MONASH University

FIT3143 Tutorial Week 10

PARALLEL ALGORITHM DESIGN – ADVANCED MPI TOPICS

OBJECTIVES

- Understanding the principles of MPI Scatter, MPI Gather and Virtual topologies.
- Solving parallel programming problems using MPI Scatter and Gather functions.
- Design solutions using MPI virtual topologies.

Note: Tutorials are not assessed. Nevertheless, please attempt the questions to improve your unit comprehension in preparation for the labs, assignments, and final assessments.

QUESTIONS

1. How is MPI Scatter different from MPI Broadcast? In addition, how is MPI Gather different from MPI Reduce?

2. This following code file implements a simple parallel vector multiplication using MPI. Modify its code to replace the MPI Send and Recv functions with MPI Scatter and MPI Gather functions.

   Note: There is no need to compile the code, focus on writing a logically correct code to replace the MPI Send and Recv functions with MPI scatter and gather functions.

3. Explain the concept of MPI virtual topologies and its benefits.

4. A high-rise building management is planning to install a series of fire alarm sensors representing a form of a 3D mesh architecture as illustrated in Figure 1.

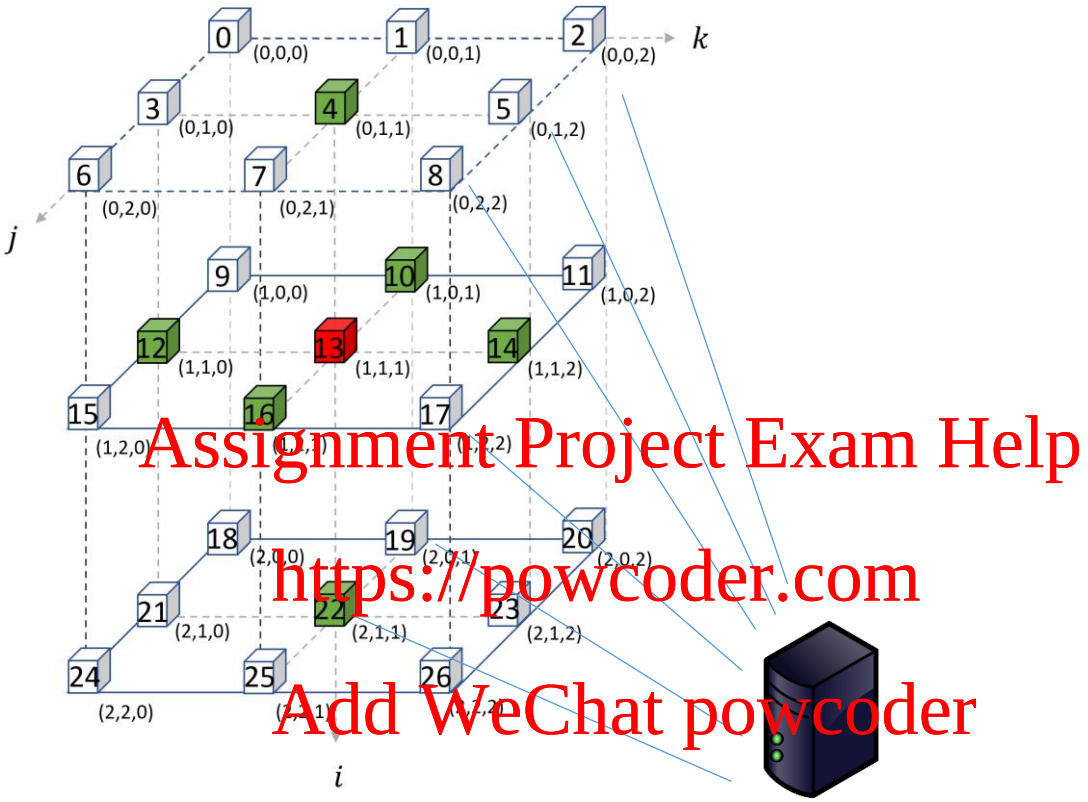

Figure 1: 3D mesh architecture of the fire alarm sensors.

In Figure 1, each sensor can directly communicate with its immediate adjacent sensors (i.e., top, bottom, left, right, front, and back). Each sensor can also directly communicate with the server.

Based on this architecture, there are two options to implement the fire alarm computing and communication system.

Option A:

I) At each interval, the sensor measures the temperature and exchanges the temperature with its neighbours.

II) If the exchanged temperature values and measured values exceed a particular threshold, the sensor sends an alert to the server, which is located outside of the building.

III) The server listens for incoming alerts from the sensor nodes and logs it.

Option B:

I) At each interval, the sensor measures the temperature and directly sends the measured value to the server.

II) The server periodically receives temperatures readings from all sensors. At each iteration, the server then compares the temperature values of each node with the adjacent nodes to determine if a fire is detected. In other words, all of the computations are done at the server.

Before implementing the architecture, a simulator is created using Message Passing Interface (MPI). Based on the aforementioned description and illustration, answer the following questions:

a) Compare **Options A** and **B**. In particular, what type of distributed computing architectures (in relation to computation and communication) do **Options A** and **B** represent respectively?

b) What is the advantage of **Option A** to that of **Option B** in terms of message passing communication?

The following code snippet describes an attempt to simulate the sensor based on **Option A**. This code first splits the communicator between the server and sensor nodes. Then, a 3D grid using MPI virtual topology is created for the MPI processes simulating the sensors. This code however is incomplete. Based on the given code snippet, answer the remaining questions.

c) Why should the **MPI_Cart_create()** function be invoked by all of the MPI processes simulating the sensor nodes? What happens if any one of the MPI processes simulating the sensor nodes does not invoke the **MPI_Cart_create()** function?

d) When passing in the first argument into the **MPI_Cart_create()** function, why doesn't this function use the default **MPI_COMM_WORLD** communicator?

e) The **MPI_Cart_coords()** function computes the process coordinates in a 3D cartesian topology based on the given rank in a group. This function essentially performs a 1D (i.e., rank index) to 3D (i.e., coordinates) mapping based on the dimension of the grid. Assuming this function is not available and that you are required to manually calculate the coordinates, what are the equations which map a 1D rank value, ***x*** to the 3D coordinates ***i, j, k*** based on the ***row width***, ***column width*** and ***depth*** of the grid?

f) The **MPI_Cart_rank()** function computes the process rank in communicator based on the given Cartesian coordinate. This function essentially performs a 3D (i.e., coordinates) to 1D (i.e., rank index) mapping based on the dimension of the grid. Assuming this function is also not available and that you are required to manually calculate the the 1D cartesian rank, what is the equation to which maps the 3D coordinates ***i, j, k*** to a 1D rank value, ***x***, based on the ***row width***, ***column width*** and ***depth*** of the grid?

Hint: Refer to this website on mapping for some guidance.

g) The **sensor_io()** function in the given code below requires each node to exchange the temperature values with its adjacent nodes. However, this region of the code is incomplete. Complete this region of the code by using non-blocking MPI send and receive functions to exchange the temperature values. You do not need to copy the entire given code into your answer template. Only write the missing code in your answer template. Use a **for** loop to implement the send and receive functions and use the available variables in the given code below. You may opt to create new variables or arrays.

Note: There is no need to compile the code, focus on writing a logically correct code.

Code snippet implementing Option A (Refer to the /* INCOMPLETE REGION - START */ in the code to complete part (g)).

```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
#include <unistd.h>
#include <string.h>

#define NON_RANGE 100
#define SHIFT_ROW 0
#define SHIFT_COL 1
#define SHIFT_DEP 2
#define DISP 1

int sensor_io(MPI_Comm world_comm, MPI_Comm comm);
int MeasureTemperature();
bool CheckTemperature(int *recvValues, int temp);
int server_io(MPI_Comm world_comm, MPI_Comm comm);

int main(int argc, char **argv){
    int rank, size;
    MPI_Comm new_comm;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Comm_split( MPI_COMM_WORLD,rank == size-1, 0, &new_comm);
    if (rank == size-1)
     server_io( MPI_COMM_WORLD, new_comm );
    else
     sensor_io( MPI_COMM_WORLD, new_comm );
    MPI_Finalize();
    return 0;
}
int sensor_io(MPI_Comm world_comm, MPI_Comm comm) {
     int ndims=3, size, my_rank;
     int reorder, my_cart_rank, ierr, worldSize;
     int nbr_i_lo, nbr_i_hi;
     int nbr_j_lo, nbr_j_hi;
     int nbr_k_lo, nbr_k_hi;
     MPI_Comm comm3D;
     int dims[ndims],coord[ndims];
     int wrap_around[ndims];
     char buf[256];

     MPI_Comm_size(world_comm,  &worldSize);  //  size  of  the  world
     communicator
```

```
    MPI_Comm_size(comm, &size); // size of the slave communicator
    MPI_Comm_rank(comm, &my_rank); // rank within the slave communicator

    dims[0]=dims[1]=dims[2]=0;
    MPI_Dims_create(size, ndims, dims);

    wrap_around[0] = 0;
    wrap_around[1] = 0;
    wrap_around[2] = 0;
    reorder = 1;
    ierr = 0;
    ierr  =  MPI_Cart_create(comm,  ndims,  dims,  wrap_around,  reorder,
    &comm3D);
    if(ierr != 0) printf("ERROR[%d] creating CART\n",ierr);

    MPI_Cart_coords(comm3D, my_rank, ndims, coord);
    MPI_Cart_rank(comm3D, coord, &my_cart_rank);

    MPI_Cart_shift( comm3D, SHIFT_ROW, DISP, &nbr_i_lo, &nbr_i_hi);
    MPI_Cart_shift( comm3D, SHIFT_COL, DISP, &nbr_j_lo, &nbr_j_hi);
    MPI_Cart_shift( comm3D, SHIFT_DEP, DISP, &nbr_k_lo, &nbr_k_hi);

    MPI_Request send_request[6];
    MPI_Request receive_request[6];
    MPI_Status send_status[6];
    MPI_Status receive_status[6];

    sleep(my_rank);
    int temp = MeasureTemperature();
    int recvValues[6] = {-1, -1, -1, -1, -1, -1};

    /* INCOMPLETE REGION - START */
    /* COMPLETE PART (S) HERE */

    /* INCOMPLETE REGION - END */

    if(CheckTemperature(recvValues, temp) == 1){
    sprintf(buf,  "Fire  alert  from  slave  %d  at  Coord:  (%d,  %d,  %d).
    Temperature: %d\n", my_rank, coord[0], coord[1], coord[2], temp);
    MPI_Send(buf, strlen(buf) + 1, MPI_CHAR, worldSize-1, 0, world_comm);
    }
    MPI_Comm_free( &comm3D );
    return 0;
}
bool CheckTemperature(int* recvValues, int temp){
    int retVal = 0;
    for (int i = 0; i < 6; i++) {
    retVal = retVal && (recvValues[i] == temp || recvValues[i] == -1);
    }
    return retVal;
}
int MeasureTemperature() {
    srand(time(NULL));
    int number;
    number = rand() % (NUM_RANGE + 1);
    return number;
}
int server_io(MPI_Comm world_comm, MPI_Comm comm){
    // Not applied to the context of the question

    }
```