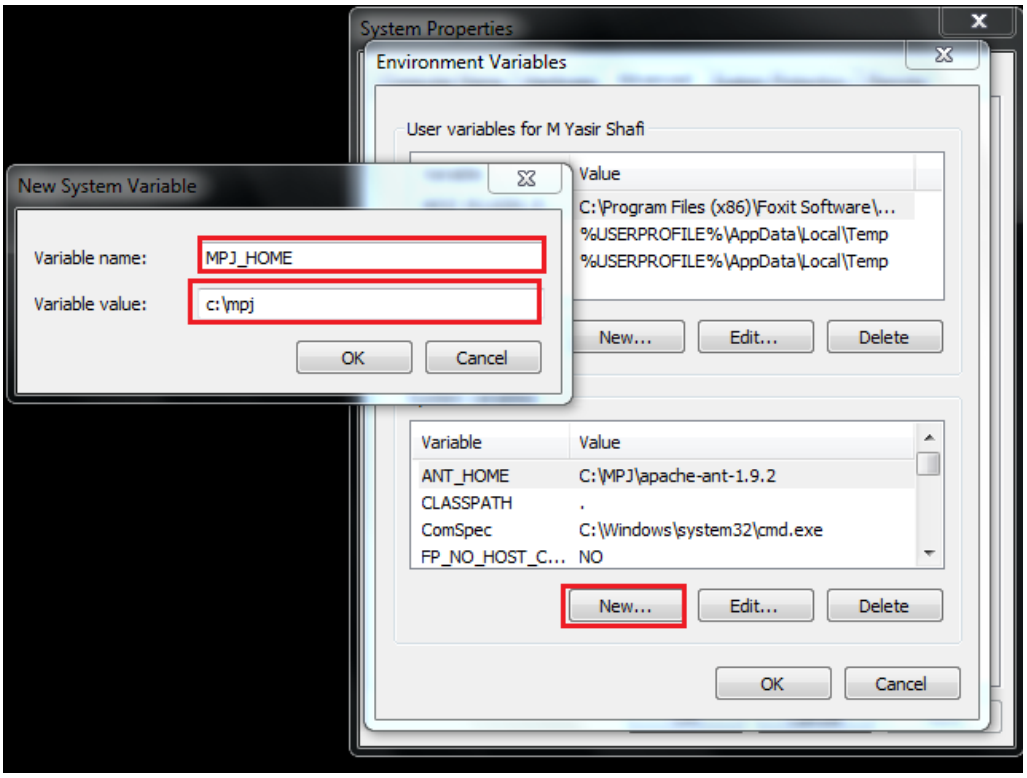


Figure 21: Add MPJ_HOME as new Environment Variable

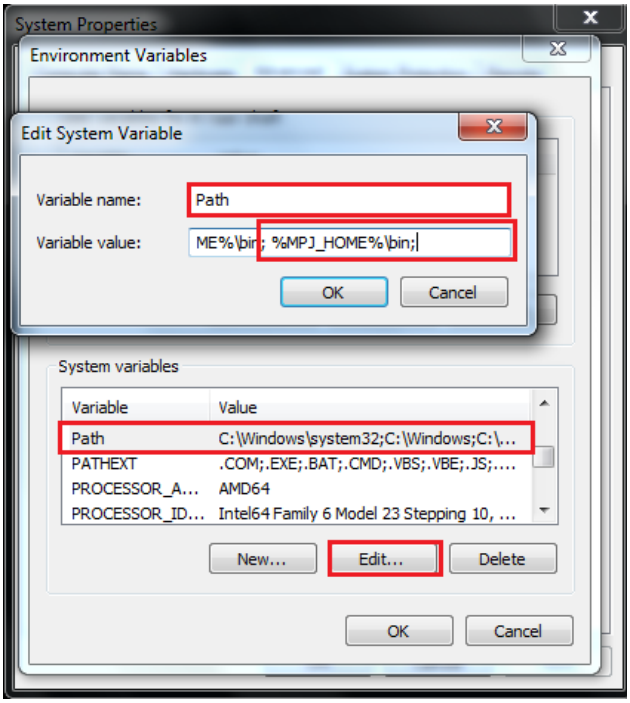


Figure 22: Append Path variable

2. 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"
```

Appendix B: Setting Environment Variables in Linux

To set MPJ_HOME and PATH variables, open .bashrc file and add these lines.

```
export MPJ_HOME=/path/to/unpacked mpj express/

export PATH=$PATH:$MPJ_HOME/bin
```

These lines may be added to “.bashrc” file. However make sure that the shell in which you are setting variables is the ‘default’ shell. For example, if your default shell is ‘bash’, then you can set environment variables in .bashrc. If you are using ‘tcsh’ or any other shell, then set the variables in the respective files.