San Jose State University
Department of Electrical Engineering
Laboratory Assignment #1

Part 1 OpenMP (20 points): Log onto a node requesting all the cores (28 on a regular PSC Bridges node). Edit laplace_serial.c and add OpenMP directives where it helps. Run your code on various numbers of cores (such as 1, 2, 4, 8, 16, 20, 24, and 28) and present the speedup you achieve in **Table.** Your report should include a table and source code. Attach execution log or slurm outputs for those cases.The laplace_serial OpenMP version should converge at 3372 iterations.

Ans.

Source Code:

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/time.h>

// size of plate
#define COLUMNS    1000
#define ROWS       1000

// largest permitted change in temp
#define MAX_TEMP_ERROR 0.01

double Temperature[ROWS+2][COLUMNS+2];      // temperature grid
double Temperature_last[ROWS+2][COLUMNS+2]; // temperature grid from last
iteration

//   helper routines
void initialize();
void track_progress(int iter);


int main(int argc, char *argv[]) {

    int i, j;                            // grid indexes
    int max_iterations;                  // number of iterations
    int iteration=1;                     // current iteration
    double dt=100;                       // largest change in t
    struct timeval start_time, stop_time, elapsed_time;  // timers

    printf("Maximum iterations [100-4000]?\n");
    scanf("%d", &max_iterations);

    gettimeofday(&start_time,NULL); // Unix timer

    initialize();   // initialize Temp_last including boundary conditions

    // do until error is minimal or until max steps
    while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
```

```
        // main calculation: average my four neighbors
        #pragma omp parallel for private(i,j)
        for(i = 1; i <= ROWS; i++) {
            for(j = 1; j <= COLUMNS; j++) {
                Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] +
Temperature_last[i-1][j] +
                                            Temperature_last[i][j+1] +
Temperature_last[i][j-1]);
            }
        }

        dt = 0.0; // reset largest temperature change

        // copy grid to old grid for next iteration and find latest dt
        #pragma omp parallel for reduction(max:dt) private(i,j)
        for(i = 1; i <= ROWS; i++){
            for(j = 1; j <= COLUMNS; j++){
              dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]),
dt);
              Temperature_last[i][j] = Temperature[i][j];
            }
        }

        // periodically print test values
        if((iteration % 100) == 0) {
            track_progress(iteration);
        }

        iteration++;
    }

    gettimeofday(&stop_time,NULL);
    timersub(&stop_time, &start_time, &elapsed_time); // Unix time
subtract routine

    printf("\nMax error at iteration %d was %f\n", iteration-1, dt);
    printf("Total time was %f seconds.\n",
elapsed_time.tv_sec+elapsed_time.tv_usec/1000000.0);

}


// initialize plate and boundary conditions
// Temp_last is used to to start first iteration
void initialize(){

    int i,j;
    for(i = 0; i <= ROWS+1; i++){
        for (j = 0; j <= COLUMNS+1; j++){
            Temperature_last[i][j] = 0.0;
        }
    }
```

```
    // these boundary conditions never change throughout run

    // set left side to 0 and right to a linear increase
    for(i = 0; i <= ROWS+1; i++) {
        Temperature_last[i][0] = 0.0;
        Temperature_last[i][COLUMNS+1] = (100.0/ROWS)*i;
    }

    // set top to 0 and bottom to linear increase
    for(j = 0; j <= COLUMNS+1; j++) {
        Temperature_last[0][j] = 0.0;
        Temperature_last[ROWS+1][j] = (100.0/COLUMNS)*j;
    }
}


// print diagonal in bottom right corner where most action is
void track_progress(int iteration) {

    int i;

    printf("---------- Iteration number: %d ------------\n", iteration);
    for(i = ROWS-5; i <= ROWS; i++) {
        printf("[%d,%d]: %5.2f  ", i, i, Temperature[i][i]);
    }
    printf("\n");
}
```

Table For Speedup:

Number Of Nodes	Time Of Execution (seconds)	Speed Up
1	15.584436	
2	7.948068	1.9607
4	3.959355	3.9361
8	2.098698	7.4257
16	1.377878	11.3104
20	1.118951	13.9277
24	0.877376	17.7625
28	0.767689	20.3004

Part 2 OpenACC (20 points): Edit laplace_serial.c and add OpenACC directives where it helps. Run your code on GPU node and present the speedup you achieve in Table. Your report should include a table and source code. Additionally, you should attach execution log or slurm output.

Ans.

Source Code:

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/time.h>

// size of plate
#define COLUMNS    1000
#define ROWS       1000

// largest permitted change in temp
#define MAX_TEMP_ERROR 0.01

double Temperature[ROWS+2][COLUMNS+2];      // temperature grid
double Temperature_last[ROWS+2][COLUMNS+2]; // temperature grid from last
iteration

//   helper routines
void initialize();
void track_progress(int iter);


int main(int argc, char *argv[]) {

    int i, j;                                            // grid indexes
    int max_iterations;                                 // number of iterations
    int iteration=1;                                     // current iteration
    double dt=100;                                      // largest change in t
    struct timeval start_time, stop_time, elapsed_time;  // timers

    printf("Maximum iterations [100-4000]?\n");
    scanf("%d", &max_iterations);

    gettimeofday(&start_time,NULL); // Unix timer

    initialize();    // initialize Temp_last including boundary conditions

    // do until error is minimal or until max steps
    #pragma acc data copy (Temperature_last), create(Temperature)
    while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {

        // main calculation: average my four neighbors
                //#pragma omp parallel for private(i,j)
        #pragma acc kernels
                for(i = 1; i <= ROWS; i++) {
            for(j = 1; j <= COLUMNS; j++) {
```

```
                Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] +
Temperature_last[i-1][j] +
                                            Temperature_last[i][j+1] +
Temperature_last[i][j-1]);
            }
        }

        dt = 0.0; // reset largest temperature change

        // copy grid to old grid for next iteration and find latest dt
        //pragma omp parallel for reduction(max:dt) private(i,j)
        #pragma acc kernels
                for(i = 1; i <= ROWS; i++){
            for(j = 1; j <= COLUMNS; j++){
           dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
              Temperature_last[i][j] = Temperature[i][j];
            }
        }

        // periodically print test values
        #pragma acc update host (Temperature)
        if((iteration % 100) == 0) {
            track_progress(iteration);
        }

        iteration++;
    }

    gettimeofday(&stop_time,NULL);
    timersub(&stop_time, &start_time, &elapsed_time); // Unix time
subtract routine

    printf("\nMax error at iteration %d was %f\n", iteration-1, dt);
    printf("Total time was %f seconds.\n",
elapsed_time.tv_sec+elapsed_time.tv_usec/1000000.0);

}


// initialize plate and boundary conditions
// Temp_last is used to to start first iteration
void initialize(){

    int i,j;

    for(i = 0; i <= ROWS+1; i++){
        for (j = 0; j <= COLUMNS+1; j++){
            Temperature_last[i][j] = 0.0;
        }
    }

    // these boundary conditions never change throughout run

    // set left side to 0 and right to a linear increase
```

```
    for(i = 0; i <= ROWS+1; i++) {
        Temperature_last[i][0] = 0.0;
        Temperature_last[i][COLUMNS+1] = (100.0/ROWS)*i;
    }

    // set top to 0 and bottom to linear increase
    for(j = 0; j <= COLUMNS+1; j++) {
        Temperature_last[0][j] = 0.0;
        Temperature_last[ROWS+1][j] = (100.0/COLUMNS)*j;
    }
}


// print diagonal in bottom right corner where most action is
void track_progress(int iteration) {

    int i;

    printf("---------- Iteration number: %d ------------\n", iteration);
    for(i = ROWS-5; i <= ROWS; i++) {
        printf("[%d,%d]: %5.2f  ", i, i, Temperature[i][i]);
    }
    printf("\n");
}
```

Table of Speedup Comparison:

Number Of Nodes	Time Of Execution (seconds)	Speed Up
OMP1	15.584436	
OMP2	7.948068	1.9607
OMP4	3.959355	3.9361
OMP8	2.098698	7.4257
OMP16	1.377878	11.3104
OMP20	1.118951	13.9277
OMP24	0.877376	17.7625
OMP28	0.767689	20.3004
OACC	1.797815	8.6685

Part 3 MPI (30 points): Write a code that runs on 8 PEs and does a “circular shift.” This means that every PE sends some data to its nearest neighbor either “up” (one PE higher) or “down.” To make it circular, PE 7 and PE 0 are treated as neighbors. Make sure that whatever data you send is received. e.g. send PE number. Present the execution result. Your report should include **commented source code.**

Ans.

Source Code:

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

main(int argc, char** argv){

int my_PE_num, numbertoreceive; //defining the variables//

MPI_Status status;  //checks the status of  MPI//
MPI_Init(&argc, &argv);  //calling the first routine//
MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);  //This routine returns to
every PE its rank, or unique address from 0 to PEs-1//
MPI_Barrier(MPI_COMM_WORLD);  //putting barrier to display output in
proper order//

        switch (my_PE_num)  //making a switch to perform the operation
using case//
        {
                //start of operation//
                case 0:  //Receiving from PE1 and sending to PE7//
                        {
                                MPI_Recv( &numbertoreceive, 1, MPI_INT,1,
MPI_ANY_TAG, MPI_COMM_WORLD, &status);  //giving address of PE1//
                                printf("PE %d received: %d\n",
my_PE_num,numbertoreceive);  //printing the output//
                                MPI_Send( &my_PE_num, 1, MPI_INT, 7, 10,
MPI_COMM_WORLD);   //giving address of PE7//
                        }

                case 1:  //Receiving from PE2 and sending to PE0//
                        {
                                MPI_Recv( &numbertoreceive, 1,
MPI_INT,my_PE_num+1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
//incrementing PE to +1 to change the receiving PE//
                                printf("PE %d received: %d\n",
my_PE_num,numbertoreceive);  //printing the output//
                                MPI_Send( &my_PE_num, 1, MPI_INT,
my_PE_num-1, 10, MPI_COMM_WORLD);  //decrementing PE to -1 to change the
semding PE//
                        }

                case 2:  //Receiving from PE3 and sending to PE1//
                        {
                                MPI_Recv( &numbertoreceive, 1,
MPI_INT,my_PE_num+1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
//incrementing PE to +1 to change the receiving PE//
```

```
                                printf("PE %d received: %d\n",
my_PE_num,numbertoreceive);  //printing the output//
                                MPI_Send( &my_PE_num, 1, MPI_INT,
my_PE_num-1, 10, MPI_COMM_WORLD);   //decrementing PE to -1 to change the
semding PE//
                        }

                case 3:  //Receiving from PE4 and sending to PE2//
                        {
                                MPI_Recv( &numbertoreceive, 1,
MPI_INT,my_PE_num+1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
//incrementing PE to +1 to change the receiving PE//
                                printf("PE %d received: %d\n",
my_PE_num,numbertoreceive);  //printing the output//
                                MPI_Send( &my_PE_num, 1, MPI_INT,
my_PE_num-1, 10, MPI_COMM_WORLD); //decrementing PE to -1 to change the
semding PE//
                        }

                case 4:  //Receiving from PE5 and sending to PE3//
                        {
                                MPI_Recv( &numbertoreceive, 1,
MPI_INT,my_PE_num+1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
//incrementing PE to +1 to change the receiving PE//
                                printf("PE %d received: %d\n",
my_PE_num,numbertoreceive);  //printing the output//
                                MPI_Send( &my_PE_num, 1, MPI_INT,
my_PE_num-1, 10, MPI_COMM_WORLD); //decrementing PE to -1 to change the
semding PE//
                        }

                case 5:  //Receiving from PE6 and sending to PE4//
                        {
                        MPI_Recv( &numbertoreceive, 1,
MPI_INT,my_PE_num+1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
//incrementing PE to +1 to change the receiving PE//
                                printf("PE %d received: %d\n",
my_PE_num,numbertoreceive);  //printing the output//
                                MPI_Send( &my_PE_num, 1, MPI_INT,
my_PE_num-1, 10, MPI_COMM_WORLD); //decrementing PE to -1 to change the
semding PE//
                        }

                case 6:  //Receiving from PE7 and sending to PE5//
                        {
                                MPI_Recv( &numbertoreceive, 1,
MPI_INT,my_PE_num+1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
//incrementing PE to +1 to change the receiving PE//
                                printf("PE %d received: %d\n",
my_PE_num,numbertoreceive); //printing the output//
                                MPI_Send( &my_PE_num, 1, MPI_INT,
my_PE_num-1, 10, MPI_COMM_WORLD); //decrementing PE to -1 to change the
semding PE//
                        }
```

```
                default:  //Receiving from PE0 and sending to PE6//
                        {
                                MPI_Send( &my_PE_num, 1, MPI_INT, 6, 10,
MPI_COMM_WORLD); //giving address of PE6//
                        }

        }

        MPI_Recv( &numbertoreceive, 1, MPI_INT,0, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);  //giving address of PE0//
        printf("PE %d received: %d\n", my_PE_num,numbertoreceive);
//printing the output//

  MPI_Finalize(); //This is the companion to MPI_Init//
 }
```

Output:

```
[sshah4@r001 ~]$ module load pgi
[sshah4@r001 ~]$ mpicc MPI3.c
[sshah4@r001 ~]$ mpirun -n 8 ./a.out
PE 6 received: 7
PE 5 received: 6
PE 4 received: 5
PE 3 received: 4
PE 2 received: 3
PE 1 received: 2
PE 0 received: 1
PE 7 received: 0
```

Part 4 MPI (30 points): Using only the routines that we have covered in the three examples in lectures (MPI_Init, MPI_Comm_Rank, MPI_Send, MPI_Recv, MPI_Barrier, MPI_Finalize), Write a program that determines how many PEs it is running on. It should perform like the following:

mpirun –n 4 ./a.out
I am running on 4 PEs.

mpirun –n 16 ./a.out
I am running on 16 PEs.

Ans.

Source Code:

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

int main(int argc, char** argv)  //main function//
{
    int rank,tea, i, function, error; //initialization stage//
    int my_PE_num, numbertoreceive;

    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); //to get rank of mpi//
    MPI_Errhandler_set(MPI_COMM_WORLD,MPI_ERRORS_RETURN); //to set mpi
error handler to return error value//

                i = 0;
                tea=0;  //tea = temperary variable//
                while(i!=70) //took a random value of 70 to check the
error//
        {
                tea+=1;
                //error=MPI_Send( &function, 1, MPI_INT, i, 10,
MPI_COMM_WORLD);
                        error=MPI_Bcast(&function, 1, MPI_INT, i,
MPI_COMM_WORLD); //to brodcast a function value//
                                if(error!=0) //to detect and print the
error//
                                {
                                        printf("\nI am running on %d
PEs\n",tea-1);

        MPI_Abort(MPI_COMM_WORLD,error);//to stop the operation//
                                }
                        i+=1;
                }
    MPI_Finalize();
    return 0;
}
```

Output:

```
[sshah4@r002 ~]$ mpicc MPI4.c
[sshah4@r002 ~]$ mpirun -n 4 ./a.out

I am running on 4 PEs
application called MPI_Abort(MPI_COMM_WORLD, 940143879) - process 3
[sshah4@r002 ~]$ mpirun -n 8 ./a.out

I am running on 8 PEs
application called MPI_Abort(MPI_COMM_WORLD, 269055239) - process 7
[sshah4@r002 ~]$ mpirun -n 16 ./a.out

I am running on 16 PEs
application called MPI_Abort(MPI_COMM_WORLD, 619783) - process 15
[sshah4@r002 ~]$ mpirun -n 28 ./a.out

I am running on 28 PEs
application called MPI_Abort(MPI_COMM_WORLD, 671708423) - process 27
[sshah4@r002 ~]$
```