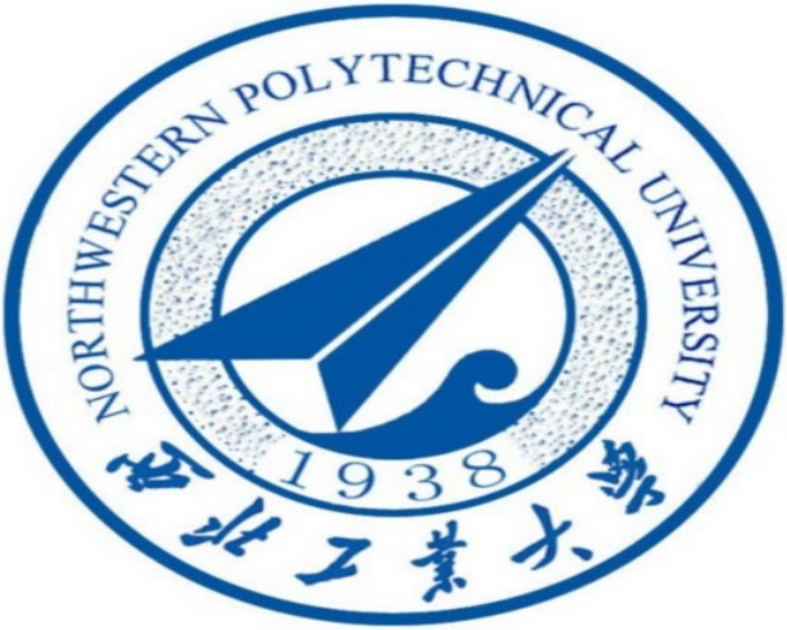


Parallel Programming

Homework 2
Process Topologies

Student Name : ABID ALI
Student ID : 2019380141
Student Email : abiduu354@gmail.com / **3616990795@qq.com**
Lecturer : Professor Zhengxiong Hou
Submission : 12/30/2022
Submitted Email : houzhx@nwpu.edu.cn

Contents

Assignment 2: Report on advanced technology using MPI 4.0

Please choose one of the following topics and give a report about the topic. It will be better if there are some experiments.

1. Nonblocking Collective Operations

2. Dynamic Process Model

3. Process Topologies

1. Introduction

1.1 MPI

MPI is a standard for a message-passing library interface. MPI primarily addresses the message-passing parallel programming model, in which data is transported from one process's address space to another process's address space via cooperative operations on each process. In collective operations, remote-memory access operations, dynamic process generation, and parallel I/O, extensions to the "traditional" message-passing architecture are offered. Simply said, the purpose of the Message-Passing Interface is to create a generally accepted standard for creating message-passing applications. The portability and convenience are the primary benefited of establishing a message-passing standard. Therefore, the interface has to provide a convenient, practical, efficient and adaptable standard for message passing. The assignment topic that I chose – topology process, is included in the standard of MPI.

1.2 MPI 4.0.

MPI 4.0. standardization endeavors point at including modern procedures, approaches or concepts to the MPI standard that will offer assistance MPI address the require of current and next generation applications and architecture. The biggest change is an alternative way to expand large-count versions of many routines to remove the restrictions when using int or INTEGER for the count parameter, and initializing persistent aggregates, partitioned communication, MPIs. Application information assertions, and improved definition of error handling. There are also some minor improvements and fixes.

1.3 Using MPI with C

Parallel programs enable users to fully utilize the multi-node structure of supercomputing clusters. Message Passing Interface (MPI) is a standard used to allow several different processors on a cluster to communicate with each other. In this tutorial we will be using the Intel C++

Compiler, GCC, IntelMPI, and OpenMPI to create a multiprocessor 'hello world' program in C++. This tutorial assumes the user has experience in both the Linux terminal and C++.

1.4 Process Topology

Process Topology describes many utility functions that MPI 4.0. standardization endeavors point at including modern procedures, approaches or concepts to the MPI standard that will offer assistance MPI address the require of current and next generation applications and architecture.

The biggest change is an alternative way to expand large-count versions of many routines to remove the restrictions when using int or INTEGER for the count parameter, and initializing persistent aggregates, partitioned communication, MPIs. Application information assertions, and improved definition of error handling. MPI offers two types of topologies: Cartesian grid-based topologies and graph-based topologies. In a graph-based topology, nodes represent processes and edges connect processes that communicate with each other.

Many applications use grid-based topologies to create rings, 2D or multidimensional grids. The lattice structure is defined by some dimensions in each coordinate direction and some processes. The simplest of the two topologies is the lattice structure. The grid structure is mapped using the Rowmajor numbering system.

Topologies can be added to an intra-communicator, but not inter-communicators, as an optional attribute to provide a convenient process naming mechanism for communication between the processes in the group/communicator. This topology can map processes to physical hardware within the runtime system (this mapping is machine-independant).

If a hardware topology is known, it can be used in the virtual topology to optimize performance by assigning processes to physical hardware to increase speed in communications. If a hardware topology is not known, a virtual topology will still provide a convenient process naming structure for use by the runtime system and increase the readablity of the program.

1.5 Virtual Topologies

MPI provides us with two types of topologies: topologies based on a cartesian grid, and topologies based on graphs. Under the topology based upon a graph the nodes stand for processes and the edges connect processes that communicate with each other. Many applications use a grid-based topology that creates a ring, two or more dimensional grids, or tori. Grid structures are defined by a number of dimensions and number of processes in each coordinate direction. The easiest of the two topologies is the grid structure. Grid structures are mapped using a row-major numbering system. Below is an example of a 2 x 2 grid.

0 [0,0]	1 [0,1]
2 [1,0]	3 [1,1]

2. Procedure

2.1 Windows MPI setup

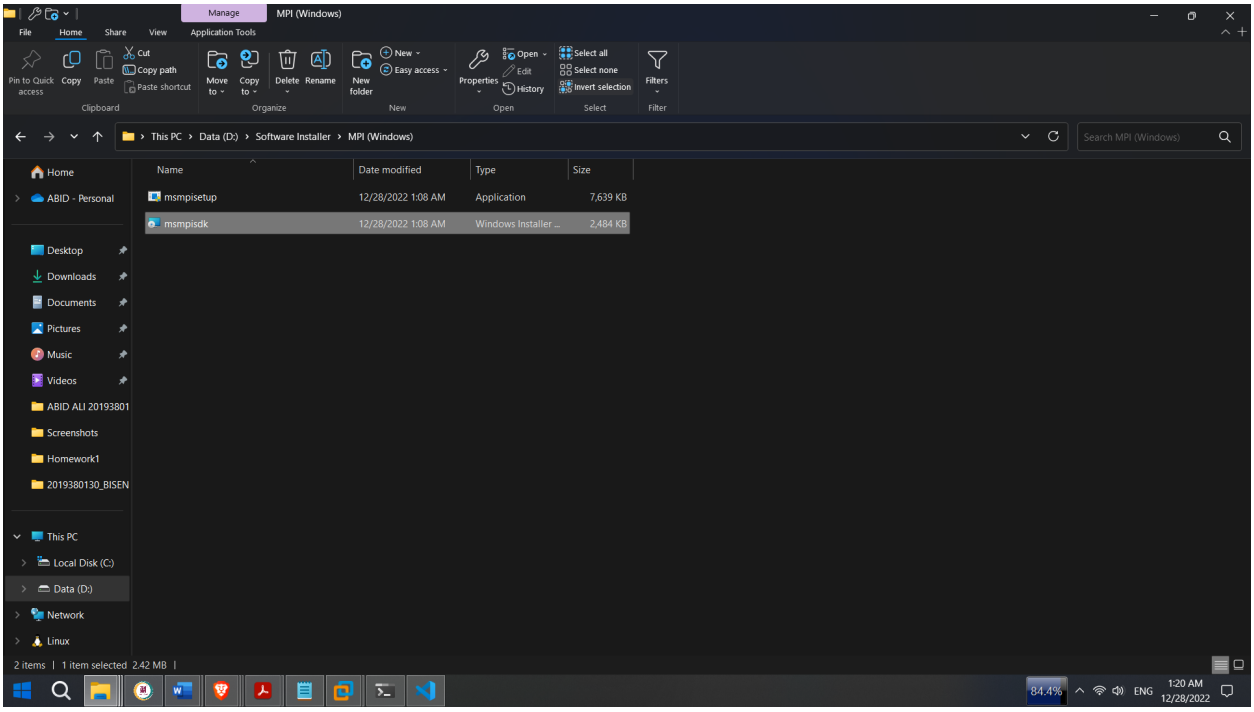


Figure: Windows MPI setup

2.2 Checking MPI in Windows

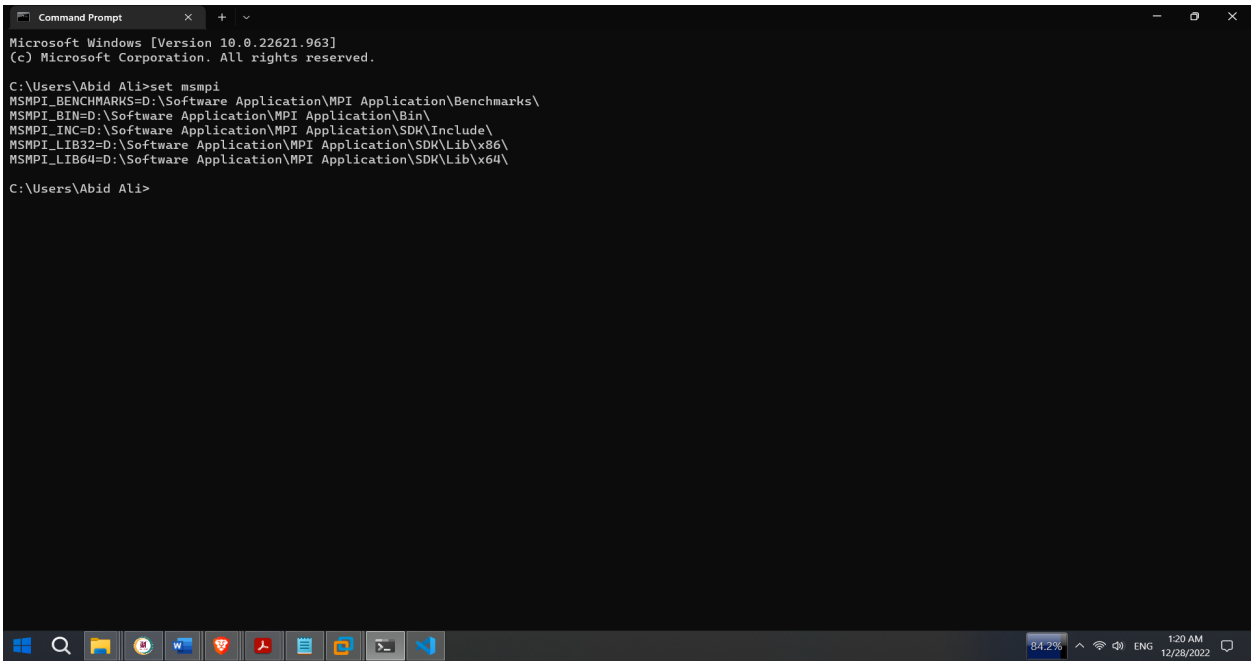


Figure: Running set msmpi in Command Prompt

2.3 Including ${MSMPI_INC} in One include path per line

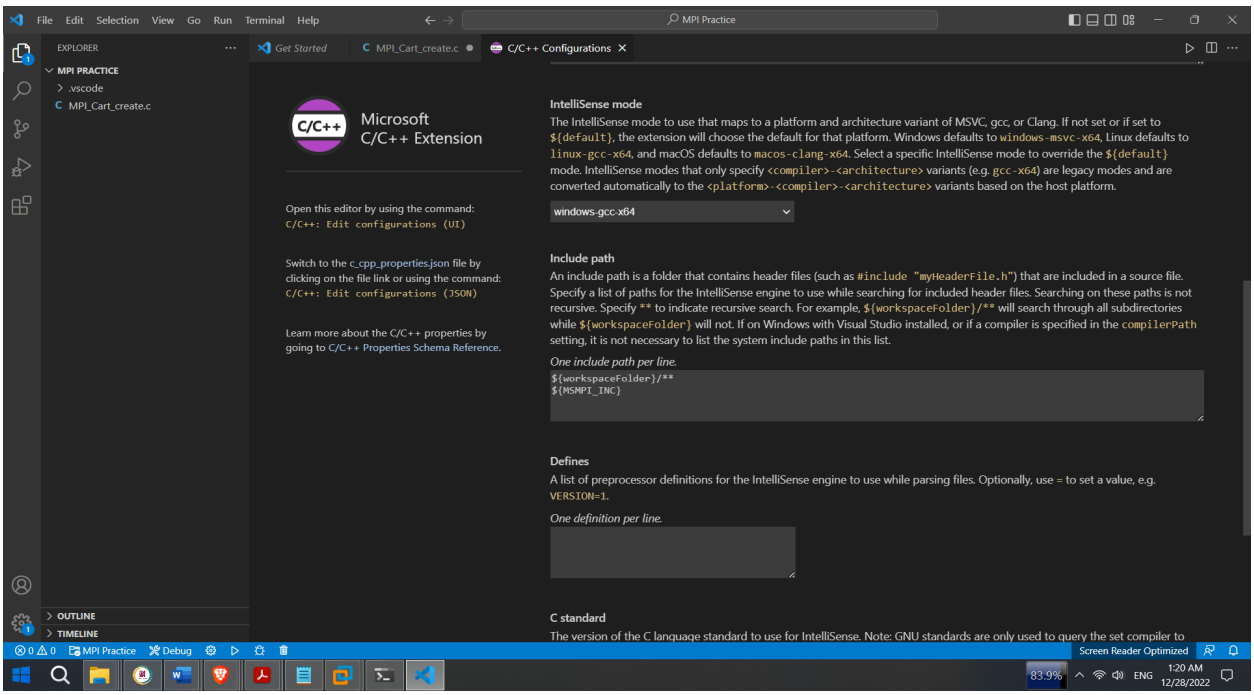


Figure: One include path per line for mpi.h header

2.4 Configuring Default Build Task

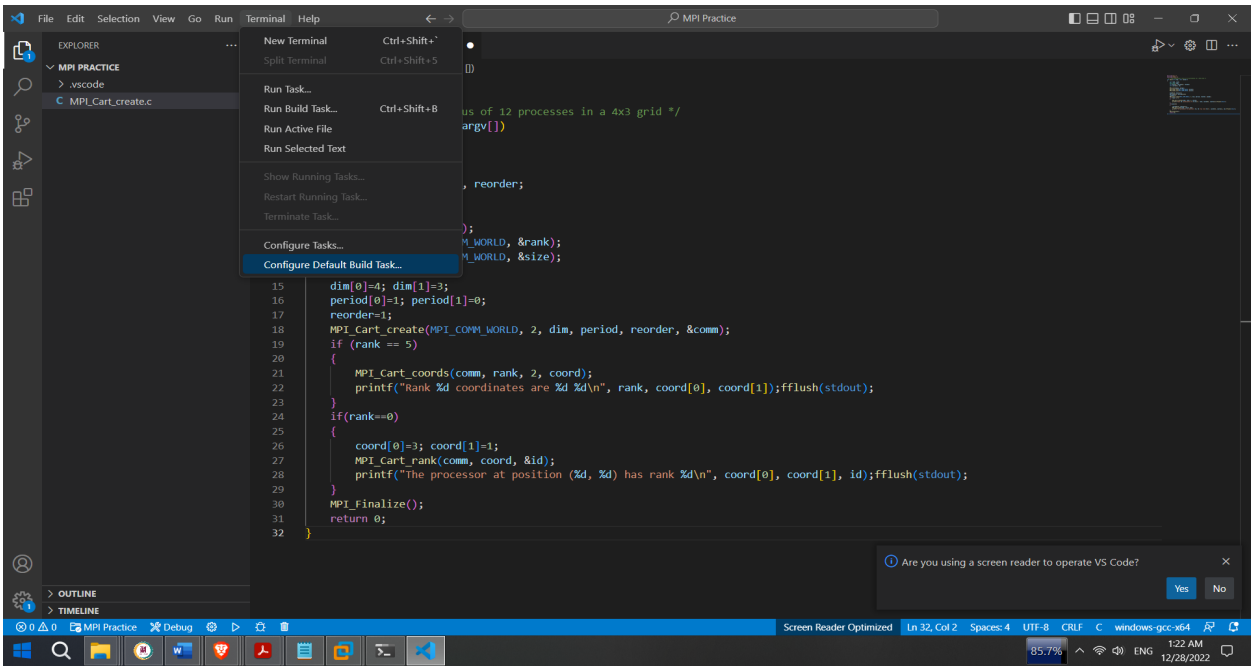


Figure: Building Default Task

2.5 Selecting the Compiler

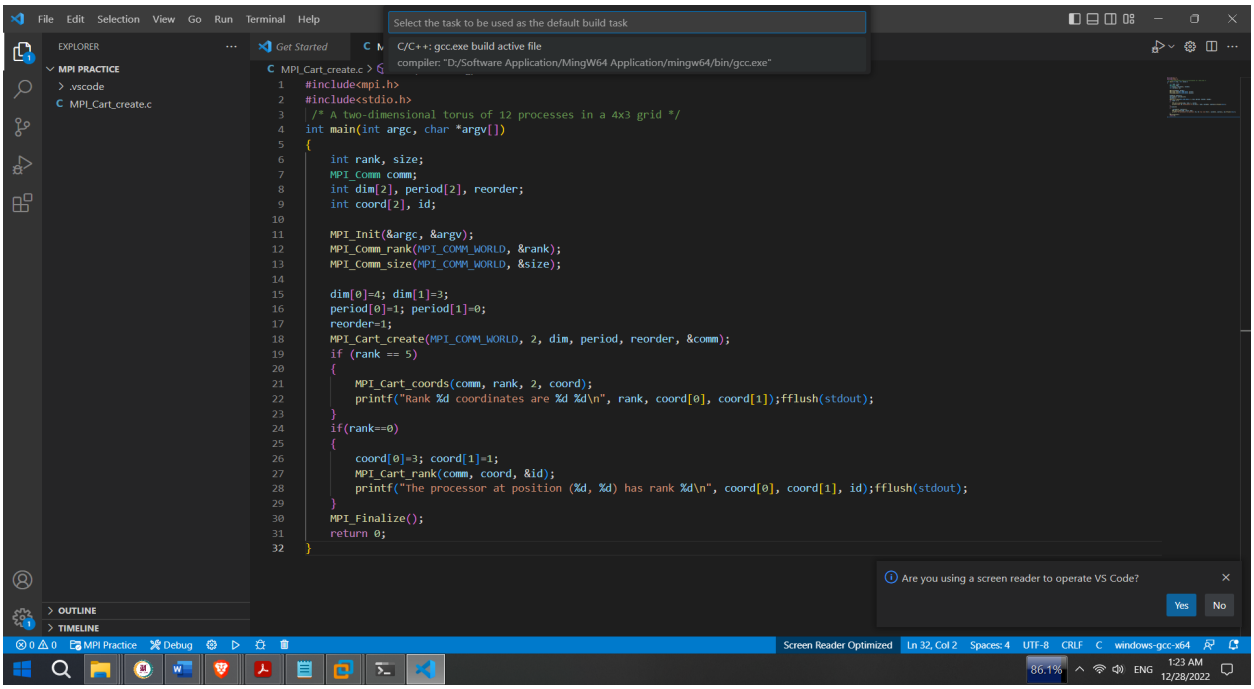


Figure: Choose the C / C++ Compiler

2.6 Writing Extra Command in Json File

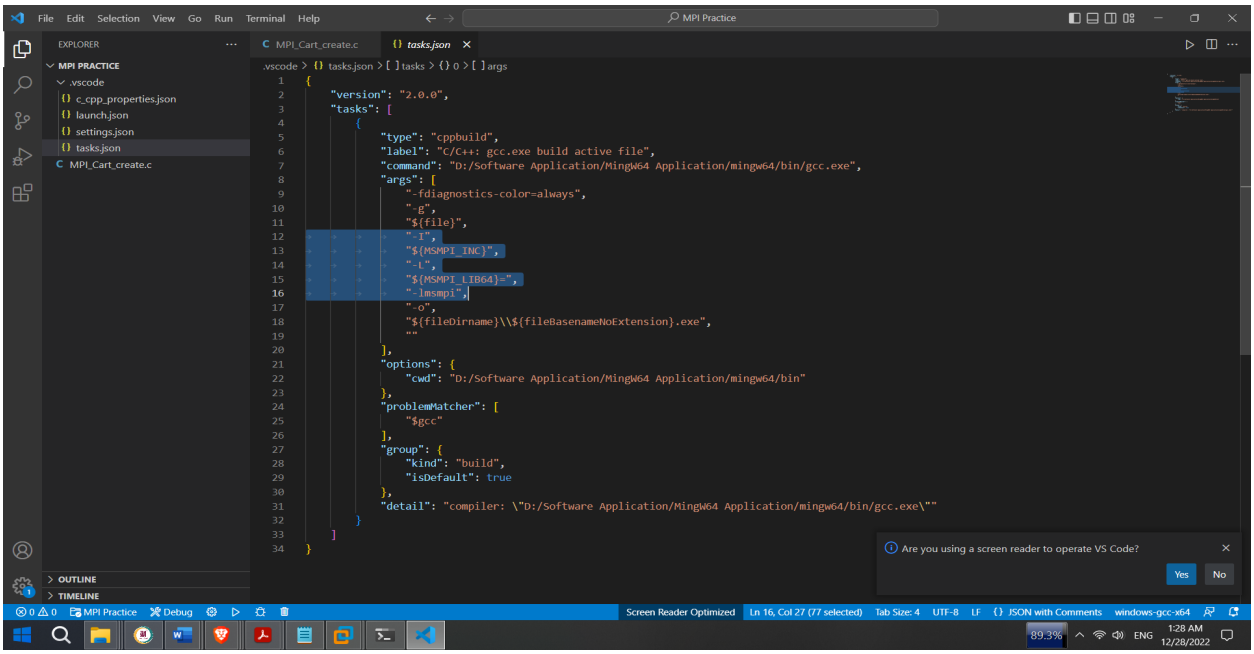


Figure: Writing in Json File

2.7 Running Build Task

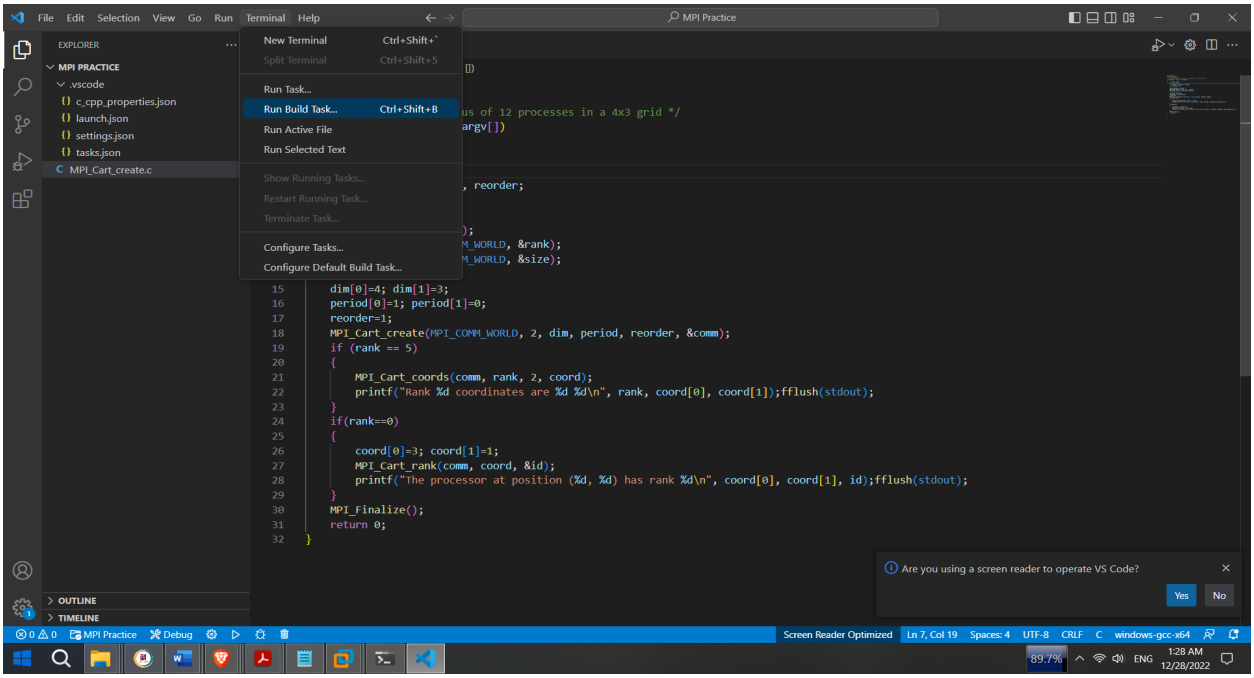


Figure: Running Build Task

2.8 Result

MPI_Cart_create 12 processes in a 4x3 grid in Windows

```
Compiling
Compilation is OK
Execution ...
The processor at position (3, 1) has rank 10
Rank 5 coordinates are 1 2
Done.
```

Figure: MPI_Cart_create 12 processes in a 4x3 grid in Windows

MPI_Graph_create 4 processes in Windows

```
Compiling
Compilation is OK
Execution ...
Rank: 0, recv[0] = 3, recv[1] = 2, recv[2] = 1, recv[3] = -1
Rank: 1, recv[0] = 0, recv[1] = 3, recv[2] = 2, recv[3] = -1
Rank: 2, recv[0] = 3, recv[1] = -1, recv[2] = -1, recv[3] = -1
Rank: 3, recv[0] = 0, recv[1] = 2, recv[2] = -1, recv[3] = -1
Done.
```

Figure: MPI_Graph_create 4 processes in Windows

VMware Linux Ubuntu Total Processor Cores 8

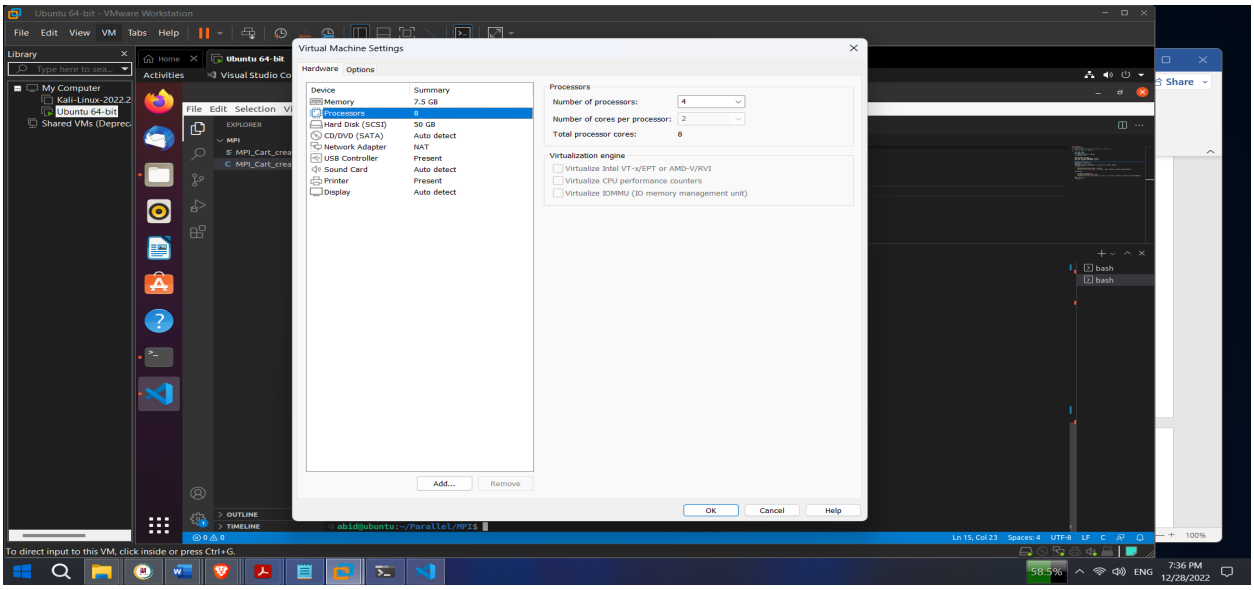


Figure: VMware Linux Ubuntu Total Processor Cores 8

Note:

We can use only 8 processor in VMware Workstation due to limitation but we can apply more than 8 processor in windows due to my computer

MPI_Cart_create 12 processes in a 4x3 grid in Linux(Ubuntu)

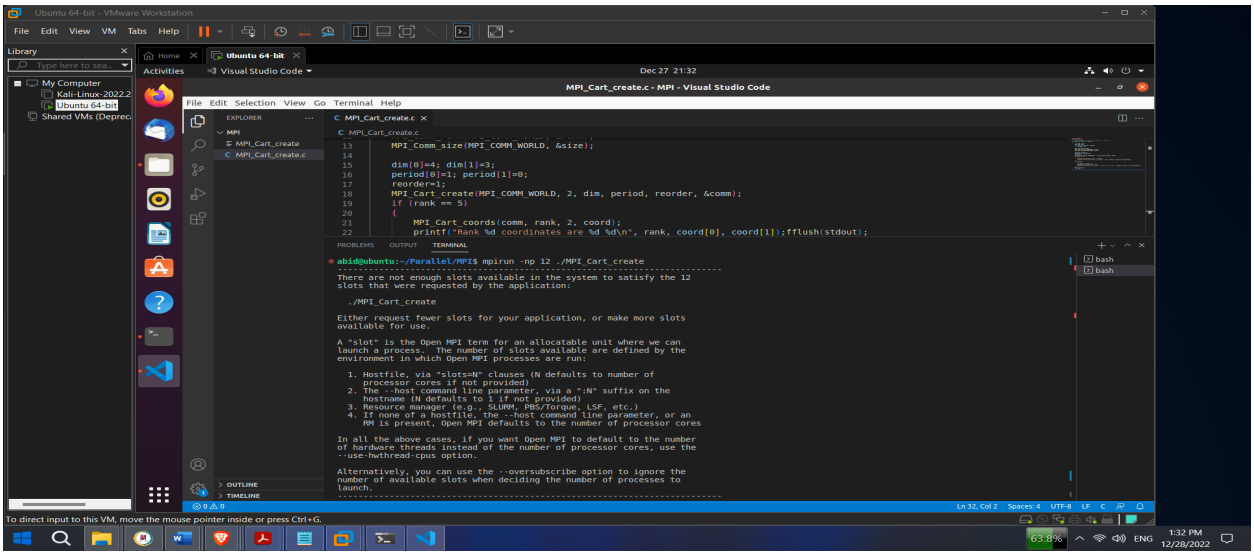


Figure: MPI_Cart_create 12 processes in a 4x3 grid in Linux(Ubuntu)

Change

dim[0]=4; dim[1]=3;

Compilation

mpicc MPI_Cart_create.c -o MPI_Cart_create

mpirun -np 12 ./MPI_Cart_create

MPI_Cart_create 9 processes in a 3x3 grid in Linux(Ubuntu)

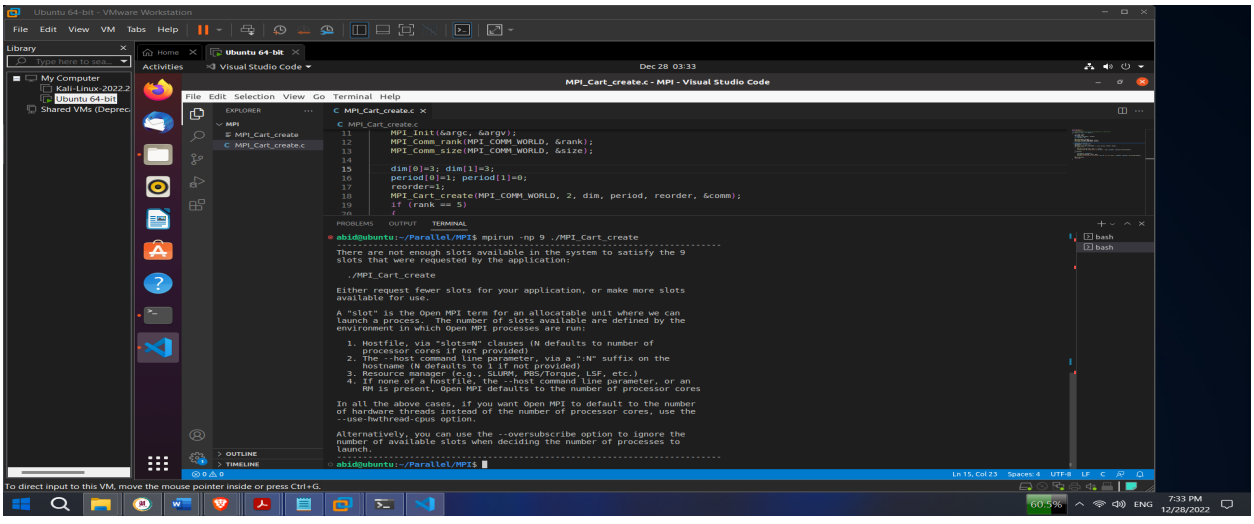


Figure: MPI_Cart_create 9 processes in a 3x3 grid in Linux(Ubuntu)

Change

dim[0]=3; dim[1]=3;

Compilation

mpicc MPI_Cart_create.c -o MPI_Cart_create

mpirun -np 9 ./MPI_Cart_create

MPI_Cart_create 6 processes in a 3x2 grid in Linux(Ubuntu)

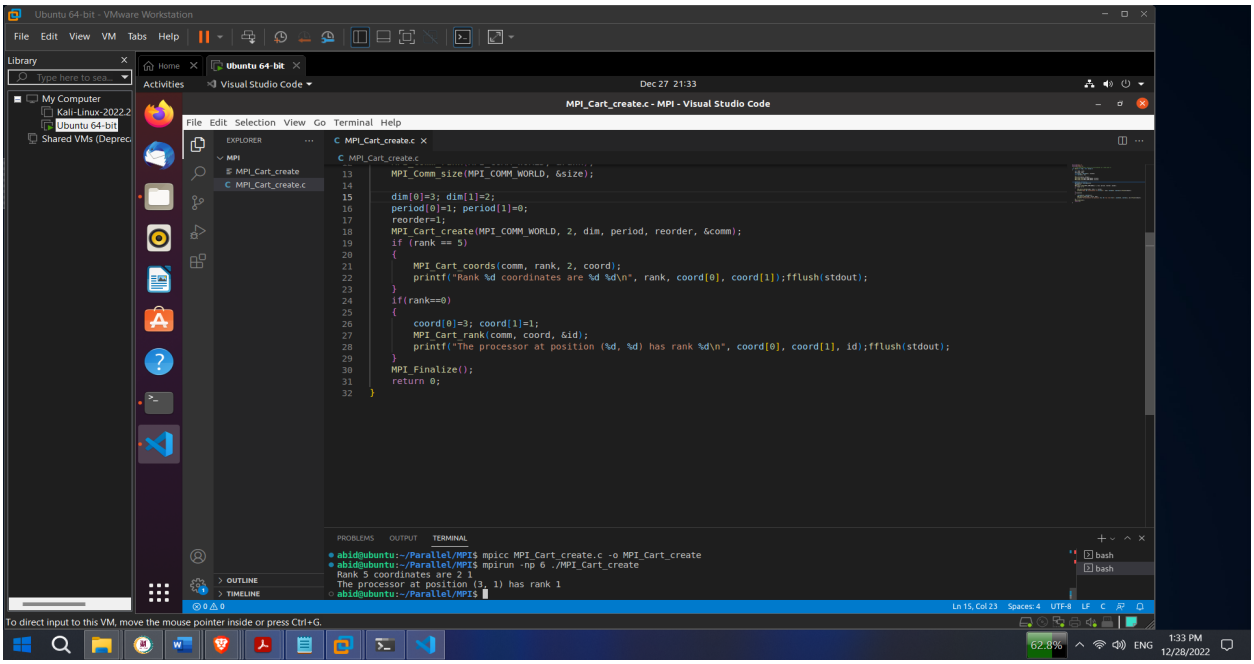


Figure: MPI_Cart_create 6 processes in a 3x2 grid in Linux(Ubuntu)

Change

dim[0]=3; dim[1]=3;

Compilation

mpicc MPI_Cart_create.c -o MPI_Cart_create

mpirun -np 6 ./MPI_Cart_create

The 6 represent the number of processes assigned to solve the problem. So, we can say 6 processes assigned the program.

Errors Compiling MPI in Windows

```
*  Executing task: C/C++: gcc.exe build active file

Starting build...
"D:/Software Application/MingW64 Application/mingw64/bin/gcc.exe" -fdiagnostics-color=always -g "D:\4th year,1st Semester\Parallel Programing\MPI Practice\MPI_Cart_c
reate.c" -I "D:\Software Application\MPI Application\SDK\Include\" -L "D:\Software Application\MPI Application\SDK\Lib\x64\=" -lmsmpi -o "D:\4th year,1st Semester\Pa
rallel Programing\MPI Practice\MPI_Cart_create.exe"
gcc.exe: error: Application\MPI: No such file or directory
gcc.exe: error: Application\SDK\Lib\x64\= -lmsmpi -o D:\4th: Invalid argument
gcc.exe: error: year,1st: No such file or directory
gcc.exe: error: Semester\Parallel: No such file or directory
gcc.exe: error: Programing\MPI: No such file or directory
gcc.exe: error: Practice\MPI_Cart_create.exe : No such file or directory

Build finished with error(s).

*  The terminal process terminated with exit code: -1.
*  Terminal will be reused by tasks, press any key to close it.
```

Figure: Errors Compiling MPI in Windows

Note:

We need to remove all the space from the name of folder not to get this error and after removing the spaces from folder then the program didn't have any error.

Including path in linux

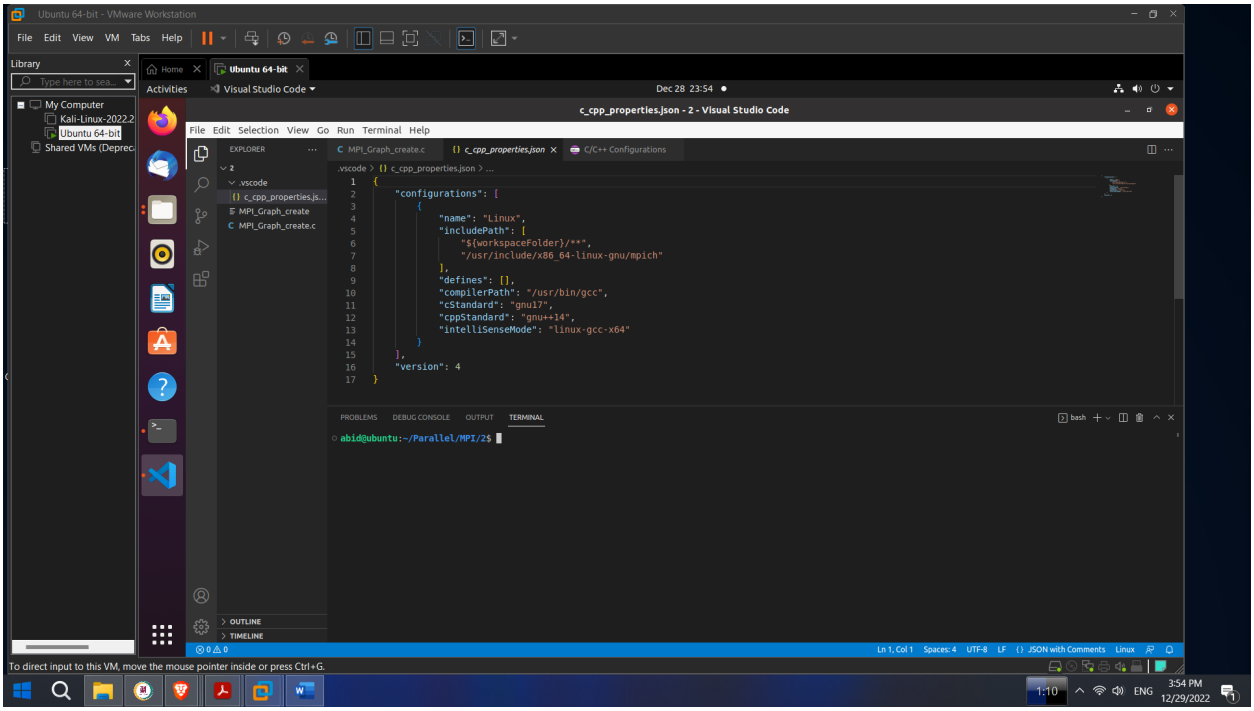


Figure: Including path in linux

MPI_Graph_create 4 processes in Linux(Ubuntu)

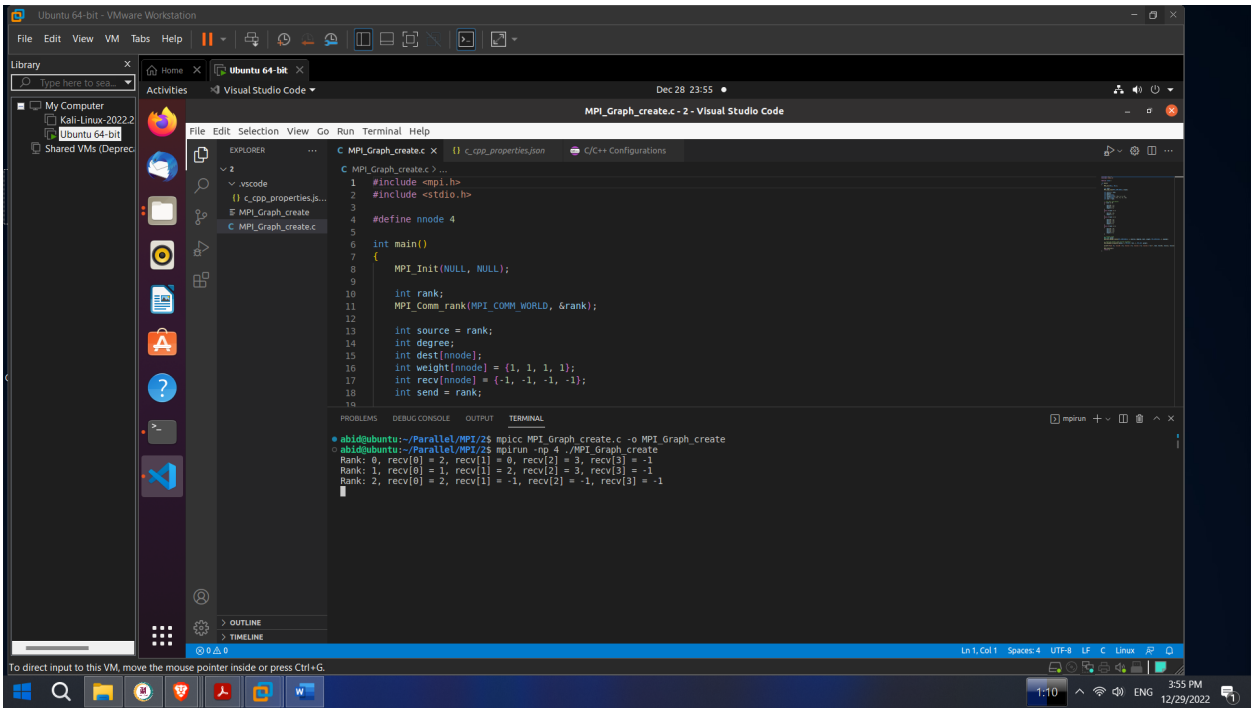


Figure: MPI_Graph_create 4 processes in Linux(Ubuntu)

Compilation

mpicc MPI_Graph_create.c -o MPI_Graph_create

mpirun -np 4 ./MPI_Graph_create

3. Topology Constructors

3.1 Cartesian Constructor

int MPI_Cart_create (MPI_Comm comm_old, int ndims, int *dims, int *periods, int reorder, MPI_Comm *comm_cart)

Parameter	Meaning of Parameter
comm_old	input communicator (handle)
Ndims	number of dimensions of cartesian grid (integer)
Dims	integer array of size ndims specifying the number of processes in each

	dimension
Periods	logical array of size ndims specifying whether the grid is periodic (true) or not (false) in each dimension
Reorder	ranking may be reordered (true) or not (false) (logical)
comm_cart	communicator with new cartesian topology (handle)

MPI_Cart_create makes a new communicator to which topology information has been attached. Periodic means that if the neighbor of a ranking process is out of range, it will wrap to the beginning if true or will be MPI_PROC_NULL if false.

3.2 Cartesian Convenience - MPI_Dims_create

int MPI_Dims_create(int nnodes, int ndims, int *dims)

Parameter	**Meaning of Parameter**
Nnodes	number of nodes in a grid (integer)
Ndims	number of cartesian dimensions (integer)
Dims	integer array of size ndims specifying the number of nodes in each dimension

MPI_Dims_create creates a division of processors in a cartesian grid.

3.3 Graph Constructor

int MPI_Graph_create (MPI_Comm comm_old, int nnodes, int *index, int *edges, int reorder, MPI_Comm *comm_graph)

Parameter	**Meaning of Parameter**
comm_old	input communicator without topology (handle)
Nnodes	number of nodes in graph (integer)
Index	array of integers describing node degrees
Edges	array of integers describing graph edges
Reorder	ranking may be reordered (true) or not (false) (logical)
comm_graph	communicator with graph topology added (handle)

4. Topology Inquiry Calls

4.1 MPI_Topo_test

int MPI_Topo_test (MPI_Comm comm, int *top_type)

Parameter	Meaning of Parameter
Comm	communicator (handle)
top_type	topology type of communicator comm (choice)

MPI_Topo_test determines the type of topology (if any) associated with a communicator. On success, top_type will be either MPI_CART, MPI_GRAPH, or MPI_UNDEFINED.

4.2 MPI_Graphdims_get

int MPI_Graphdims_get (MPI_Comm comm, int *nnodes, int *nedges)

Parameter	Meaning of Parameter
Comm	communicator for group with graph structure (handle)
Nnodes	number of nodes in graph (integer)
Nedges	number of edges in graph (integer)

MPI_Graphdims_get retrieves graph topology information associated with a communicator

4.3 MPI_Graph_get

int MPI_Graph_get (MPI_Comm comm, int maxindex, int maxedges, int *index, int *edges)

Parameter	Meaning of Parameter
comm	communicator with graph structure (handle)
maxindex	length of vector index in the calling program (integer)
maxedges	length of vector edges in the calling program (integer)
index	array of integers containing the graph structure (for details see the definition of MPI_GRAPH_CREATE)
edges	array of integers containing the graph structure

MPI_Graph_get retrieves graph topology information associated with a communicator.

4.4 MPI_Cartdim_get

int MPI_Cartdim_get (MPI_Comm comm, int *ndims)

Parameter	Meaning of Parameter
Comm	communicator with cartesian structure (handle)
Ndims	number of dimensions of the cartesian structure (integer)

MPI_Cartdim_get retrieves Cartesian topology information associated with a communicator.

4.5 MPI_Cart_get

int MPI_Cart_get (MPI_Comm comm, int maxdims, int *dims, int *periods, int *coords)

Parameter	Meaning of Parameter
Comm	communicator with cartesian structure (handle)
Maxdims	length of vectors dims , periods , and coords in the calling program (integer)
Dims	number of processes for each cartesian dimension (array of integer)
Periods	periodicity (true/false) for each cartesian dimension (array of logical)
Coords	coordinates of calling process in cartesian structure (array of integer)

MPI_Cart_get retrieves Cartesian topology information associated with a communicator.

4.6 MPI_Cart_rank

int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank)

Parameter	Meaning of Parameter
Comm	communicator with cartesian structure (handle)
Cords	integer array (of size ndims) specifying the cartesian coordinates of a process
Rank	rank of specified process (integer)

MPI_Cart_rank determines process rank in communicator given Cartesian location.

4.7 MPI_Cart_coords

int MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims, int *coords)

Parameter	Meaning of Parameter
comm	communicator with cartesian structure (handle)
Rank	rank of a process within group of comm (integer)
maxdims	length of vector coords in the calling program (integer)
coords	integer array (of size ndims) containing the Cartesian coordinates of specified process (integer)

MPI_Cart_coords determines process coords in cartesian topology given rank in group.

4.8 MPI_Graph_neighbors_count

int MPI_Graph_neighbors_count (MPI_Comm comm, int rank, int *nneighbors)

Parameter	Meaning of Parameter
Comm	communicator with graph topology (handle)
Rank	rank of process in group of comm (integer)
Nneighbors	number of neighbors of specified process (integer)

MPI_Graph_neighbors_count returns the number of neighbors of a node associated with a graph topology.

4.9 MPI_Graph_neighbors

int MPI_Graph_neighbors (MPI_Comm comm, int rank, int maxneighbors, int *neighbors)

Parameter	Meaning of Parameter
Comm	communicator with graph topology (handle)
Rank	rank of process in group of comm (integer)
maxneighbors	size of array neighbors (integer)

Neighbors	ranks of processes that are neighbors to specified process (array of integer)

MPI_Graph_neighbors return the neighbors of a node associated with a graph topology.

5. Process

5.1 MPI topology

MPI topology mechanism is an extra, optional attribute that one can give to an intra-communicator; topologies cannot be added to inter-communicators. A topology can provide a convenient naming mechanism for the processes of a group (within a communicator), and additionally, *may assist the runtime system in mapping the processes onto hardware*.

As stated for Groups, Communicators, a process group in MPI is a collection of n processes. Each process in the group is assigned a rank between 0 and n-1. In many parallel applications a linear ranking of processes does not adequately reflect the logical communication pattern of the processes (which is usually determined by the underlying problem geometry and the numerical algorithm used).

Often the processes are arranged in topological patterns such as two- or three-dimensional grids. More generally, the logical process arrangement is described by a graph.

A clear distinction must be made between the virtual process topology and the topology of the underlying, physical hardware. The virtual topology can be exploited by the system in the assignment of processes to physical processors, if this helps to improve the communication performance on a given machine. How this mapping is done, however, is outside the scope of MPI. The description of the virtual topology, on the other hand, depends only on the application, and is machine-independent.

5.2 What is a virtual topology

The communication pattern of a set of processes can be represented by a graph. The nodes represent processes, and the edges connect processes that communicate with each other. MPI provides message-passing between any pair of processes in a group. There is no requirement for opening a channel explicitly. Therefore, a *missing link* in the user-defined process graph does not prevent the corresponding processes from exchanging messages. It means rather that this connection is neglected in the virtual topology. This strategy implies that the topology gives no convenient way of naming this pathway of communication. Another possible consequence is that an automatic mapping tool (if one exists for the runtime environment) will not take account of this edge when mapping.

Specifying the virtual topology in terms of a graph is sufficient for all applications. However, in many applications the graph structure is regular, and the detailed set-up of the graph would be inconvenient for the user and might be less efficient at run time. A large fraction of all parallel applications uses process topologies like *rings, two- or higher-dimensional grids, or tori*. These structures are completely defined by the number of dimensions and the numbers of processes in each coordinate direction. Also, the mapping of grids and tori is generally an easier problem than that of general graphs. Thus, it is desirable to address these cases explicitly.

Example Cartesian Topology

Process coordinates in a Cartesian structure begin their numbering at 0. Row-major numbering is always used for the processes in a Cartesian structure. This means that, for example, the relation between group rank and coordinates for four processes in a (2 × 2) grid is as follows.

- coord (0,0): rank 0
- coord (0,1): rank 1
- coord (1,0): rank 2
- coord (1,1): rank 3

5.3 Virtual topology and Cartesian topology

MPI_GRAPH_create and MPI_CART_create are needed to produce ordinary (graph) Virtual topology and Cartesian topology. A graph can be used to represent the **virtual topology**. Each process is a point, and the connection between two points is communication and connection; nevertheless, MPI can communicate with any two processes. The absence of a connection between two sites does not rule out the possibility of communication. However, the virtual topology has a fatal flaw: it is hard to gather information about the process between the topologies naturally. More or less will produce similar issues if the number of nodes in the freshly constructed **Cartesian topology** is not equal to the number of nodes in original communication group. The processes in the new communication group will be renumbered if reorder is true.

5.4 Topologies Constructor

In the following several MPI topologies are described. The available MPI topologies are: **MPI_GRAPH** the graph topology, **MPI_CART** the cartesian topology, and **MPI_DIST_GRAPH** distributed graph topology. When is not defined a topology for a communicator the value **MPI_UNDEFINED** is used.

Moreover, MPI provides useful function to inquiry topology, here the most important:

- **MPI_TOPO_TEST** returns the type of topology that is assigned to a communicator.

- int MPI_Topo_test(MPI_Comm comm, int *status)

    - IN comm, communicator (handle)
    - OUT status, topology type of communicator comm (state)

- **MPI_GRAPH_GET** retrieve the graph-topology information that was associated with a communicator.

- int MPI_Graph_get(MPI_Comm comm, int maxindex, int maxedges, int index[],int edges[])

    - IN comm, communicator with graph structure (handle)
    - IN maxindex, length of vector index in the calling program (integer)
    - IN maxedges, length of vector edges in the calling program (integer)
    - OUT index, array of integers containing the graph structure (for details see the definition of MPI_GRAPH_CREATE)
    - OUT edges, array of integers containing the graph structure

- **MPI_CART_GET** return the Cartesian topology information that was associated with a communicator.

- int MPI_Cart_get(MPI_Comm comm, int maxdims, int dims[], int periods[], int coords[])

    - IN comm, communicator with Cartesian structure (handle)
    - IN maxdims, length of vectors dims, periods, and coords in the calling program (integer)
    - OUT dims, number of processes for each Cartesian dimension (array of integer)
    - OUT periods, periodicity (true/ false) for each Cartesian dimension (array of logical)
    - OUT coords, coordinates of calling process in Cartesian structure (array of integer)

- **MPI_CART_RANK** translates the logical process coordinates to process ranks as they are used by the point-to-point routines.

- int MPI_Cart_rank(MPI_Comm comm, const int coords[], int *rank)

    - IN comm, communicator with Cartesian structure (handle)
    - IN coords, integer array (of size ndims) specifying the Cartesian coordinates of a process
    - OUT rank, rank of specified process (integer)

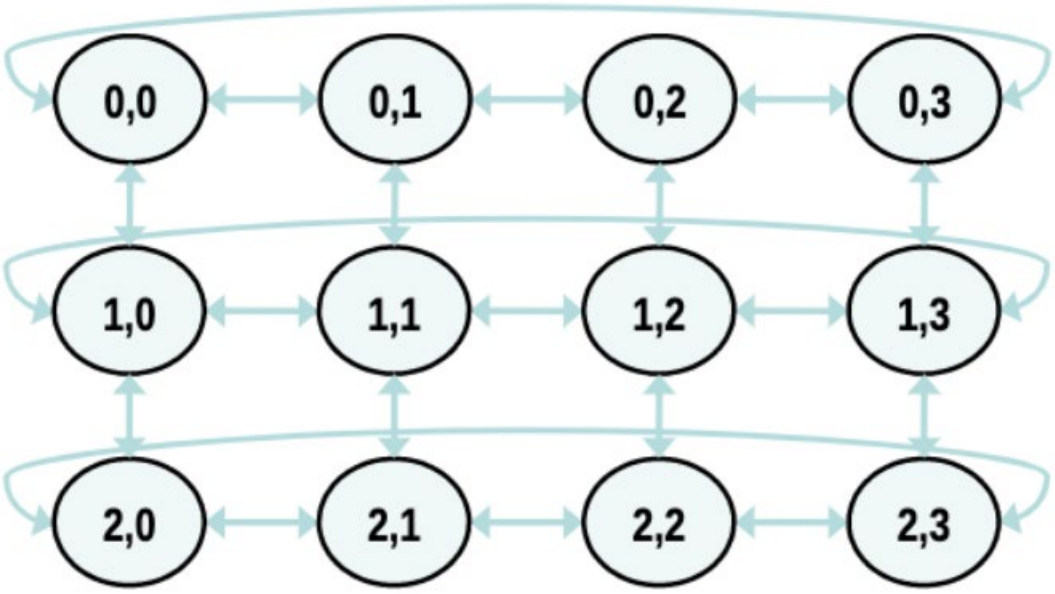
- **MPI_CART_COORDS** provides rank-to-coordinates translation (inverse mapping),

- int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int coords[])\

    - IN comm, communicator with Cartesian structure (handle)
    - IN rank, rank of a process within group of comm (integer)
    - IN maxdims, length of vector coords in the calling program (integer)
    - OUT coords, integer array (of size ndims) containing the Cartesian coordinates of specified process (array of integers)

5.5 MPI_Graph_create and MPI_Cart_create

MPI CART CREATE creates a new communicator and attaches the Cartesian topology information to it. If reorder is false, each process's rank in the new group is the same as its rank in the previous group. The function may reorganize the processes if this is not the case (possibly so as to choose a good embedding of the virtual topology onto the physical machine). Some processes are returned MPI COMM NULL if the overall size of the Cartesian grid is smaller than the size of the comm old group, similar to MPI COMM SPLIT. A zero-dimensional Cartesian topology is formed if ndims is 0. If the call specifies a grid that is greater than the group size or if ndims is negative, it is incorrect.

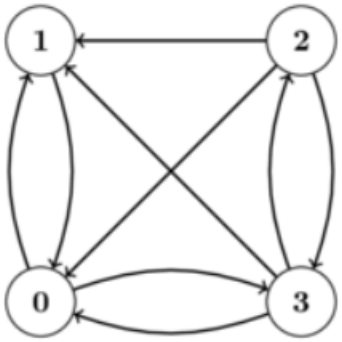


int MPI_Cart_create(MPI_Comm comm_old, int ndims, const int dims[], const int periods[], int reorder, MPI_Comm *comm_cart)

- IN comm_old, input communicator (handle)
- IN ndims, number of dimensions of Cartesian grid (integer)
- IN dims, integer array of size ndims specifying the number of processes in each dimension
- IN periods, logical array of size ndims specifying whether the grid is periodic (true) or not (false) in each dimension
- IN reorder, ranking may be reordered (true) or not (false) (logical)
- OUT comm_cart, communicator with new Cartesian topology (handle)

MPI GRAPH CREATE creates a new communicator and attaches the graph topology information to it. If reorder = false, each process's rank in the new group is the same as its rank in the previous group. The function may reorganize the processes if this is not the case. If the graph's size, nnodes, is smaller than the size of the comm old group, some processes will be returned as MPI COMM NULL, similar to MPI CART CREATE and MPI COMM SPLIT. MPI COMM NULL is returned in all processes if the graph is empty, i.e nnodes == 0. If the call specifies a graph that is greater than the input communicator's group size, it is incorrect.

The graph structure is defined by the three parameters nnodes, index, and edges. The number of nodes in the graph is nnodes. The nodes are assigned numbers ranging from 0 to nnodes-1. The total number of neighbors of the first I graph nodes is stored in the array index's i-th entry. The neighbors of nodes 0, 1,..., nnodes-1 are stored in array edges in a sequential order. The edge lists are represented flattened in the array edges. The entire number of index entries is nnodes, and the total number of edge entries is the same as the number of graph edges.



int MPI_Graph_create (MPI_Comm comm_old, int nnodes, const int index [], const int edges [], int reorder, MPI_Comm *comm_graph)

- IN comm_old, input communicator (handle)
- IN nnodes, number of nodes in graph (integer)
- IN index, array of integers describing node degrees (see below)
- IN edges, array of integers describing graph edges (see below)
- IN reorder, ranking may be reordered (true) or not (false) (logical)
- OUT comm_graph, communicator with graph topology added (handle)

5. Conclusion

Many computational and technical problems, whether by differentiation, integration, or other methods, will be reduced to either a series of matrices or some form of grid operation. The dimensions of a matrix or grid are often determined by physical problems. For the functions of multiprocessing, often those matrices or grids are partitioned, or domain-decomposed, in order that every partition (or subdomain) is assigned to a technique. Although it's far viable to nevertheless discuss with every of those pxq subdomains via way of means of a linear rank number, it's far apparent that a mapping of the linear technique rank to a 2nd digital rank numbering might facilitate a far clearer and herbal computational representation. To deal with the wishes of this and different topological layouts, the MPI library presents forms of topology: Cartesian and graph topologies. MPI always has been modified and changed into a better version, to make the working process more convenient and adding different functions. For that reason, MPI 4.0. is also under a consideration to be updated and perform following:

- Extensions to better support hybrid programming models
- Support for fault tolerance in MPI applications
- Persistent collectives
- Performance Assertions and Hints
- RMA/One-sided communication

6. Reference

(a) https://www.microsoft.com/en-us/download/details.aspx?id=57467

(b) https://www.open-mpi.org/software/ompi/v4.1/

(c) https://www.mpich.org/downloads/

(d) https://learn.microsoft.com/en-us/message-passing-interface/microsoft-mpi

(e) https://www.ibm.com/docs/en/iis/11.3?topic=topology-parallel-processing-grid-topologies

(f) https://www.cs.iusb.edu/~danav/teach/b424/b424_24_networks.html

(g) https://youtu.be/nNCDgAU76D0

(h)https://www.codingame.com/playgrounds/47058/have-fun-with-mpi-in-c/mpi-process-topologies