Assignment 3 MPI Tutorial
Compiling and Executing MPI programs

B. Wilkinson: Modification date: February 11, 2016.

This assignment is a tutorial to learn how to execute MPI programs and explore their characteristics. Mostly you will test the programs on your own computer using the provided virtual machine or with a native Linux installation. The programs are provided. Both command line (Part 1) and using Eclipse (Part 2) are explored. The purpose of using both the command line and Eclipse is so that you can see what is involved in each way. You will be asked to compare and contrast the two ways. In Part 3, you will test the programs on a remote cluster and measure the speedup. You will have to modify the matrix multiplication program to make it work on the cluster. The matrix multiplication code, as written, only works if ***P*** (the number of processes) divides evenly into ***N*** (the number or rows and columns in the matrices). Part 4 asks you to modify the program to handle any value of ***N***.

Preliminaries – Software Environment

MPI is process-based where individual processes can execute at the same time on local or distributed computers and explicit message-passing routines are used to pass data between the processes. You will need MPI software to compile and execute MPI programs. The provided VM has OpenMPI and Eclipse-PTP already installed. The sample MPI programs and data files are in **~/ParallelProg/MPI**. Eclipse-PTP is needed for Part 2.

If you are using your own Linux installation, you will need to install MPI software, such as OpenMPI or MPICH. Unfortunately there are differences in these implementations that make some MPI programs not transferrable as we shall see. *For consistency install OpenMPI*. How to install OpenMPI is described at the link "Additional Information" on Moodle (see "Installing Software"). Instructions on installing Eclipse-PTP, and the sample MPI programs and data files can be found there.

Cd to the directory where the sample MPI programs reside (**~/ParallelProg/MPI)**.

Part 1 Using Command Line (30%)

Hello World Program

A simple hello world program, called **hello.c**, is given below demonstrating MPI sends and receives. The program is provided within the MPI directory.

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char **argv ) {
	char message[256];
	int i,rank, size, tag=99;
	char machine_name[256];
```

```
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    gethostname(machine_name, 255);

    if(rank == 0) {
        printf ("Hello world from master process %d running on %s\n",rank,machine_name);
        for (i = 1; i < size; i++) {
            MPI_Recv(message, 256, MPI_CHAR, i, tag, MPI_COMM_WORLD, &status);
            printf("Message from process = %d : %s\n", i, message);
        }
    } else {
        sprintf(message, "Hello world from process %d running on %s",rank,machine_name);
        MPI_Send(message, 256, MPI_CHAR, 0, tag, MPI_COMM_WORLD);
    }

    MPI_Finalize();
    return(0);
}
```

MPI routines are shown in red.

Task 1 Compiling and Executing Hello World Program

Compile and execute the hello world program. To compile, issue the command:

```
mpicc hello.c -o hello
```

from the **MPI** directory. Execute with the command:

```
mpiexec -n 4 hello
```

which will use four processes. In general, we do not have specify the current directory (e.g. ./hello). So far, the four instances of the program will execute just on one computer. You should get the output:

```
Hello world from master process 0 running on …
Message from process = 1 : Hello world from process 1 running on …
Message from process = 2 : Hello world from process 2 running on …
Message from process = 3 : Hello world from process 3 running on …
```

Comment on the how MPI processes map to processors/cores. Try 16 processes and see the CPU usage (in Windows, the Task Manager).

Experiments with Code

Modify the hello world program by specifying the rank and tag of the receive operation to MPI_ANY_SOURCE and MPI_ANY_TAG, respectively. Recompile the program and execute.

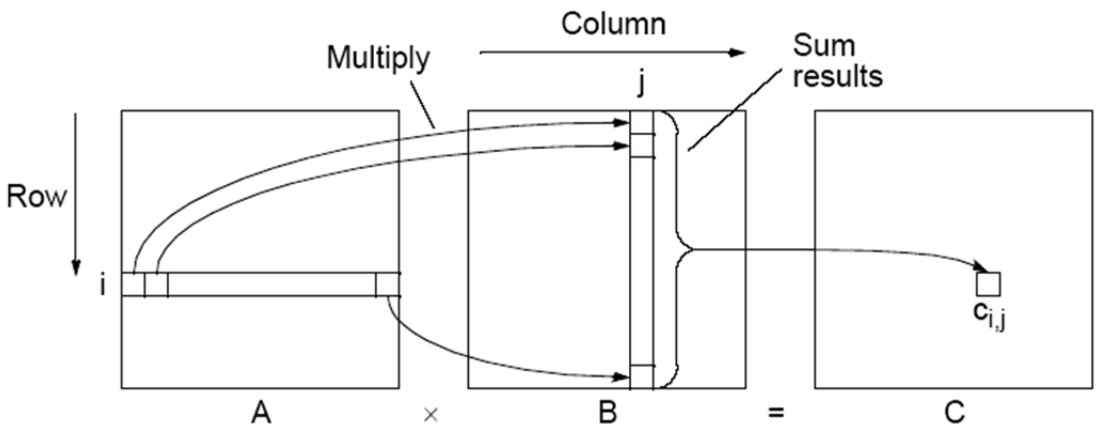
Is the output in order of process number? Why did the first version of hello world sort the output by process number but not the second?

Include in your submission document for Task 1:

1. A screenshot or screenshots showing:
    a. Compilation of the hello world program
    b. Executing the program with its output
2. The effects of changing the number of processes
3. The effects of using wild cards (MPI_ANY_SOURCE and MPI_ANY_TAG)
4. Answer to the question about process order.

Task 2 Compiling and Executing Matrix Multiplication Program

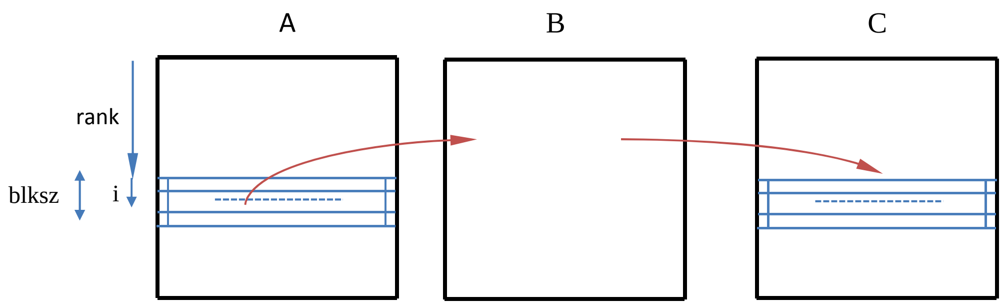
Multiplication of two matrices, **A** and **B**, produces matrix **C** whose elements, $c_{i,j}$(0 <= *i* <*n*, 0 <= *j* <*m*), computed as follows:$C_{i,j} = \sum_{k=0}^{l-1} a_{i.j}\, b_{k,j}$ where **A** is an *n* x *l* matrix and **B** is a *i* x *m* matrix:



The sequential code to compute **A** x **B** (assumed square *N* x *N*) could simply be:

```
for (i = 0; i < N; i++)              // for each row of A
   for (j = 0; j < N; j++) {         // for each column of B
      c[i][j] = 0;
      for (k = 0; k < N; k++)
         c[i][j] = c[i][j] + a[i][k] * b[k][j];
   }
```

It requires N^3 multiplications and N^3 additions with a sequential time complexity of O(N^3). It is very easy to parallelize as each result is independent. Usually we map one process onto each processor so process and processor are the same, but not necessarily. (*It is important not to confuse process with processor.)* Often the size of matrices (*N*) is much larger than number of processes (*P*), so rather than having one process for each result, we can have each process compute a group of result elements. In MPI, a convenient arrangement is to take a group of rows of **A** and multiply that with **B** to create a groups of rows of **C**, which can then be gathered using MPI_Gather():



This does require **B** to be broadcast to all processes. A simple MPI matrix multiplication program, called **matrixmult.c**, is given below and provided within the MPI directory:

```
#define N 256
#include <stdio.h>
#include <time.h>
#include "mpi.h"

int main(int argc, char *argv[])      {
      int i, j, k, error = 0;
      double A[N][N], B[N][N], C[N][N], D[N][N], sum;

      double time1, time2;        // use clock for timing

      MPI_Status status;          // MPI variables
      int rank, P, blksz;

      MPI_Init(&argc, &argv);          // Start MPI
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
      MPI_Comm_size(MPI_COMM_WORLD, &P);

      if ((rank == 0) && (N % P != 0)) {
            printf("Error -- N/P must be an integer\n"); // should really stop now
            MPI_Abort(MPI_COMM_WORLD,1);
      }

      blksz = N/P;

      if (rank == 0) {
            printf ("N = %d P = %d\n",N, P);
            for (i = 0; i < N; i++) {        // set some initial values for A and B
                  for (j = 0; j < N; j++) {
                        A[i][j] = j*1;
                        B[i][j] = i*j+2;
                  }
            }
            for (i = 0; i < N; i++) {        // sequential matrix multiplication
                  for (j = 0; j < N; j++)   {
                        sum = 0;
                        for (k=0; k < N; k++) {
                              sum += A[i][k]*B[k][j];
                        }
                        D[i][j] = sum;
                  }
            }
            time1 = MPI_Wtime();        // record  time stamp
      }
```

```
    MPI_Scatter(A, blksz*N, MPI_DOUBLE, A, blksz*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);// Scatter input
matrix A
    MPI_Bcast(B, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD); // Broadcast the input matrix B

    for(i = 0 ; i < blksz; i++) {
        for(j = 0 ; j < N ; j++) {
            sum = 0;
            for(k = 0 ; k < N ; k++) {
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }

    MPI_Gather(C, blksz*N, MPI_DOUBLE, C, blksz*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if(rank == 0) {
        time2 = MPI_Wtime();       // record  time stamp

        int error = 0;       // check sequential and parallel versions same answers, within rounding
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++)   {
                if ( (C[i][j] - D[i][j] > 0.001) || (D[i][j] - C[i][j] > 0.001)) error = -1;
            }
        }
        if (error == -1) printf("ERROR, sequential and parallel code give different answers.\n");
        else printf("Sequential and parallel code give same answers.\n");

        printf("elapsed_time =\t%lf (seconds)\n", time2 - time1);  // print out execution time
    }
    MPI_Finalize();
    return 0;
}
```

Actually the file was corrected.

~~The file matrixmult.c provided within the MPI directory of the VM (and on the course home) page has a mistake. It uses clock() for timing, which may not give accurate results as clock() does not record the time if the process is de-scheduled as might be the case in the message passing or otherwise, So correct the provided file to use MPI_Wtime() instead of clock().~~ Also add MPI_Abort(MPI_COMM_WORLD,1); where shown above to make sure program terminates immediately if *N*/*P* is not an integer.

Compile and execute the matrix multiplication program. To compile, issue the command:

```
mpicc matrixmult.c -o matrixmult
```

from the **MPI** directory.

Execute the program:

```
mpiexec -n 4 matrixmult
```

One can experiment with a different number of processes. Unfortunately generally one will not see a particular increase in speed on a personal computer because of the message passing overhead. (VirtualBox limits the number of cores available. Go to **Machines** > **Settings** > **System** > **Processors** without any VM running to alter and reboot the OS.)

Include in your submission document for Task 2:

A screenshot or screenshots showing:

a. Compilation of the multiplication program
b. Executing the program with its output, with different numbers of processes

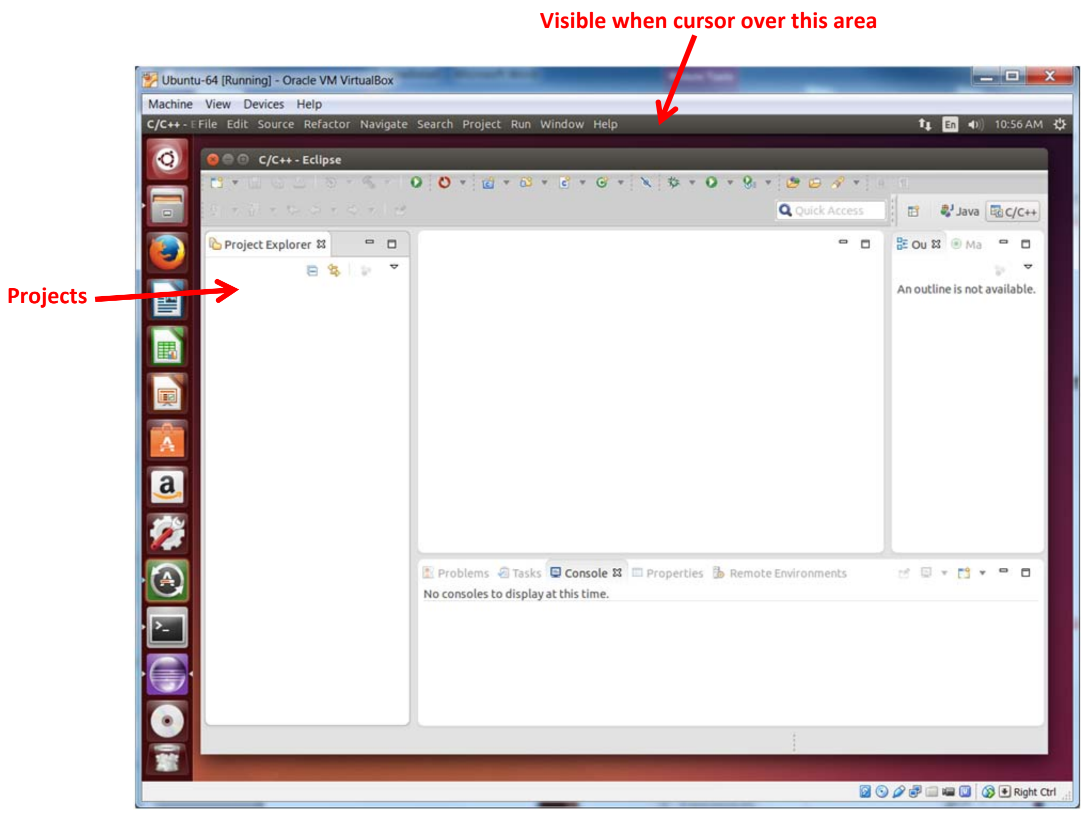
Part 2 Eclipse-PTP (30%)

Eclipse-PTP (Eclipse with the tools for Parallel Applications Developers) can be used to compile and execute MPI programs. In Eclipse, there are different project types for different environments. MPI programs are done as C/C++ projects with build/compilation for MPI pre-configured in Eclipse-PTP. Programs here will be C projects.

Start Eclipse on the command line by typing:

```
eclipse
```

Create/select a workbench location within the MPI directory (i.e. **~/ParallelProg/MPI/workspace)** and go to the workbench. This workspace will be empty until we create projects for the programs we want to execute:


Visible when cursor over this area
Projects
Ubuntu-64 [Running] - Oracle VM VirtualBox
Machine View Devices Help
C/C++ - File Edit Source Refactor Navigate Search Project Run Window Help
10:56 AM
C/C++ - Eclipse
Quick Access
Java
C/C++
Project Explorer
Ou
Ma
Problems
Tasks
Console
Properties
Remote Environments
Right Ctrl

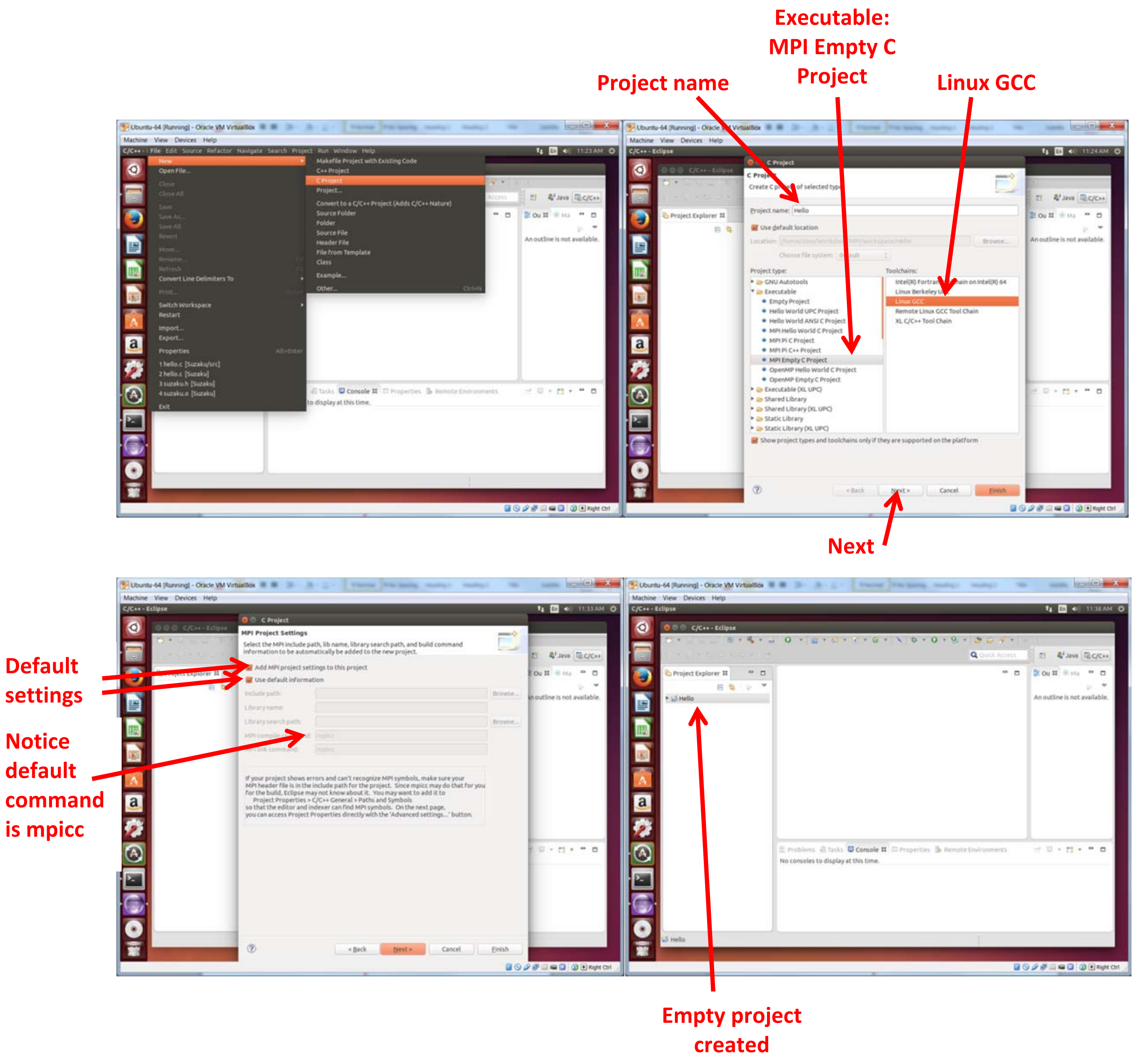
An outline is not available.

No consoles to display at this time.

Task 1 Creating and executing Hello world program through Eclipse

(a) Creating project and adding source files

Create new C project (**File** > **New** > **C project**) called **Hello** of type "**MPI Empty C project**"[1] with default settings:



C Project

Create C project of selected type

Project name: Hello

[✓] Use default location

Project type: Executable > MPI Empty C Project

Toolchains: Linux GCC

[✓] Show project types and toolchains only if they are supported on the platform

MPI Project Settings

Select the MPI include path, lib name, library search path, and build command information to be automatically be added to the new project.

[✓] Add MPI project settings to this project

[✓] Use default information

MPI compile command: mpicc

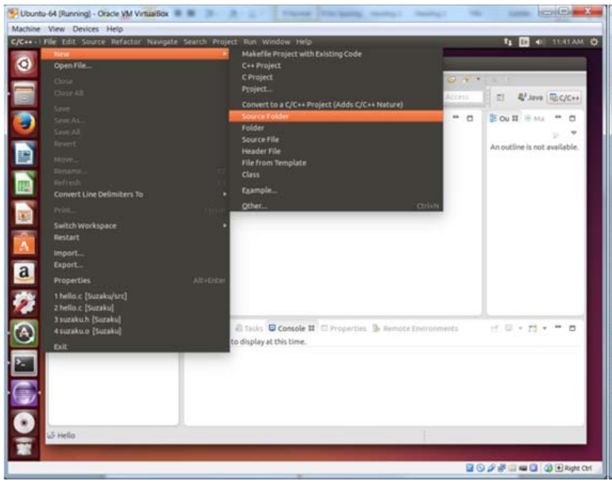
MPI link command: mpicc

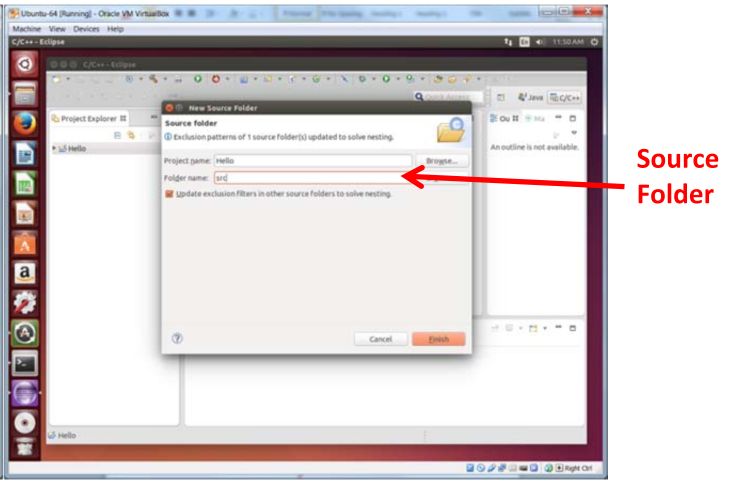
If your project shows errors and can't recognize MPI symbols, make sure your MPI header file is in the include path for the project. Since mpicc may do that for you for the build, Eclipse may not know about it. You may want to add it to
    Project Properties > C/C++ General > Paths and Symbols
so that the editor and indexer can find MPI symbols. On the next page, you can access Project Properties directly with the 'Advanced settings...' button.

[1] Eclipse comes with sample programs pre-installed for C, OpenMP, and MPI ("Hello World" and MPI Pi), which are useful for testing the environment.

Adding program source file

Select Hello project and create a source folder called **src (File > New >Source Folder** or **right click project > New >Source Folder)**:





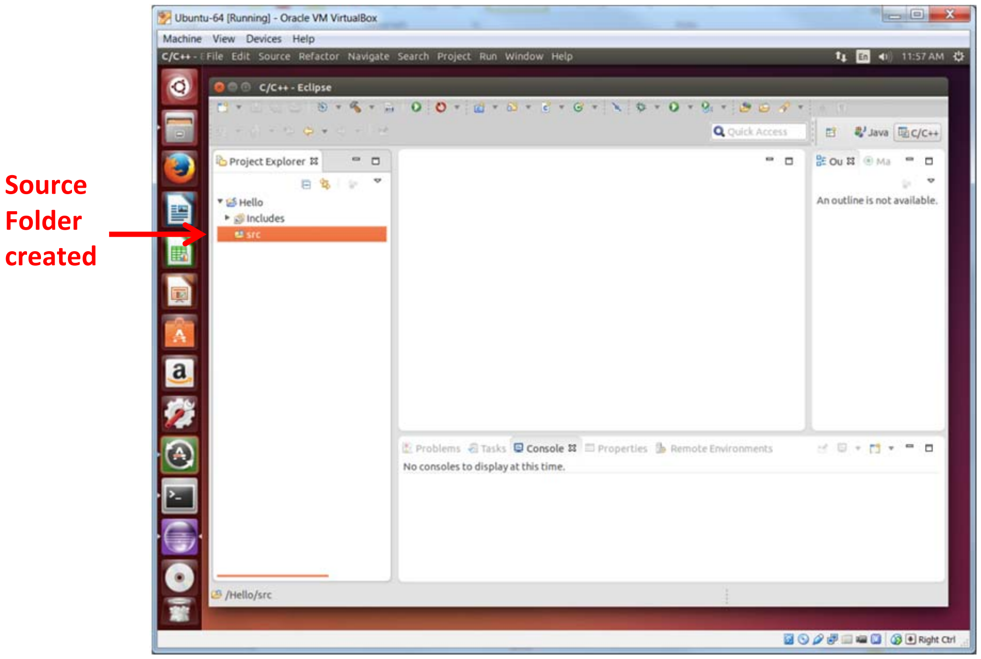
Source folder

Exclusion patterns of 1 source folder(s) updated to solve nesting.

Project name: Hello

Folder name: src

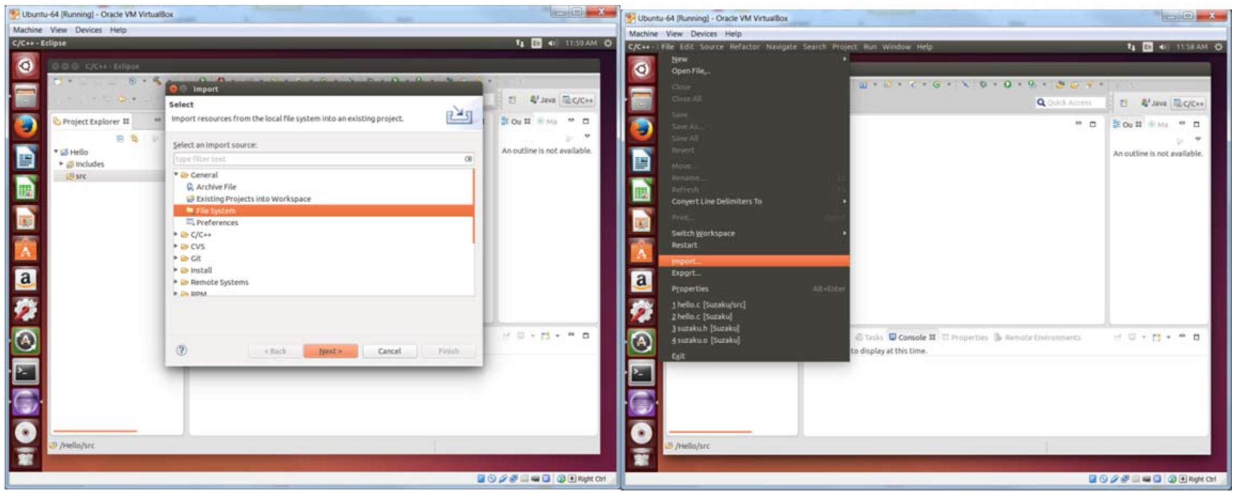
Update exclusion filters in other source folders to solve nesting.



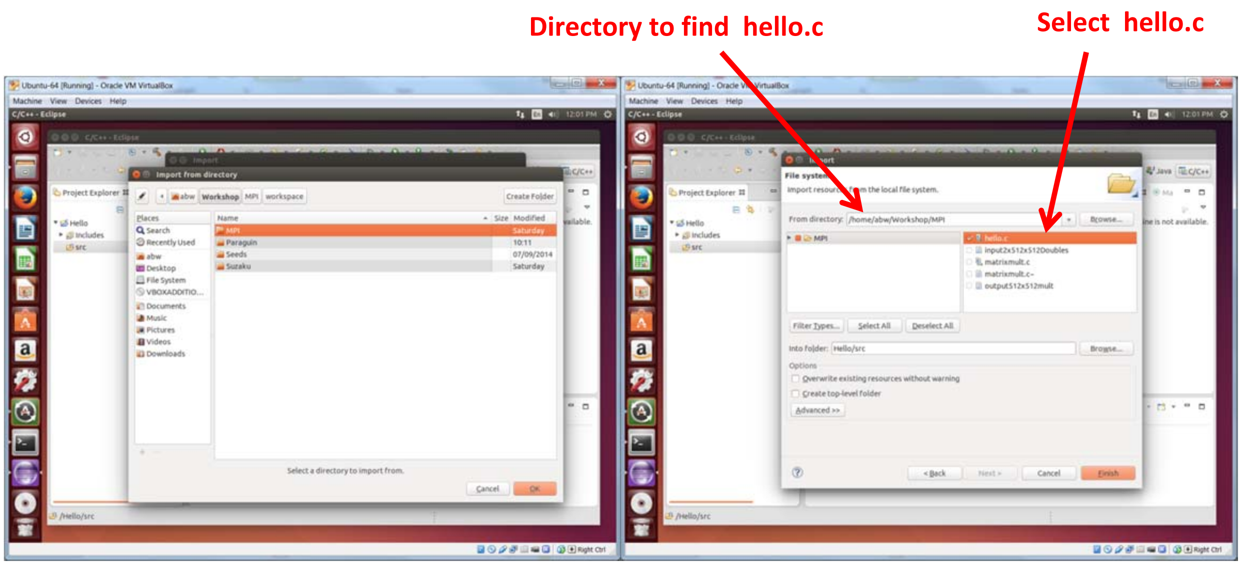
No consoles to display at this time.

An outline is not available.

Expand the **Hello** project and select the newly generated **src** folder. Select **File > Import > General > File System:**



From **"Import from directory"**, browse for the directory that holds your **hello.c** files (**~/ParallelProg/MPI/**). Click **OK**. Select the **hello.c** file to copy into the **MPI/src** project folder.[2]



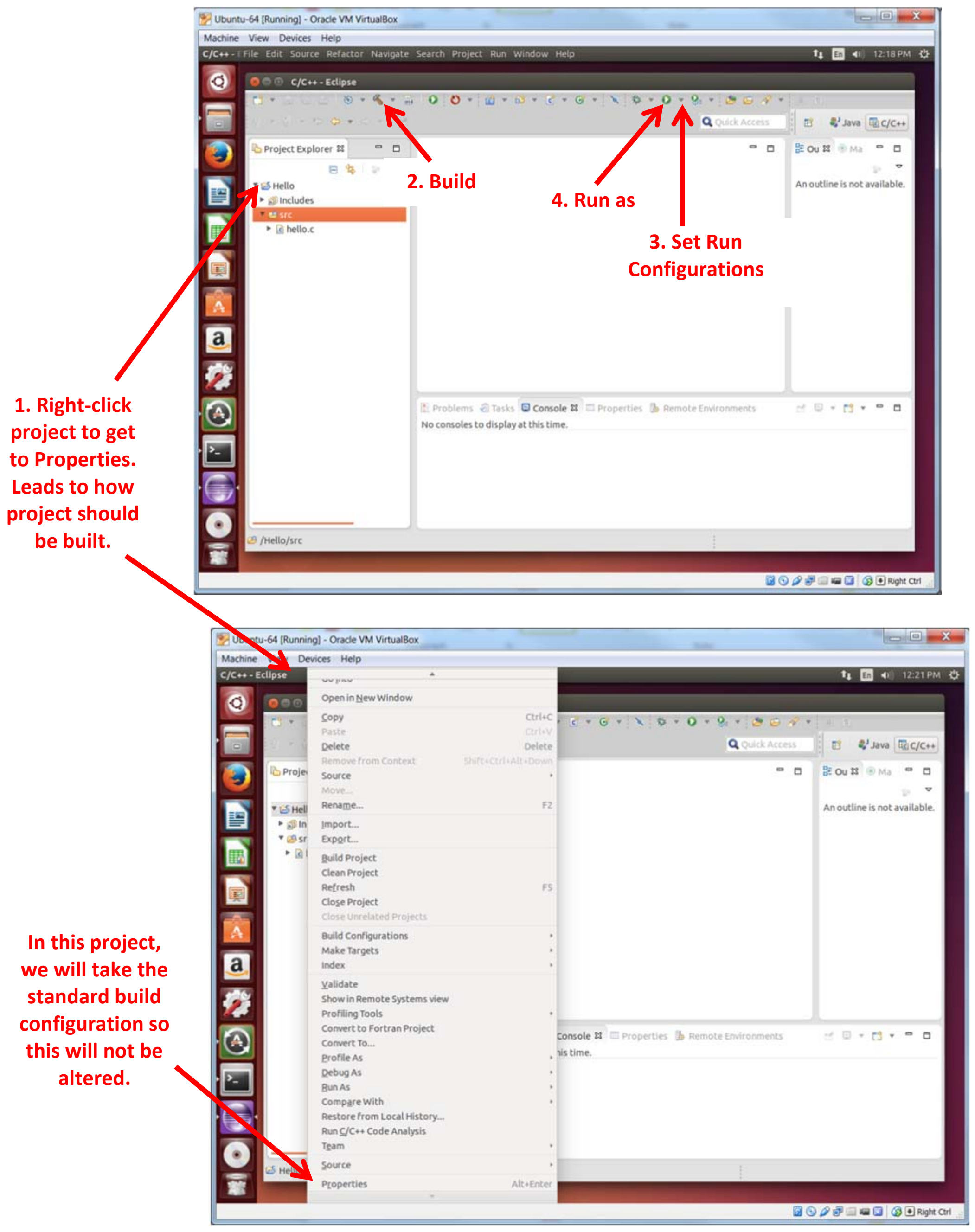
Note: The paths to files shown in the screenshots may not be the same as shown. You will need to find their locations on your system.

[2]In a later example, we will demonstrate using a link to the source file at their original location rather than copying it, which is usually better.

(b) Build and Execute Program

Basic steps:

1. Set how to build (compile) project in **Properties > ... Build**
2. Build project (compile to create executable)
3. Set how to execute compiled program in **Run Configurations**
4. **Run** (execute) using the specified run configurations


2. Build

4. Run as

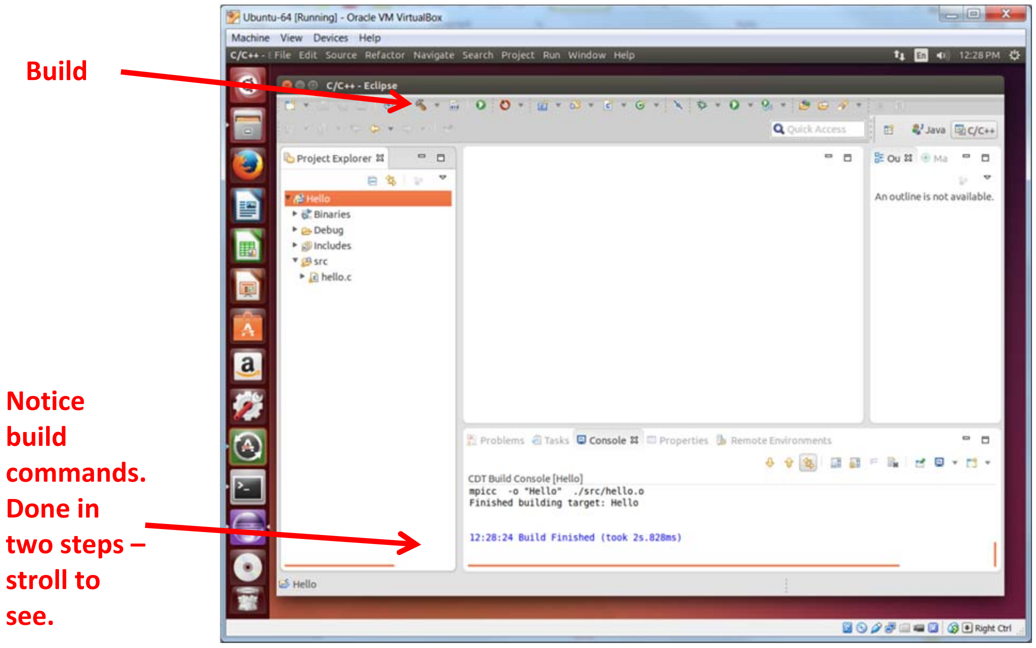
3. Set Run Configurations

1. Right-click project to get to Properties. Leads to how project should be built.

In this project, we will take the standard build configuration so this will not be altered.

Build project

Click **Build** icon (Hammer) to build the project (default “Debug” option):


Build
Notice build commands. Done in two steps – stroll to see.
Ubuntu-64 [Running] - Oracle VM VirtualBox
C/C++ - Eclipse
Project Explorer
Hello
Binaries
Debug
Includes
src
hello.c
An outline is not available.
Problems
Tasks
Console
Properties
Remote Environments

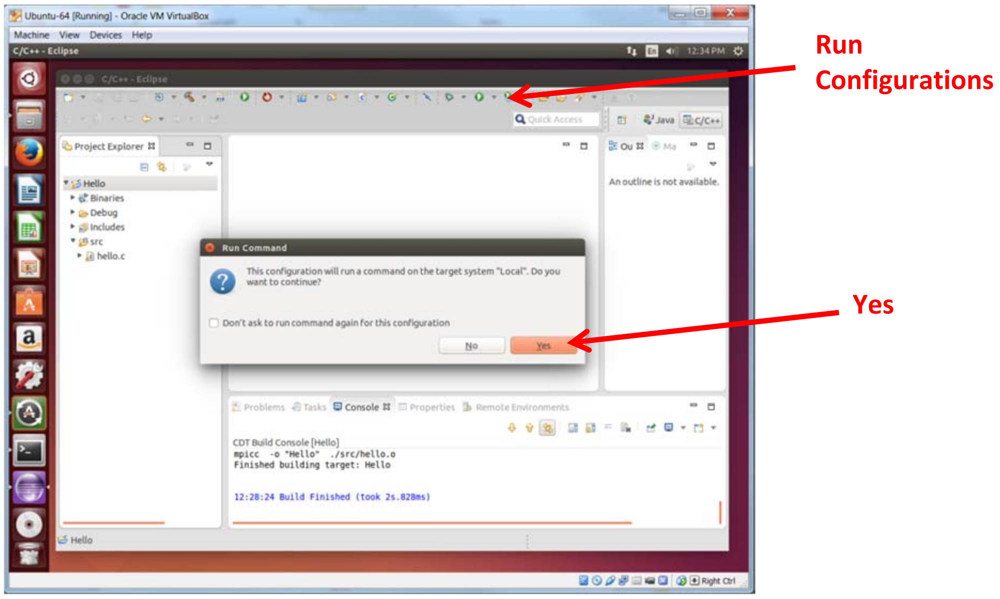
```
CDT Build Console [Hello]
mpicc  -o "Hello"  ./src/hello.o
Finished building target: Hello

12:28:24 Build Finished (took 2s.828ms)
```

Although building will happen automatically when a project is executed next, sometimes it is handy to know if there are any build errors first.

Execution

To execute the program, first the **Run Configurations** need to be set up that specify local execution, the software environment, etc. Select **Run Configurations**



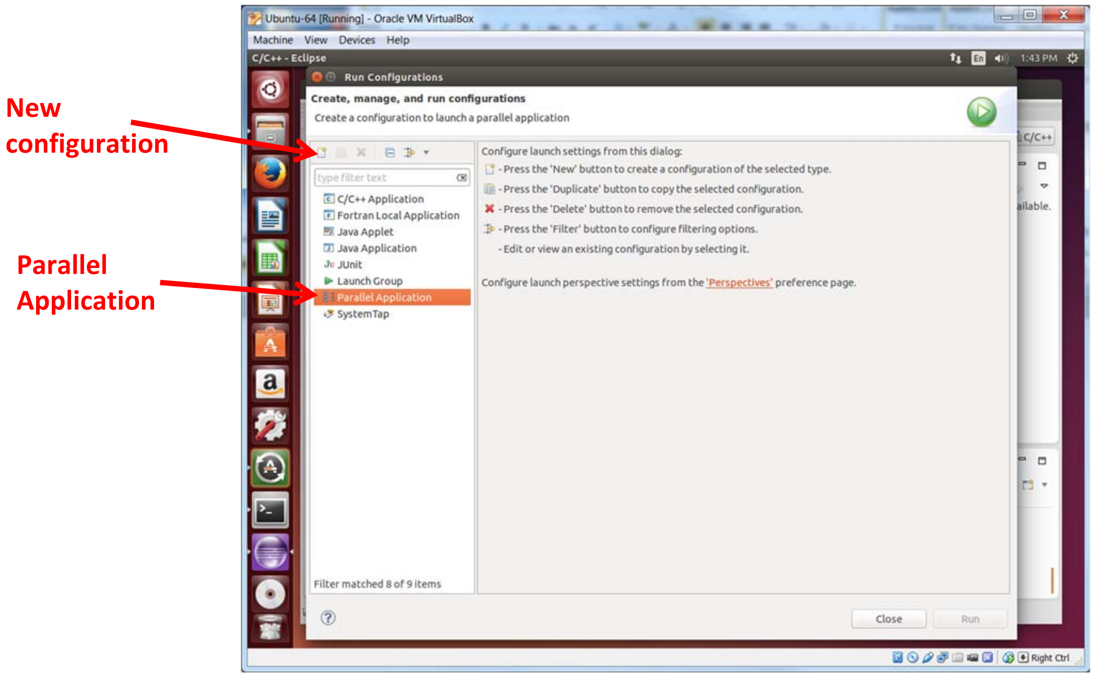
This configuration will run a command on the target system "Local". Do you want to continue?

Don't ask to run command again for this configuration

```
CDT Build Console [Hello]
mpicc  -o "Hello"  ./src/hello.o
Finished building target: Hello

12:28:24 Build Finished (took 2s.828ms)
```

Select “Parallel Application” and click the new configuration button:



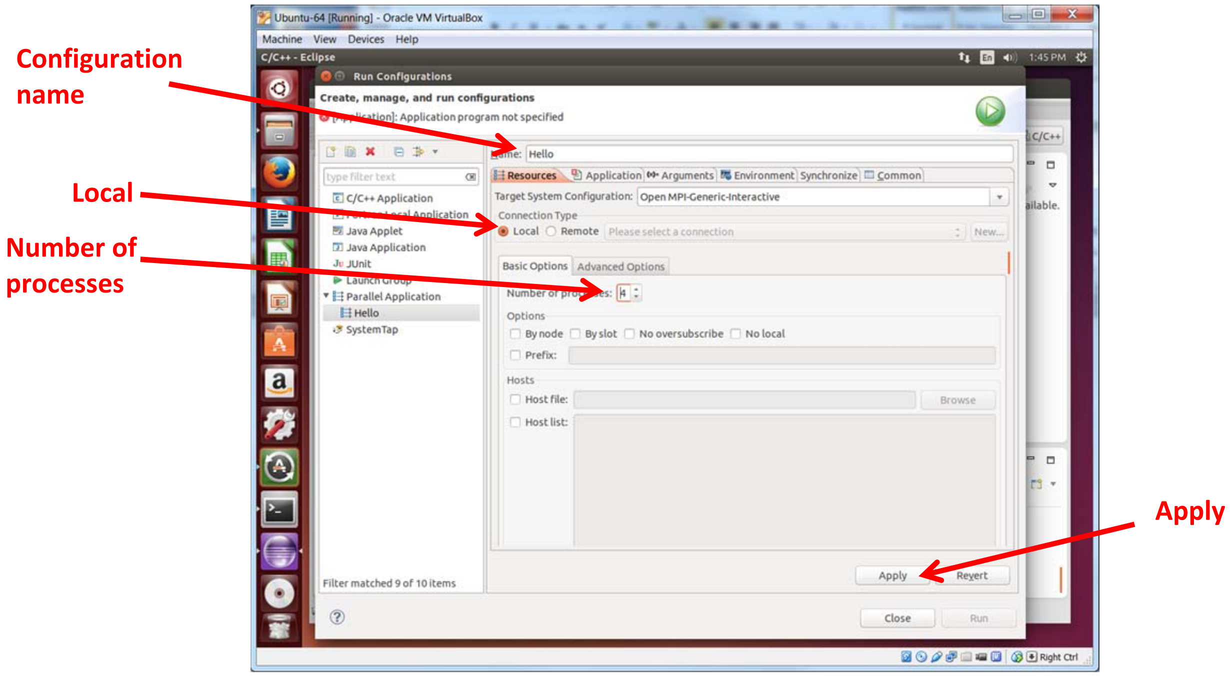
Create, manage, and run configurations

Create a configuration to launch a parallel application

Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Create a new run configuration called **Hello**. In the *Resource* tab, select the Target Type as “**Open-MPI-Generic-Interactive**”, the connection type as “**Local**”.[3] Set the number of processes to say 4.Apply.



Create, manage, and run configurations

[Application]: Application program not specified

Name: Hello

Target System Configuration: Open MPI-Generic-Interactive

Connection Type: [✓ Local] Remote Please select a connection New...

Number of processes: 4

Options: By node By slot No oversubscribe No local

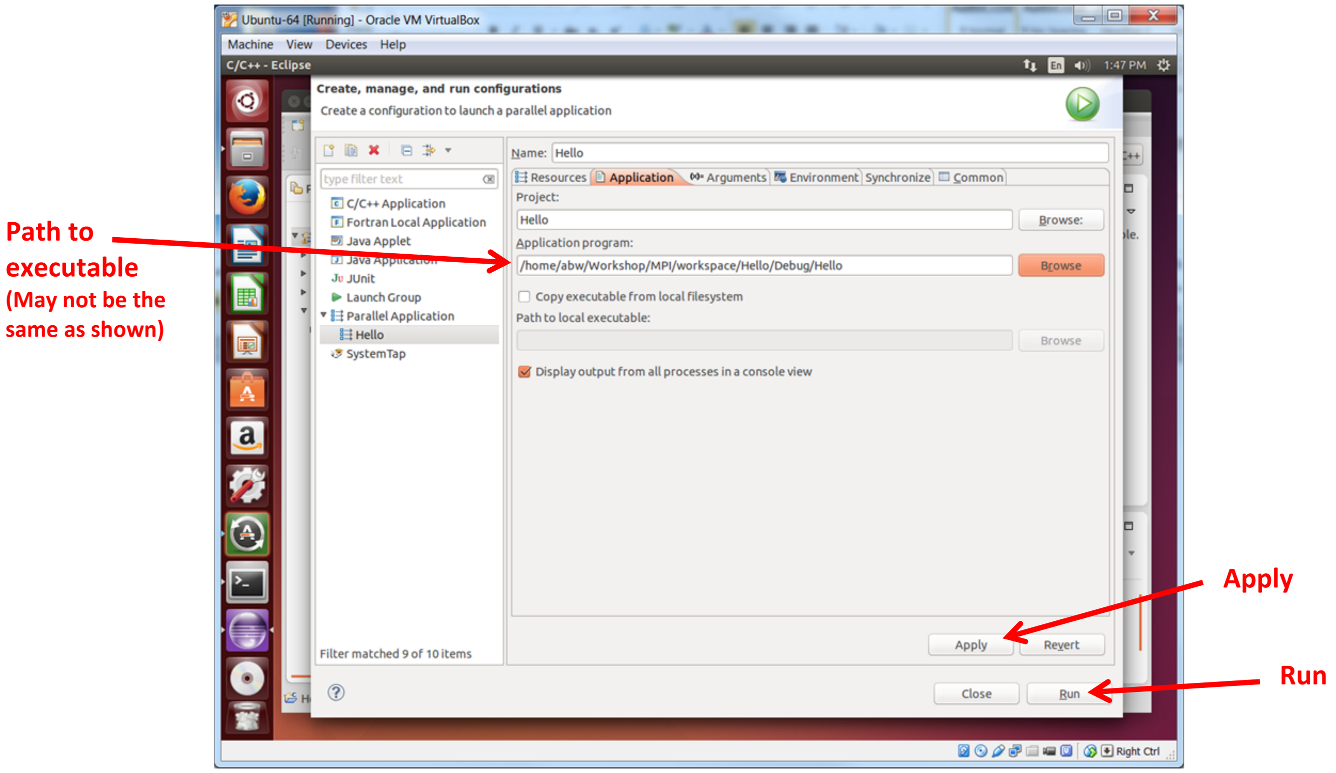
Prefix:

Hosts: Host file: Browse

Host list:

[3] Selecting “Local” will generate a message confirming you want this and create local resources to do this. Select “Don’t ask to run command again for this configuration” in the Run Command message when it appears to stop the message. The message is most relevant when doing both local and remote executions.

In the *Application* tab, set the Project name to “Hello” and browse for the path to the executable (**... /workspace/MPI/Debug/Hello**).


Path to executable
(May not be the same as shown)
Apply
Run
Ubuntu-64 [Running] - Oracle VM VirtualBox
Machine View Devices Help
C/C++ - Eclipse
Create, manage, and run configurations
Create a configuration to launch a parallel application
C/C++ Application
Fortran Local Application
Java Applet
Java Application
JUnit
Launch Group
Parallel Application
Hello
SystemTap
Filter matched 9 of 10 items
Resources
Application
Arguments
Environment
Synchronize
Common
Browse
Apply
Revert
Close
Run

Name: Hello

Project: Hello

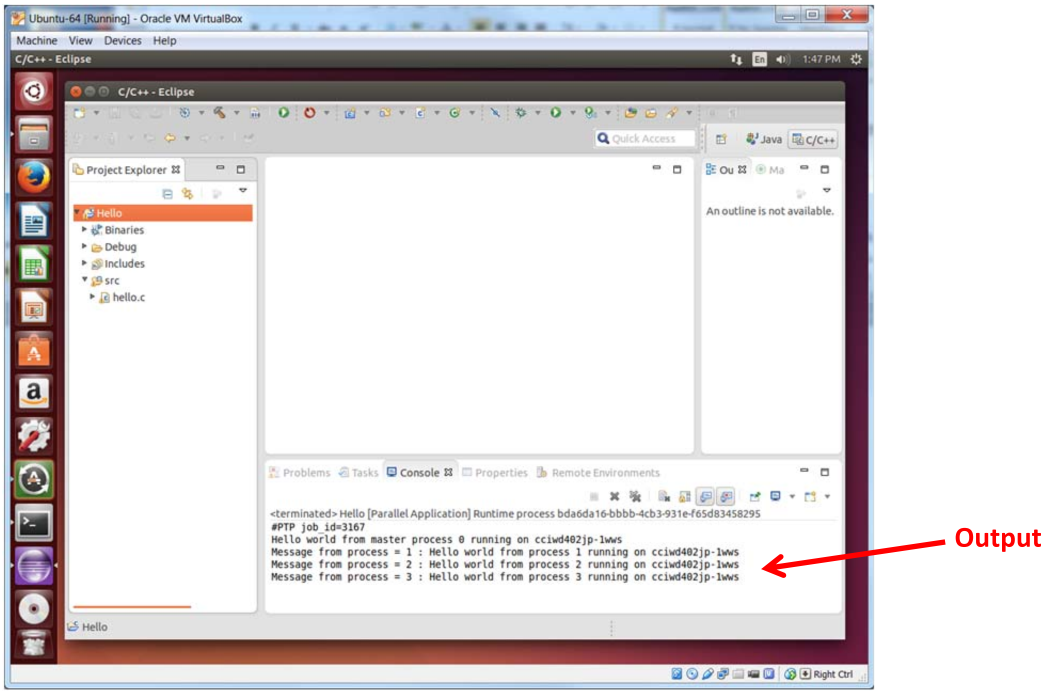
Application program: /home/abw/Workshop/MPI/workspace/Hello/Debug/Hello

☐ Copy executable from local filesystem

Path to local executable:

☑ Display output from all processes in a console view

Click **RUN**. (If **RUN** is grayed out, there are build or compile errors that prevent execution.)



```
<terminated> Hello [Parallel Application] Runtime process bda6da16-bbbb-4cb3-931e-f65d83458295
#PTP job_id=3167
Hello world from master process 0 running on cciwd402jp-1wws
Message from process = 1 : Hello world from process 1 running on cciwd402jp-1wws
Message from process = 2 : Hello world from process 2 running on cciwd402jp-1wws
Message from process = 3 : Hello world from process 3 running on cciwd402jp-1wws
```

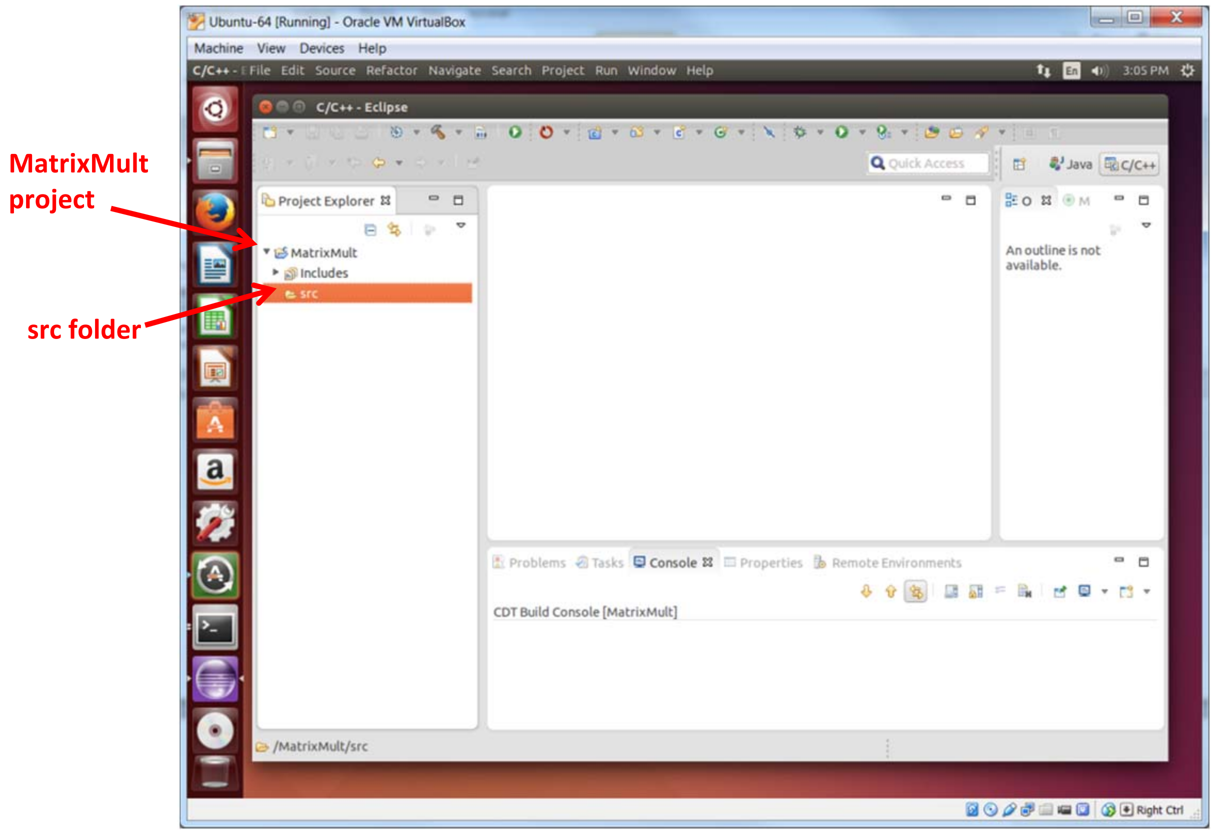
Include in your submission document for Task 1:

A screenshot or screenshots showing executing the hello world program with its output through Eclipse

Task 2 Creating and executing Matrix Multiplication program through Eclipse

The process for the matrix multiplication code is similar. We will go through the steps, this time not copying the source file but referring to it in the original location, which generally is a better approach.

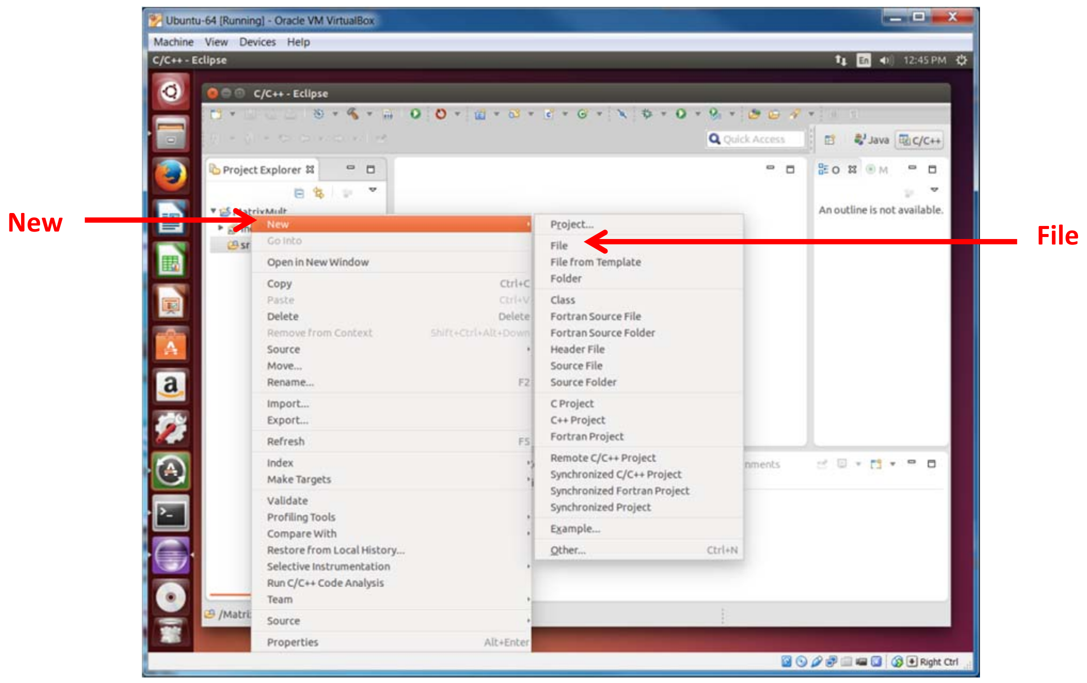
Create an MPI project called **MatrixMult** and a source directory called **src**:


MatrixMult project
src folder
Ubuntu-64 [Running] - Oracle VM VirtualBox
Machine View Devices Help
C/C++ - File Edit Source Refactor Navigate Search Project Run Window Help
3:05 PM
C/C++ - Eclipse
Quick Access
Java
C/C++
Project Explorer
MatrixMult
Includes
src
Problems Tasks Console Properties Remote Environments
/MatrixMult/src
Right Ctrl

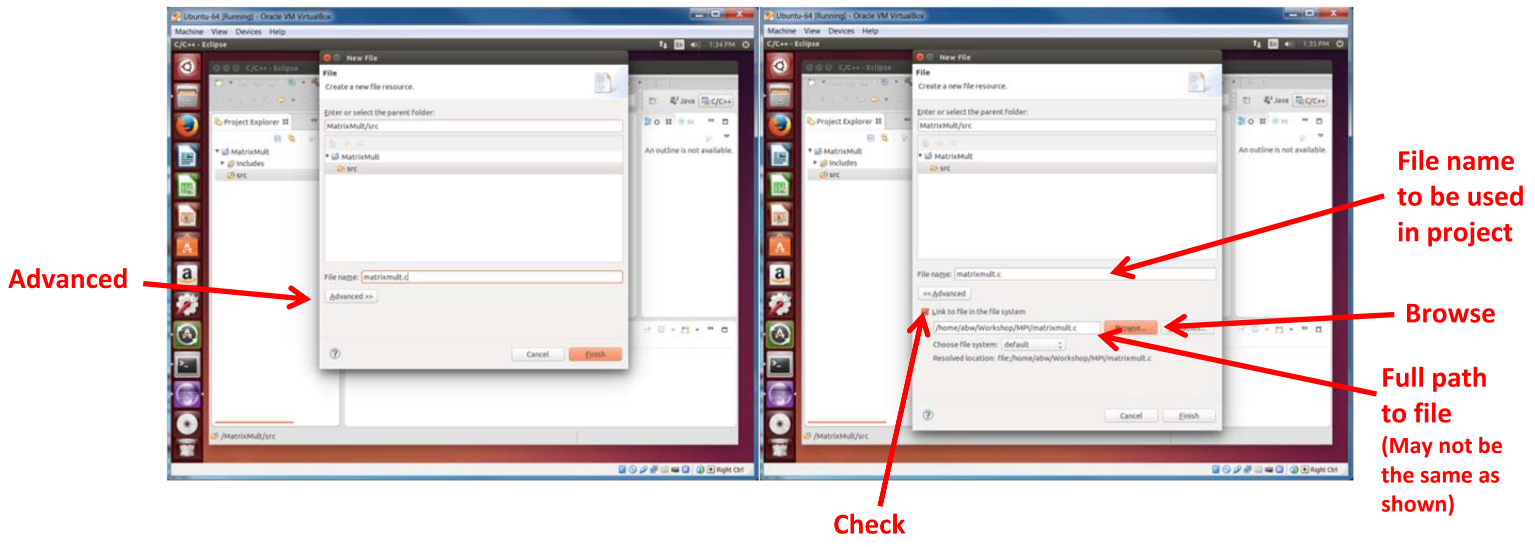
An outline is not available.

CDT Build Console [MatrixMult]

Now we will link source file **matrixmult.c** rather than copy it. Select the source folder **src**, right click, and select **NEW > FILE:**



Click on **Advanced>>** and enter the file name (**matrixmult.c**), check **"Link to file in the file system"** box and enter the full path to the source file (browse for file) and **Finish**:


Advanced
File name to be used in project
Browse
Full path to file
(May not be the same as shown)
Check
New File
Create a new file resource.
Advanced >>
<< Advanced
Link to file in the file system
Browse...
Cancel
Finish

Enter or select the parent folder: MatrixMult/src

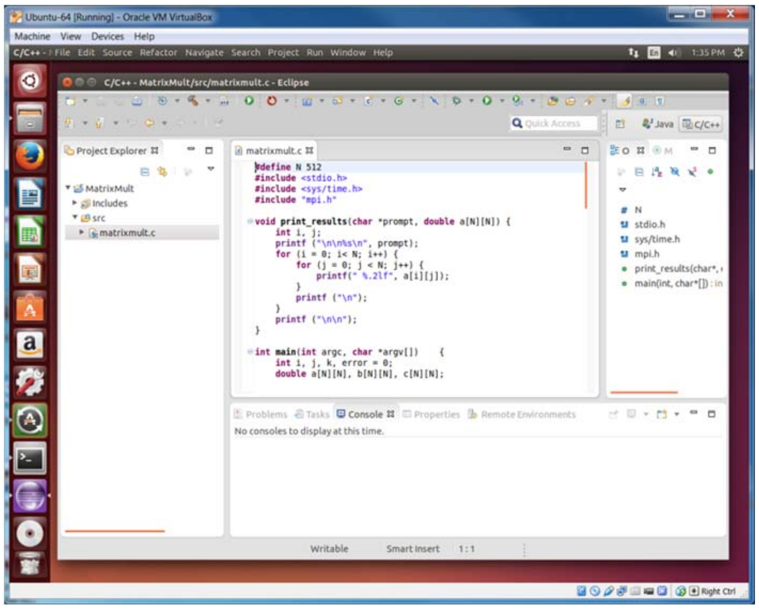
File name: matrixmult.c

/home/abw/Workshop/MPI/matrixmult.c

Choose file system: default

Resolved location: file:/home/abw/Workshop/MPI/matrixmult.c

You should now see the source file:


Ubuntu-64 [Running] - Oracle VM VirtualBox
Machine View Devices Help
C/C++ - MatrixMult/src/matrixmult.c - Eclipse
Project Explorer
MatrixMult
Includes
src
matrixmult.c
matrixmult.c
Problems Tasks Console Properties Remote Environments
Writable Smart Insert 1 : 1

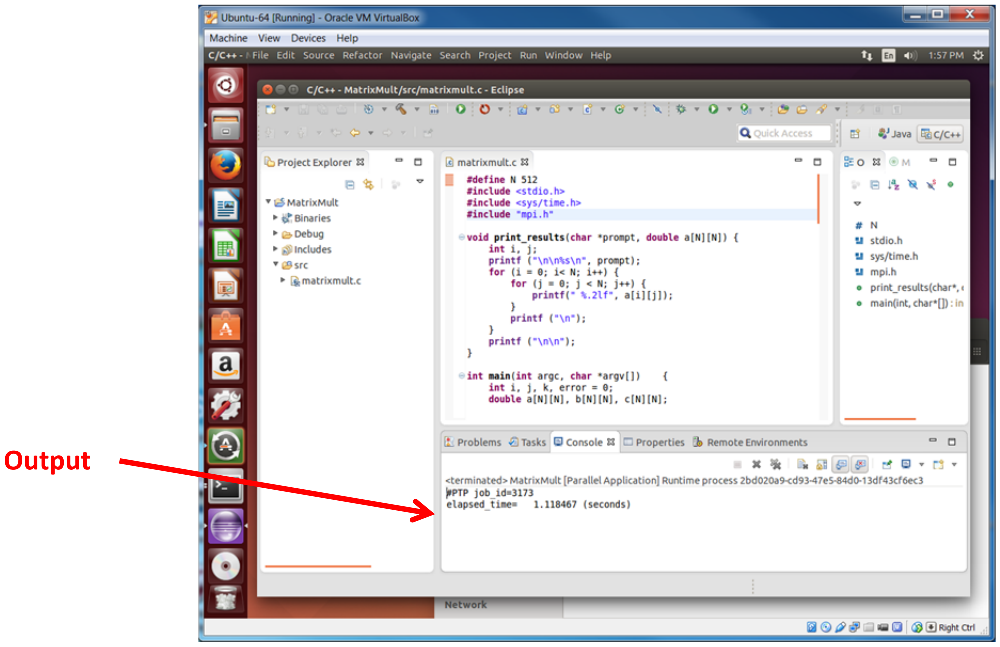
```c
#define N 512
#include <stdio.h>
#include <sys/time.h>
#include "mpi.h"

void print_results(char *prompt, double a[N][N]) {
    int i, j;
    printf ("\n\n%s\n", prompt);
    for (i = 0; i< N; i++) {
        for (j = 0; j < N; j++) {
            printf(" %.2lf", a[i][j]);
        }
        printf ("\n");
    }
    printf ("\n\n");
}

int main(int argc, char *argv[])    {
    int i, j, k, error = 0;
    double a[N][N], b[N][N], c[N][N];
```

```
No consoles to display at this time.
```

Run Configuration. Now create a run configurations (**MatrixMult**). As before, in the *Resource* tab, select the Target Type as "**Open-MPI-Generic-Interactive**", the connection type as "**Local".** Set the number of processes. Then run the project:


Output
Ubuntu-64 [Running] - Oracle VM VirtualBox
Machine View Devices Help
C/C++ - MatrixMult/src/matrixmult.c - Eclipse
Project Explorer
MatrixMult
Binaries
Debug
Includes
src
matrixmult.c
matrixmult.c
Problems Tasks Console Properties Remote Environments
Network

```c
#define N 512
#include <stdio.h>
#include <sys/time.h>
#include "mpi.h"

void print_results(char *prompt, double a[N][N]) {
    int i, j;
    printf ("\n\n%s\n", prompt);
    for (i = 0; i< N; i++) {
        for (j = 0; j < N; j++) {
            printf(" %.2lf", a[i][j]);
        }
        printf ("\n");
    }
    printf ("\n\n");
}

int main(int argc, char *argv[])    {
    int i, j, k, error = 0;
    double a[N][N], b[N][N], c[N][N];
```

```
<terminated> MatrixMult [Parallel Application] Runtime process 2bd020a9-cd93-47e5-84d0-13df43cf6ec3
#PTP job_id=3173
elapsed_time=   1.118467 (seconds)
```

Include in your submission document for Task 2:

1. A screenshot or screenshots showing executing matrix multiplication program with its output through Eclipse.
2. Conclusions on using Eclipse compare to using the command line. Compare and contrast the two ways. (3-10 sentences). (Important 5 points)

Part 3 Execution matrix multiplication program on a remote cluster (30%)

***First very carefully* review the notes "Using UNC-C ccigridgw.uncc.edu cluster' in Additional Information. Watch for announcements on Moodle about the availability of the cluster.**

Task 1 Changing the source and destination locations

Before we can execute the matrix multiplication program on the cluster, we have to correct the coding. Although the matrix multiplication code as given will work with OpenMPI installed on the VM, it will not work with MPICH as installed on the UNC-Charlotte cluster because the same buffer is used the source and destination in the scatter and gather routines, i.e.:

```
MPI_Scatter(a, blksz*N, MPI_DOUBLE, a, blksz*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Gather(c, blksz*N, MPI_DOUBLE, c, blksz*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

The MPI standard seems to be quiet about this matter. Change the coding so as to use different buffers, and re-test the code on your local machine. Verify correct results are produced on your computer.

Task 2 Executing on cci-gridgw.uncc.edu

Connect to the remote cluster **cci-gridgw.uncc.edu** and make a directory in your home directory called **MPI** that will be used for the MPI programs, and **cd** into that directory. Transfer the MPI source program **matrixmult.c** to that directory. Compile **matrixmult.c** on the remote cluster front node **cci-gridgw.uncc.edu** with **mpicc**:

```
mpicc -o matrixmult matrixmult.c
```

Execute **matrixmult** with the Hydra process manager (**IMPORTANT: NOT mpiexec**):

```
mpiexec.hydra -n numprocesses ./matrixmult
```

with 32 processes. With this command, the program will run just on **cci-gridgw.uncc.edu**.

Task 3 Executing on four nodes, cci-gridgw.uncc.edu, cci-grid05, cci-grid07, and cci-grid08.

To execute the MPI program using multiple computers, a file needs to be created that specifies the computers**.** Create a file called ***machines*** containing the list of machines, using the local names of four nodes:[4]

```
cci-grid05
```

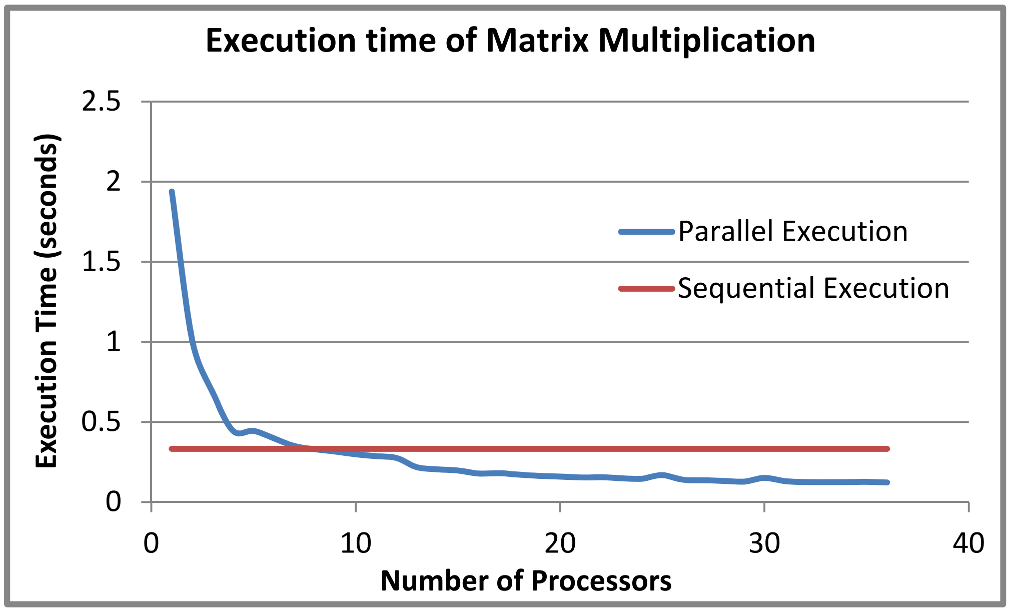
[4] In the event some nodes are not accessible, use just those nodes available.

```
cci-grid07
cci-grid08
cci-gridgw.uncc.edu
```

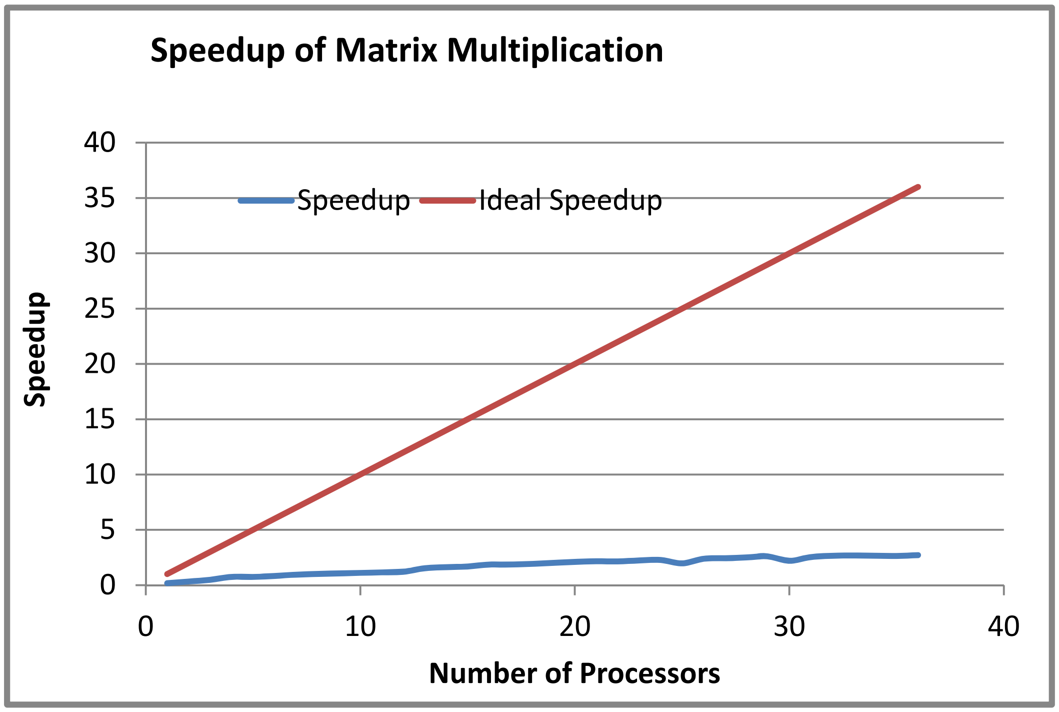
To execute the **matrixmult** program on the computers specified in the machines file with 4 processes, the command is:

```
mpiexec.hydra -machinefile machines -n 4 ./matrixmult
```

Record the elapsed times for the program running on the different number of processes. Create a graph of these results using a graphing program, such as a spreadsheet. Give a graph of execution times and a graph of speed-up. Compare with linear speedup. The graphs should look something like the figures below and overleaf but the shape of the curves will not. Your curves may show something entirely different. The figures are just examples. Make sure that you provide axes labels, a legend and a title to the graphs. Include copies of the graphs in your submission document.



Example Execution Time Graph



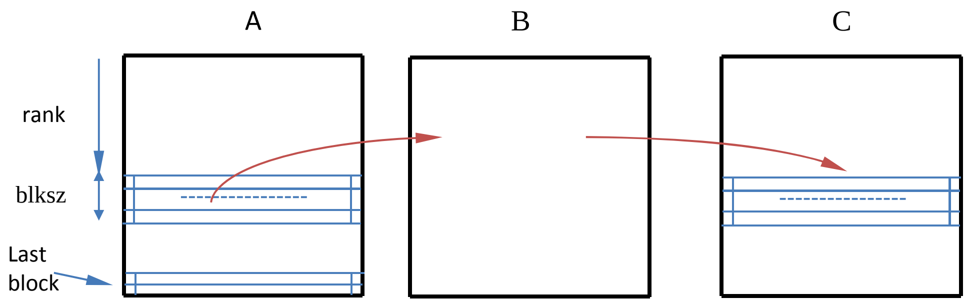
Example Speedup Graph

Include in your submission document:

1. Matrix multiplication program code modified to execute on the cluster
2. Screenshot(s) showing it execute on the cluster. Show the output is correct.
3. A copy of your execution time and speedup graphs.
4. Conclusions

Part 4 Changing the matrix multiplication program to handle any value of N. (10%)

The matrix multiplication code, as written, only works if ***P*** divides evenly into N. Modify the program to handle any value of ***N***. The final block will be less that than other blocks as illustrated below:



The stored matrices and messages should not be any larger than necessary, i.e. simply padding out arrays with zeros is not acceptable. The easiest solution is to treat the last block separately by the master process after all the processes including the master have handled the other blocks. This solution is acceptable although more parallelism is achieved if the process that handles the last block is not also used for one of the other blocks.

The code is to have the same features as the previous matrix multiplication code including comparing the parallel result with sequential result and displaying ***N*** and ***P***. Also display the size of **blksz** and the size of the last block.

Include in your submission document for Part 4:

1. Matrix multiplication program code that handles any value of ***N***

2. A full explanation of your code modifications

3. Two screenshots showing the program executing on your computer with the output correct, one with ***N*** = 250 and ***P*** = 4, and one with ***N*** = 257 and ***P*** = 9.

Assignment Report Preparation and Grading

Produce a report that shows that you successfully followed the instructions and performs all tasks by taking screenshots and include these screenshots in the document.

Every task and subtask specified will be allocated a score so make sure you clearly identify each part/task you did. Make sure you include everything that is specified in the "What to include ..." section at the end of each task/part. **Include all code, not as screen shots of the listings but complete properly documented code listing.** The programs are often too long to see in a single screenshot and also not easy to read if small. Using multiple screenshots is not acceptable for code. Copy/paste the code into the Word document or whatever you are using to create the report.

Assignment Submission

Convert your report into a ***single pdf*** document and submit the pdf file on Moodle by the due date as described on the course home page. It is extremely important you submit only a single pdf file. ***You will lose 10 points if you submit in any other format (e.g. Word, OpenOffice, ...). zip with multiple files is especially irritating.***