Report on :

Time Dependent 2D Heat Equation Problem

Course: Parallel and Concurrent Programming

Instructor: Dr. Shirley Moore

- By Madhuri Nannaware

Contents:

Given :

2D Heat Equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

On the domain $\Omega = [0,1]^2$ with initial condition $u(x,,0) = f(x,y)$

Boundary conditions :

$u(0,y,t) = u(1,y,t) = u(x,0,t) = u(x,1,t) = 0$

$x= ih, y= jh$, $t_n= nk$ are discrete points on grid along x and y dimension and time respectively

$\boldsymbol{\alpha} = k/h^2$ β

=1- 4 $\boldsymbol{\alpha}$

let $V^n_{i,j}$ *be* discretization of initil $u_{i,j}$ over time interval h.

thus we can compute $V^n_{i,j}$ with the following expression :

$$\begin{aligned} v_{i,j}^{(n+1)} &= v_{i,j}^{(n)} + \frac{k}{h^2}(v_{i-1,j}^{(n)} + v_{i,j-1}^{(n)} - 4v_{i,j}^{(n)} + v_{i,j+1}^{(n)} + v_{i+1,j}^{(n)}) \\ &= \alpha v_{i-1,j}^{(n)} + \alpha v_{i,j-1}^{(n)} + \beta v_{i,j}^{(n)} + \alpha v_{i,j+1}^{(n)} + \alpha v_{i+1,j}^{(n)}, \end{aligned}$$

In algorithm we have :

N = number of intervals all together n =

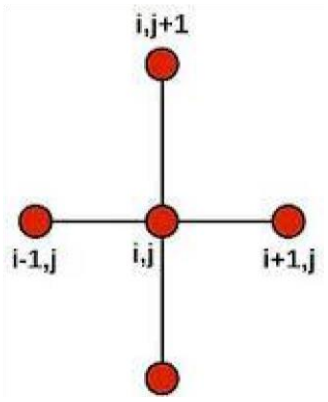
number of intervals across each dimension h =

step size between intervals

k = uniform step size = 0.00001 (assumed value)

Serial Algorithm :

As mentioned in problem We have used 5 point stencil model for computing the value of $V^{n+1}_{i,j}$. that means value of $V^{n+1}_{i,j}$ depends on $V^{n}_{i,j}$ *at the given point* as well as the surrounding points four neighboring points.



Grid is defined with extreme border points considered as zero for further calculation

Exchange border grid points at each point with up, down, left and right elements and own value. Compute next value based on these five.

```
                    for (i = 1; i < n-1; i++)
          for (j = 1; j < n-1; j++)

                              {
                                        aNew[i*n+j] = beta*a[i*n + j] + alpha*(a[(i - 1)*n + j] + a[(i + 1)*n + j]
+ a[i*n + j - 1] + a[i*n + j + 1]);  /*5 point avg updation*/
                              }
```

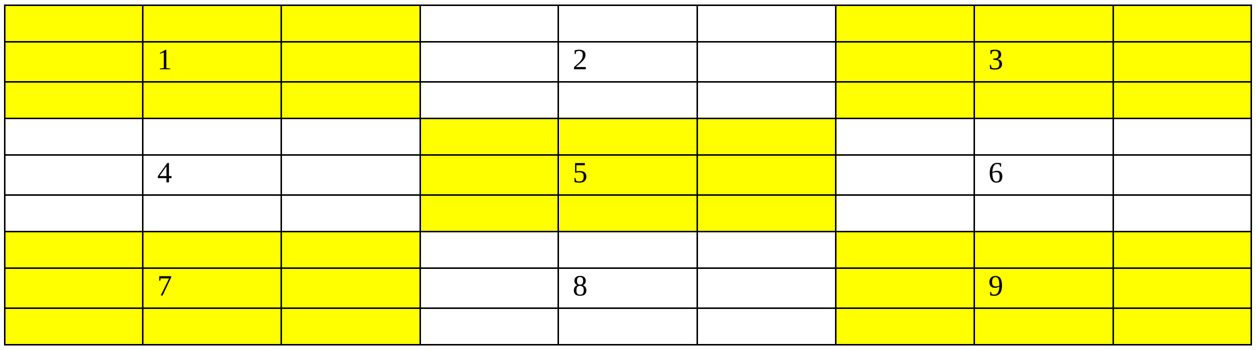
Until the total number of time steps are covered.

Performance optimization using MPI :

Algorithm

In this grid is divided into blocks. so as to initialize the parallel computation. We are considering with only one layer of data and using same 5-point stencil model within distributed block. Thus this algorithm implement simple one layer exchange of data to generate inputs for next iteration

Lets say the grid is 9*9 then topology for the distributed blocks are defined as follows :



so as to initialize the parallel computation

For block 1 :

Its top left block. To carry out one iteration on this block data has to be exchanged with block 2 and block 4.

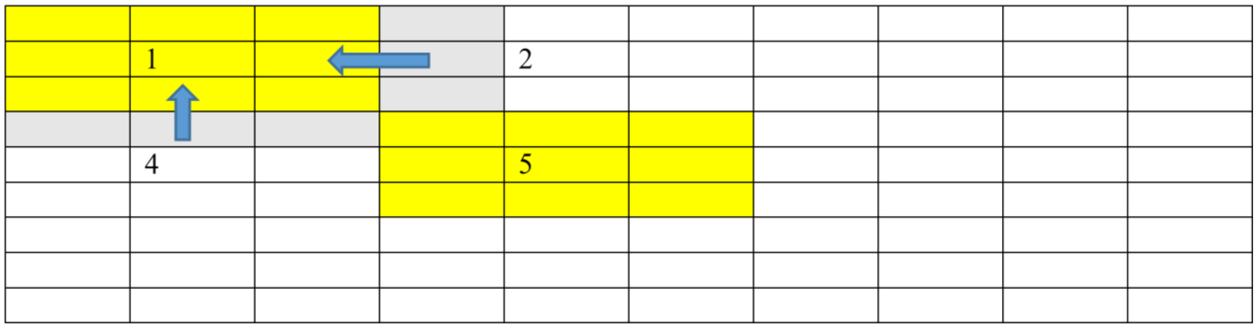


Figure : block 1

For block 1 the data exchange is from right side i.e left row of block 2 and

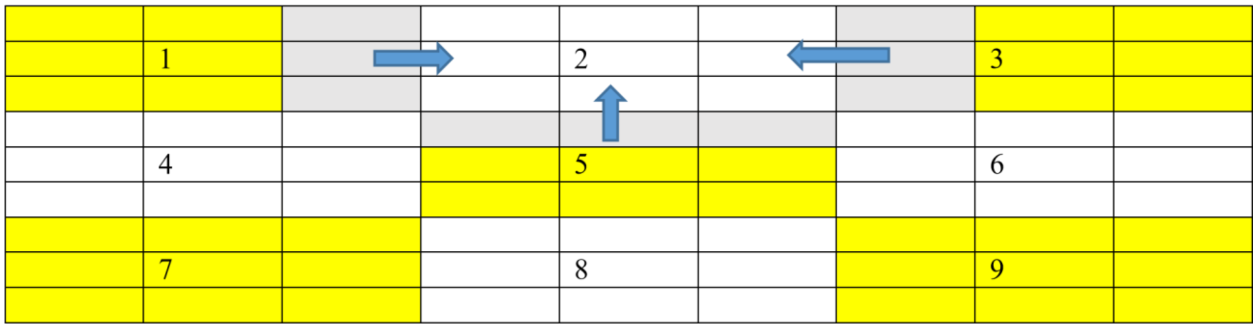
From bottom   i.e. upper row of block 4

Code line for this block:

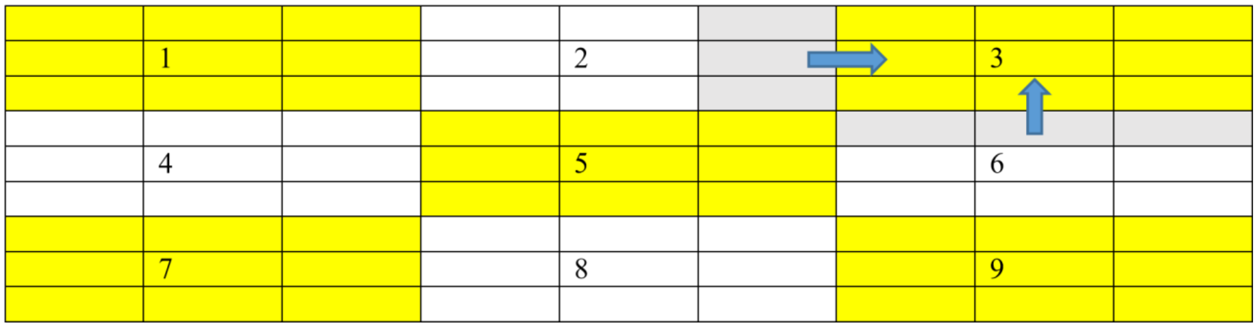
```
for (i=0;i&lt;nforDB+row;i++)

for (j=0;j&lt;nforDB+row;j++)

{
5 point calculation
}                                   /* nforDB is current value distributed block*/
```

Similarly in further tables I have highlighted the rows from which the data is to be exchanged for the given block :

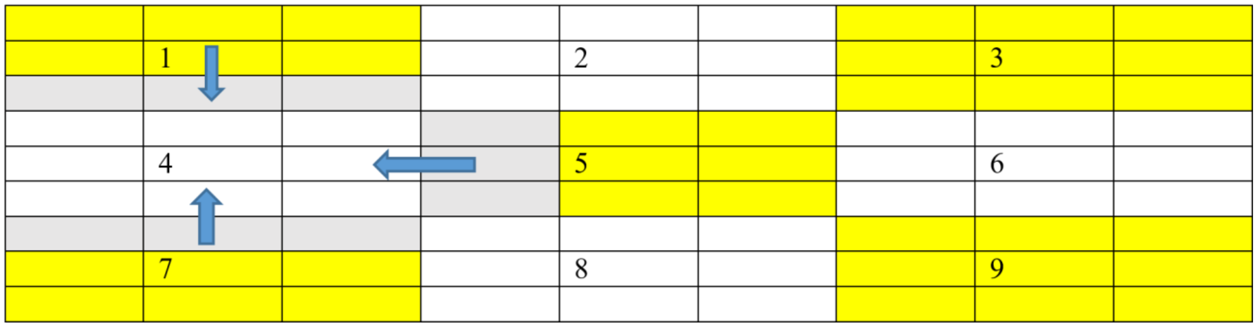
For block 2 :



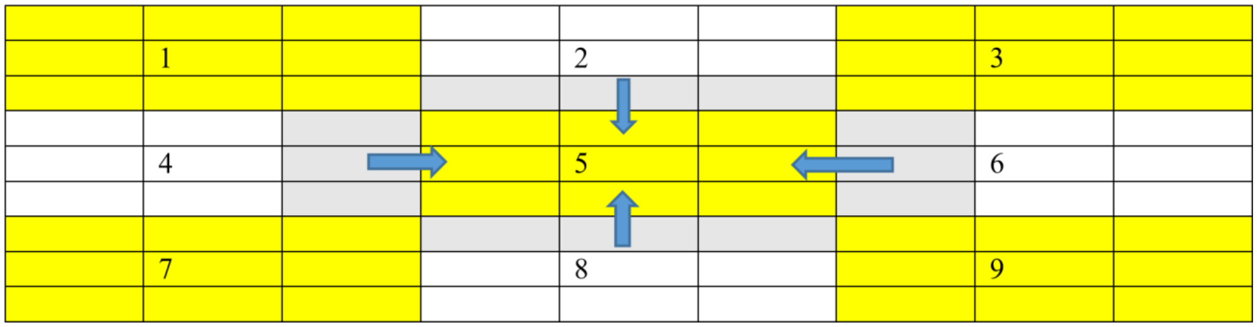
For block 3



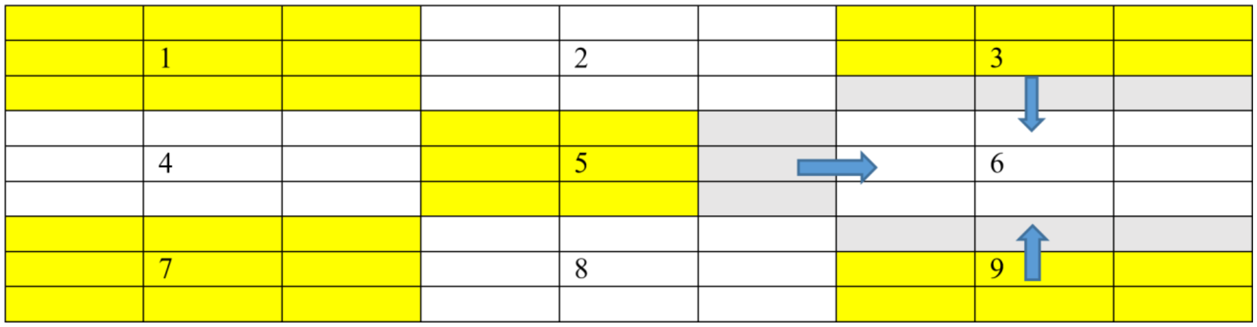
For block 4



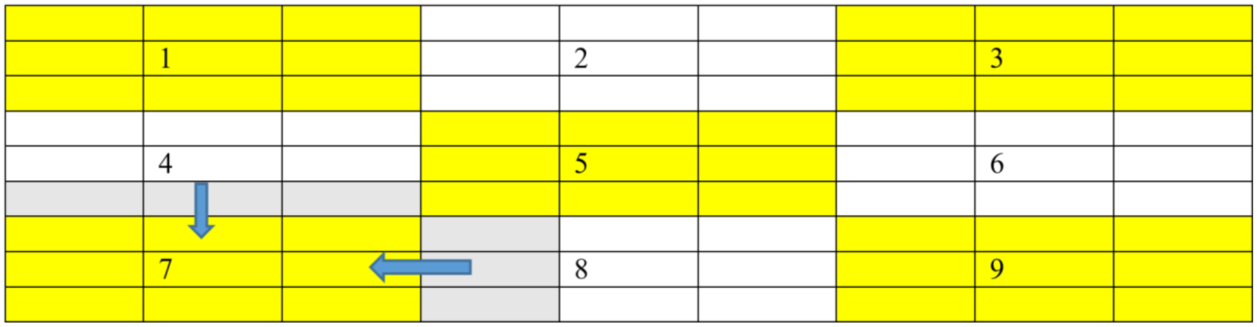
For block 5



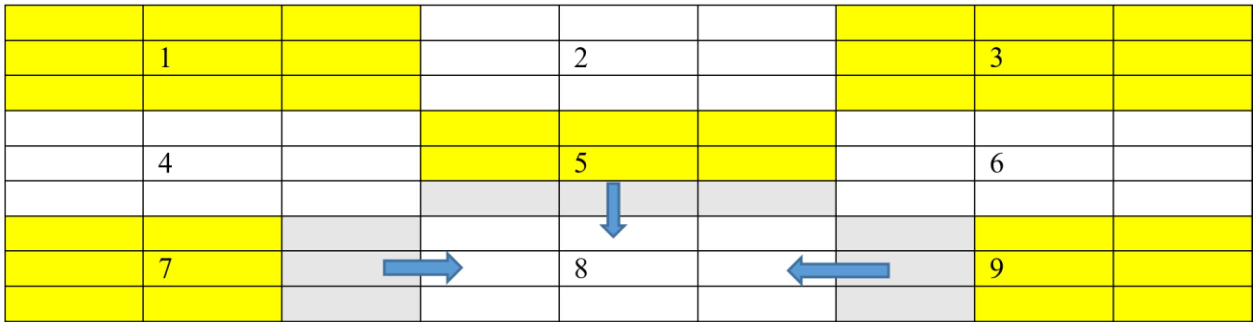
For block 6



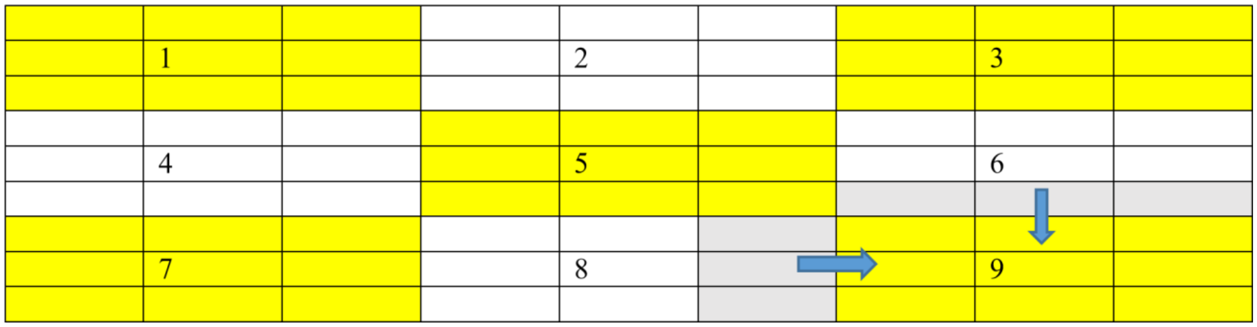
For block 7



For block 8



For block 9

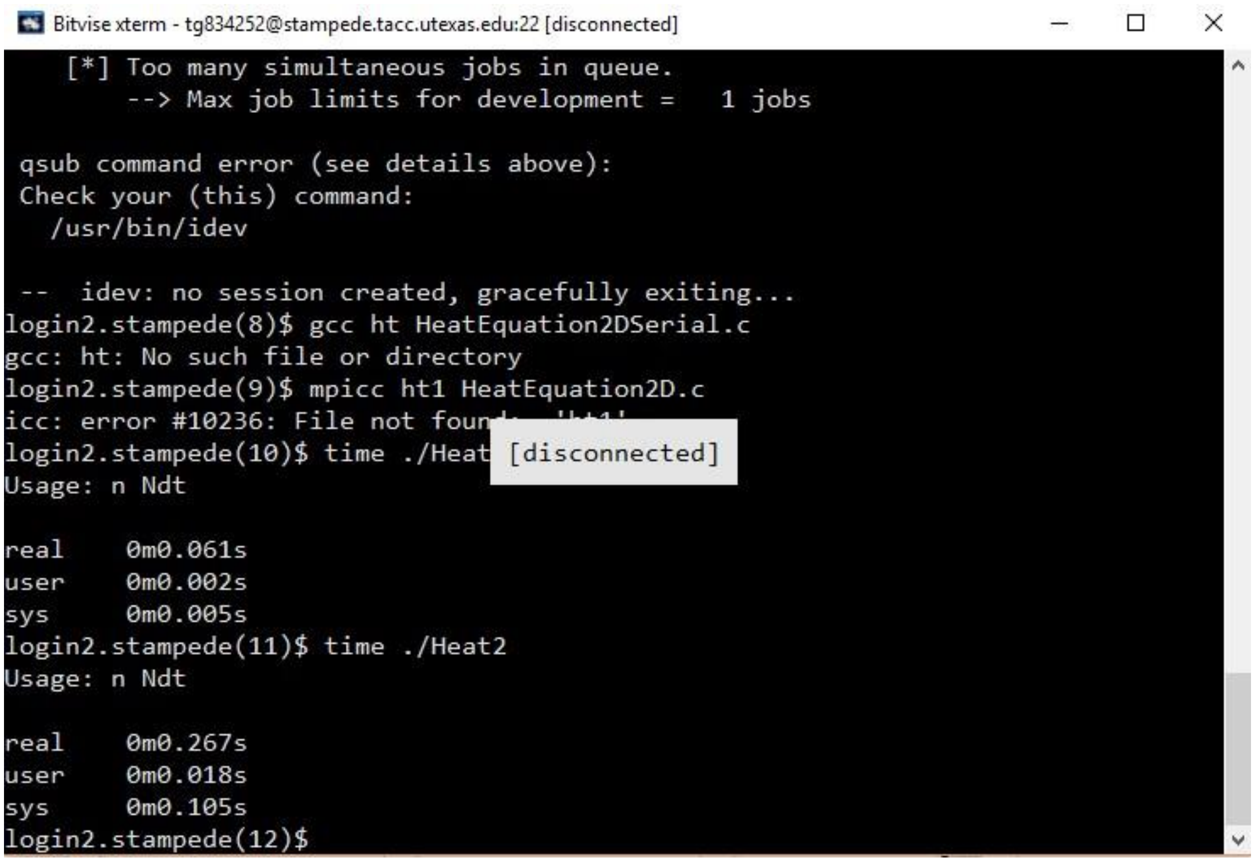


After all blocks are done with their calculations and data exchange for the given nth iteration then (n+1)th iteration is carried out.

Program Limitation:

This code can work when both the whole grid and its distributed blocks both are square ordered. .e. for 9*9 or 16*16 etc

Program Scalability and comparison


Bitvise xterm - tg834252@stampede.tacc.utexas.edu:22 [disconnected]

```
   [*] Too many simultaneous jobs in queue.
       --> Max job limits for development =   1 jobs

 qsub command error (see details above):
 Check your (this) command:
   /usr/bin/idev

 --  idev: no session created, gracefully exiting...
login2.stampede(8)$ gcc ht HeatEquation2DSerial.c
gcc: ht: No such file or directory
login2.stampede(9)$ mpicc ht1 HeatEquation2D.c
icc: error #10236: File not foun
login2.stampede(10)$ time ./Heat [disconnected]
Usage: n Ndt

real    0m0.061s
user    0m0.002s
sys     0m0.005s
login2.stampede(11)$ time ./Heat2
Usage: n Ndt

real    0m0.267s
user    0m0.018s
sys     0m0.105s
login2.stampede(12)$
```

Time required for serial code to compile :

0.267sec

Time required for parallel program to compile: 0.061sec

Speedup:

	Problem Size	Serial Runtime	Parallel Runtime		Speedup	
			With 16 processors	With 4		
1	Grid size 16 and iteration 1000	0.000000	0.002218	0.000877	-	-
2	Grid size 64 and iteration 2000	0.06	0.021941	0.010381	2.7346	5.7798

3	Grid size 64 problem iteration 1000	0.03	0.011647	0.005634	2.5758	5.3248
4	Grid Size 256 problem iteration 10000	6.03	0.165227	0.031891	36.4953	189.081
5	Grid size 625 problem iteration 1000	3.74	0.053921	0.144719	69.3607	25.8432
6	Grid size 10000 problem iteration 100000	44.71	27.9101	36.7607	1.6019	1.2162

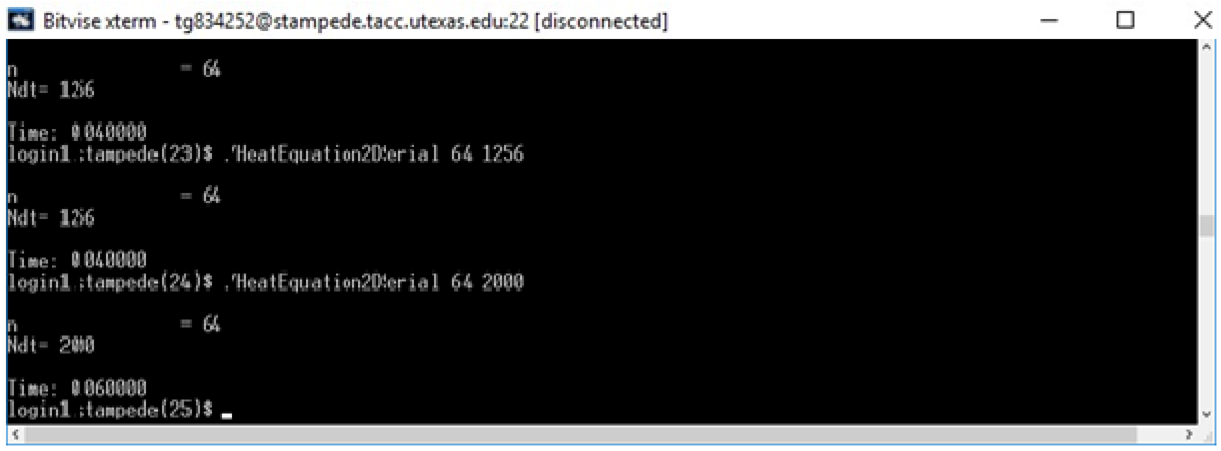
Thus from 2 & 3 when we keep problem size constant and half the number of iterations we see that the time to compute gets halved . in each case i. e. Serial & Parallel.

From 5 &3 we can say if we increase problem size keeping number of iterations constant the computing time increases.

A parallel code is efficient when its used for greater problem size. More the number of cores used more should be the problem size for better efficiency. Greater number of cores leads to more overhead hence loss efficiency hence when we need to have optimum number of cores for given problem size.

From 1 to 5 we can observe that when we run the program on 4 cores it shows more speedup than on 4 16 cores. Its because 16 cores will have more overhead (e.g startup time etc) But when we have greater problem size in 6 , we have more speedup for 16 processors than for 4 processors.

Example snapshot to run serial code:


Bitvise xterm - tg834252@stampede.tacc.utexas.edu:22 [disconnected]

```
n                 = 64
Ndt= 1256

Time: 0.040000
login1.stampede(23)$ ./HeatEquation2DSerial 64 1256

n                 = 64
Ndt= 1256

Time: 0.040000
login1.stampede(24)$ ./HeatEquation2DSerial 64 2000

n                 = 64
Ndt= 2000

Time: 0.060000
login1.stampede(25)$ _
```

Codes:

Serial Code :

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>



int main(int argc, char **argv)

{

        int Ndt, dt;                /*Ndt is number of steps of time to be carried out */

                                  /* dt is time intervaD */    double L, h,

x, y, k, alpha, beta;    /*all i/p parameters */      int N, n, i, j, ;

/* N=n^2 and is for original grid */

        double *a, *aNew, *aTemp;

FILE *outputFile;

        char outputFileName[] = "OutputFile.txt";

char outputFileNameStart[] = "OutputFileStart.txt";

clock_t begin, end;              double time_spent;



        L = 1; /* dimmension of the problem */

k = 0.00001; /* dt */



        if (argc != 3) {

                fprintf(stderr, "Usage: n Ndt\n");

                exit(0);

        }



 n = atoi(argv[1]);              /* Reading square grid (row, coumn) dimension and number of time steps from command
Line i.e. n & Ndt*/
```

```
        Ndt = atoi(argv[2]);

        h               = L / (n - 1);          /*input formulas as given */
        alpha      = k / (h*h); beta      = 1 - 4 * alpha;

        printf("\nn          = %d\n", n);
        printf("Ndt= %d\n", Ndt);

        N = n*n; //size of matrix n x n

        a = (double *)malloc(N * sizeof(double));          /*space allocation */
aNew = (double *)malloc(N * sizeof(double));

        for (i = 0; i < n; i++) {                     /* define A*/
x = i*h;
                for (j = 0; j < n; j++)
                {
y = j*h;
                        a[i * n + j] = 100 * x*y*(L - x)*(L - y);
                aNew[i * n + j] = a[i * n + j];

                }
        }

        outputFile = fopen(outputFileNameStart, "w");
if (outputFile == NULL)

        {
                printf("Error opening file!\n");
                exit(1);
        }

        for (i = 0; i < n; i++) {
```

```
                    for (j = 0; j < n; j++)
fprintf(outputFile, "%10.6f ", a[i * n + j]);          fprintf(outputFile,
"\n");

          }
          fprintf(outputFile, "\n"); fclose(outputFile);


          begin = clock();


          dt = 0;
          while (dt < Ndt)
          {
                    for (i = 1; i < n-1; i++)
          for (j = 1; j < n-1; j++)

                              {
  aNew[i*n+j] = beta*a[i*n + j] + alpha*(a[(i - 1)*n + j] + a[(i + 1)*n + j] + a[i*n + j - 1] + a[i*n + j + 1]);  /*5 point avg
updation*/
                              }


                    dt++;


                    aTemp = a;                /*update for next iteration */
                    a = aNew;
          aNew = aTemp;

          }


          end = clock();
          time_spent = (double)(end - begin) / CLOCKS_PER_SEC;          /*print time */


          printf("\nTime: %f\n", time_spent);


          outputFile = fopen(outputFileName, "w");
if (outputFile == NULL)
```

```
        {
                printf("Error opening file!\n");
                exit(1);
        }

        for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++)
                        fprintf(outputFile, "%10.6f ", a[i * n + j]);
                fprintf(outputFile, "\n");
        }
        fprintf(outputFile, "\n");

        fclose(outputFile);

        return 0;
}
```

Code using MPI

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

#include <string.h>

int main(int argc, char **argv)
{
        int Ndt, dt;                    /*Ndt is number of steps of time to be carried out */
                                       /* dt is time intervaD */
double L, h, x, y, k, alpha, beta;     /*all i/p parameters */

        int   numtasks, taskid;
 int N, n, i, j, ia, ja, nforDB, NforDB;       /* N=n^2 and is for original grid  , NforDB=nforDB^2 and is for distributed blocks
*/
        double *a, *ABlock, *ABlockNew, *ALocalTemp;    /*variables for local distributed blocks created*/
        double sum;
```

```
                double start_time, end_time;

                MPI_Datatype blockTypeGlobal, blockTypeLocal, rowType, columnType;

int *disps, dims[2], periods[2];

                int proc, proc1, proc2, proc3;

                MPI_Comm comm2d;

                int my2drank;

                int numberOfBlocks, rowsize; int

                MyCoords[2], coords[2];

 MPI_Status status;  int row =

0, col = 1;  FILE *outputFile;

                char outFileName[] = "OutputFile.txt";

char outputFileTime[] = "OutputFileTime.txt";



                L = 1;

                k = 0.00001;


                MPI_Init(&argc, &argv);

                MPI_Comm_rank(MPI_COMM_WORLD, &taskid);


                if (argc < 3) {

if (taskid == 0) {

                                fprintf(stderr, "Usage: n Ndt\n");

                        }

                        MPI_Finalize();

exit(0);

                }


 n = atoi(argv[1]);             /* Reading square grid dimension and number of time steps from command Line i.e. n &
Ndt*/

                Ndt = atoi(argv[2]);


/*input formulas as given */
```

```
                h = L / (n - 1);
alpha = k / (h*h);    beta = 1 -
4 * alpha; if (taskid == 0) {

                        printf("\nn             = %d\n", n);
                        printf("Ndt= %d\n", Ndt);
                }


                MPI_Comm_size(MPI_COMM_WORLD, &numtasks);


 dims[row] = dims[col] = (int)sqrt(numtasks);  /*no. of rows & Columns equal to tasks to be performed in each iteration i.e. n */


                periods[row] = periods[col] = 0;          /*setting false i.e. grid is non periodic in both row & column */


                MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 1, &comm2d);
                MPI_Comm_rank(comm2d, &my2drank);
                MPI_Cart_coords(comm2d, my2drank, 2, MyCoords);


                N = n*n; //size of matrix n x n
numberOfBlocks = (int)sqrt(numtasks);
nforDB = n / numberOfBlocks;            NforDB =
nforDB*nforDB;    rowsize =
numberOfBlocks*NforDB;



                if (my2drank == 0) {
                        a = (double *)malloc(N * sizeof(double));


                        disps = (int *)malloc(numtasks * sizeof(int));


                        for (i = 0; i < n; i++) {
x = i*h;
                                for (j = 0; j < n; j++)
                                {
y = j*h;
```

```
                                        a[i * n + j] = 100 * x*y*(L - x)*(L - y);

                              }
                    }


                    for (i = 0; i < numberOfBlocks; i++)

                    for (j = 0; j < numberOfBlocks; j++) {

                                        coords[row] = i;

                                        coords[col] = j;

                                        MPI_Cart_rank(comm2d, coords, &proc);

                    disps[proc] = i*rowsize + j*nforDB;

                              }

          }


 ABlock = (double *)calloc((nforDB + 2)*(nforDB + 2), sizeof(double));                /* space allocation for grid and block
storage */

          ABlockNew = (double *)calloc((nforDB + 2)*(nforDB + 2), sizeof(double));


          MPI_Type_vector(nforDB, nforDB, n, MPI_DOUBLE, &blockTypeGlobal);

          MPI_Type_commit(&blockTypeGlobal);


          MPI_Type_vector(nforDB, nforDB, nforDB + 2, MPI_DOUBLE, &blockTypeLocal);

          MPI_Type_commit(&blockTypeLocal);


          MPI_Type_contiguous(nforDB, MPI_DOUBLE, &rowType);

          MPI_Type_commit(&rowType);


          MPI_Type_vector(nforDB, 1, (nforDB + 2), MPI_DOUBLE, &columnType);

          MPI_Type_commit(&columnType);


 /* block division into right, left, top, bottom corners & middle / centre and their respective calculations and message passing for
next iterations */


          if (my2drank == 0) {                    for (i =

0; i < nforDB; i++)                              for (j =

0; j < nforDB; j++)
```

```
                        {

                                AB1ock[(i + 1)*(nforDB + 2) + j + 1] = a[i*n + j];
                                ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = ABlock[(i + 1)*(nforDB + 2) + j + 1];

                        }


                for (i = 1; i < numtasks; i++)

                        MPI_Send(a + disps[i], 1, blockTypeGlobal, i, 1, comm2d);

        }


        if (my2drank != 0) {

                MPI_Recv(ABlock + nforDB + 3, 1, blockTypeLocal, 0, 1, comm2d, &status);

        for (i = 0; i < nforDB; i++)                    for (j = 0; j < nforDB; j++)

                                ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = ABlock[(i + 1)*(nforDB + 2) + j + 1];

        }


        MPI_Barrier(comm2d);


        if (my2drank == 0)

start_time = MPI_Wtime();



        dt = 0;

        while (dt < Ndt)

        {

                if ((MyCoords[0] == 0) && (MyCoords[1] == 0))

                {

                        coords[row] = MyCoords[row] + 1;        /*data exchanges from bottom corners */

                        coords[col] = MyCoords[col];

                        MPI_Cart_rank(comm2d, coords, &proc);

                        MPI_Send(ABlock + (nforDB)*(nforDB + 2) + 1, 1, rowType, proc, 1, comm2d);


                        MPI_Recv(ABlock + (nforDB + 1)*(nforDB + 2) + 1, 1, rowType, proc, 1, comm2d, &status);




                        coords[row] = MyCoords[row];                    /* Right  side corners*/
```

```
                              coords[col] = MyCoords[col] + 1;
                              MPI_Cart_rank(comm2d, coords, &proc2);


                                MPI_Send(ABlock + (nforDB + 2) + nforDB + 0, 1, columnType, proc2, 0, comm2d);

                               MPI_Recv(ABlock + (nforDB + 2) + nforDB + 1, 1, columnType, proc2, 0, comm2d, &status);


for (i = 1; i < nforDB; i++)                 /* calculating local new values & storing inputs/updates for next iteration */

                                          for (j = 1; j < nforDB; j++)

                                          {

                                                sum = beta*ABlock[(i + 1)*(nforDB + 2) + j + 1];

                                                ia = i - 1; ja = j;

                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];

                                                ia = i + 1; ja = j;

                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];

                                                ia = i; ja = j - 1;

                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];

                                                ia = i; ja = j + 1;

                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];


                                                ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = sum;

                                          }

                        }

                        else if ((MyCoords[0] == 0) && (MyCoords[1] == numberOfBlocks - 1))

                        {

coords[row] = MyCoords[row] + 1;    coords[col] = MyCoords[col];

                              MPI_Cart_rank(comm2d, coords, &proc);


                              MPI_Send(ABlock + (nforDB)*(nforDB + 2) + 1, 1, rowType, proc, 1, comm2d);

MPI_Recv(ABlock + (nforDB + 1)*(nforDB + 2) + 1, 1, rowType, proc, 1, comm2d, &status);


coords[row] = MyCoords[row];    coords[col] = MyCoords[col] - 1;

                              MPI_Cart_rank(comm2d, coords, &proc3);
                              MPI_Send(ABlock + (nforDB + 2) + 1, 1, columnType, proc3, 0, comm2d);

                        MPI_Recv(ABlock + (nforDB + 2) + 0, 1, columnType, proc3, 0, comm2d, &status);
```

```
                    for (i = 1; i < nforDB; i++)
            for (j = 0; j < nforDB - 1; j++)

                            {
                                    sum = beta*ABlock[(i + 1)*(nforDB + 2) + j + 1];
                                    ia = i - 1; ja = j;
                                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                    ia = i + 1; ja = j;
                                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                    ia = i; ja = j - 1;
                                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                    ia = i; ja = j + 1;
                                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];


                                    ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = sum;
                            }


            }
            else if ((MyCoords[0] == numberOfBlocks - 1) && (MyCoords[1] == 0))
            {
                    coords[row] = MyCoords[row] - 1;
    coords[col] = MyCoords[col];

                    MPI_Cart_rank(comm2d, coords, &proc);


                    MPI_Send(ABlock + (nforDB + 2) + 1, 1, rowType, proc, 1, comm2d);
    MPI_Recv(ABlock + 1, 1, rowType, proc, 1, comm2d, &status);



     coords[row] = MyCoords[row];   coords[col] =
    MyCoords[col] + 1;

                    MPI_Cart_rank(comm2d, coords, &proc2);
                    MPI_Send(ABlock + (nforDB + 2) + nforDB + 0, 1, columnType, proc2, 0, comm2d);
            MPI_Recv(ABlock + (nforDB + 2) + nforDB + 1, 1, columnType, proc2, 0, comm2d, &status);
```

```
                for (i = 0; i < nforDB - 1; i++)
                        for (j = 1; j < nforDB; j++)

                        {
                                sum = beta*ABlock[(i + 1)*(nforDB + 2) + j + 1];
                                ia = i - 1; ja = j;
                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                ia = i + 1; ja = j;
                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                ia = i; ja = j - 1;
                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                ia = i; ja = j + 1;
                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];


                                ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = sum;
                        }
        }
        else if ((MyCoords[0] == numberOfBlocks - 1) && (MyCoords[1] == numberOfBlocks - 1))
        {
coords[row] = MyCoords[row] - 1;    coords[col] = MyCoords[col];

                MPI_Cart_rank(comm2d, coords, &proc);
                MPI_Recv(ABlock + 1, 1, rowType, proc, 1, comm2d, &status);
                MPI_Send(ABlock + (nforDB + 2) + 1, 1, rowType, proc, 1, comm2d);


                /* Left */
    coords[row] = MyCoords[row];   coords[col] =
   MyCoords[col] - 1;

                MPI_Cart_rank(comm2d, coords, &proc3);


                MPI_Send(ABlock + (nforDB + 2) + 1, 1, columnType, proc3, 0, comm2d);
                MPI_Recv(ABlock + (nforDB + 2) + 0, 1, columnType, proc3, 0, comm2d, &status);
                for (i = 0; i < nforDB - 1; i++)
                        for (j = 0; j < nforDB - 1; j++)
                        {
```

```
                                                sum = beta*ABlock[(i + 1)*(nforDB + 2) + j + 1];

                                                ia = i - 1; ja = j;

                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];

                                                ia = i + 1; ja = j;

                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];

                                                ia = i; ja = j - 1;

                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];

                                                ia = i; ja = j + 1;

                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];


                                                ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = sum;

                                        }

                        }

                        else if ((MyCoords[0] == 0) && (MyCoords[1] > 0))

                        {

coords[row] = MyCoords[row] + 1;    coords[col] = MyCoords[col];

                                MPI_Cart_rank(comm2d, coords, &proc);

                                MPI_Send(ABlock + (nforDB)*(nforDB + 2) + 1, 1, rowType, proc, 1, comm2d);

MPI_Recv(ABlock + (nforDB + 1)*(nforDB + 2) + 1, 1, rowType, proc, 1, comm2d, &status);


coords[row] = MyCoords[row];    coords[col] = MyCoords[col] - 1;

                                MPI_Cart_rank(comm2d, coords, &proc3);


                                MPI_Send(ABlock + (nforDB + 2) + 1, 1, columnType, proc3, 0, comm2d);

                        MPI_Recv(ABlock + (nforDB + 2) + 0, 1, columnType, proc3, 0, comm2d, &status);



      coords[row] = MyCoords[row];   coords[col] = MyCoords[col]

     + 1; MPI_Cart_rank(comm2d, coords, &proc2);



                                MPI_Send(ABlock + (nforDB + 2) + nforDB + 0, 1, columnType, proc2, 0, comm2d);

                        MPI_Recv(ABlock + (nforDB + 2) + nforDB + 1, 1, columnType, proc2, 0, comm2d, &status);
                                for (i = 1; i < nforDB; i++)
                                        for (j = 0; j < nforDB; j++)
                                        {
                                                sum = beta*ABlock[(i + 1)*(nforDB + 2) + j + 1];
                                                ia = i - 1; ja = j;
                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                                ia = i + 1; ja = j;
                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                                ia = i; ja = j - 1;
                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                                ia = i; ja = j + 1;
                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];


                                                ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = sum;
                                        }
                        }
                        else if ((MyCoords[0] == numberOfBlocks - 1) && (MyCoords[1] > 0))
                        {
coords[row] = MyCoords[row] - 1;    coords[col] = MyCoords[col];

                                MPI_Cart_rank(comm2d, coords, &proc);
                                MPI_Recv(ABlock + 1, 1, rowType, proc, 1, comm2d, &status);
                                MPI_Send(ABlock + (nforDB + 2) + 1, 1, rowType, proc, 1, comm2d);


                                /* Left (n,1) */
                        coords[row] = MyCoords[row];
                        coords[col] = MyCoords[col] - 1;

                                MPI_Cart_rank(comm2d, coords, &proc3);
                                MPI_Send(ABlock + (nforDB + 2) + 1, 1, columnType, proc3, 0, comm2d);
                                MPI_Recv(ABlock + (nforDB + 2) + 0, 1, columnType, proc3, 0, comm2d, &status);


                         /*(1,n)    right    */         coords[row]    =
                        MyCoords[row];              coords[col]      =
                        MyCoords[col] + 1;
```

```
                            MPI_Cart_rank(comm2d, coords, &proc2);

                            MPI_Send(ABlock + (nforDB + 2) + nforDB + 0, 1, columnType, proc2, 0, comm2d);
MPI_Recv(ABlock + (nforDB + 2) + nforDB + 1, 1, columnType, proc2, 0, comm2d, &status);

                            for (i = 0; i < nforDB - 1; i++)
                            for (j = 0; j < nforDB; j++)
                                      {
                                                sum = beta*ABlock[(i + 1)*(nforDB + 2) + j + 1];
                                                ia = i - 1; ja = j;
                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                                ia = i + 1; ja = j;
                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                                ia = i; ja = j - 1;
                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                                ia = i; ja = j + 1;
                                                sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];

                                                ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = sum;
                                      }
                  }
                  else if ((MyCoords[0] > 0) && (MyCoords[1] == 0))
                  {
         /* bottom leftmost */   coords[row] = MyCoords[row]
        - 1;   coords[col] = MyCoords[col];

                            MPI_Cart_rank(comm2d, coords, &proc);
                            MPI_Send(ABlock + (nforDB + 2) + 1, 1, rowType, proc, 1, comm2d);

                            MPI_Recv(ABlock + 1, 1, rowType, proc, 1, comm2d, &status);

                            /*topmost left*/
                   coords[row] = MyCoords[row] + 1;  coords[col] =
                  MyCoords[col];

                            MPI_Cart_rank(comm2d, coords, &proc1);
                            MPI_Send(ABlock + nforDB*(nforDB + 2) + 1, 1, rowType, proc1, 1, comm2d);
MPI_Recv(ABlock + (nforDB + 1)*(nforDB + 2) + 1, 1, rowType, proc1, 1, comm2d, &status);


                            /* Right */
coords[row] = MyCoords[row];    coords[col] = MyCoords[col] + 1;

                            MPI_Cart_rank(comm2d, coords, &proc2);


                            MPI_Send(ABlock + (nforDB + 2) + nforDB + 0, 1, columnType, proc2, 0, comm2d);
MPI_Recv(ABlock + (nforDB + 2) + nforDB + 1, 1, columnType, proc2, 0, comm2d, &status);


                            for (i = 0; i < nforDB; i++)
                  for (j = 1; j < nforDB; j++)

                                 {
                                           sum = beta*ABlock[(i + 1)*(nforDB + 2) + j + 1];
                                           ia = i - 1; ja = j;
                                           sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                           ia = i + 1; ja = j;
                                           sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                           ia = i; ja = j - 1;
                                           sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                           ia = i; ja = j + 1;
                                           sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];


                                           ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = sum;
                                 }
                  }
                  else if ((MyCoords[0] > 0) && (MyCoords[1] == numberOfBlocks - 1))
                  {
                   coords[row] = MyCoords[row] - 1;  coords[col] =
                  MyCoords[col];

                            MPI_Cart_rank(comm2d, coords, &proc);
```

```
                                    MPI_Send(ABlock + (nforDB + 2) + 1, 1, rowType, proc, 1, comm2d);
                                    MPI_Recv(ABlock + 1, 1, rowType, proc, 1, comm2d, &status);

coords[row] = MyCoords[row] + 1;    coords[col] = MyCoords[col];
                                    MPI_Cart_rank(comm2d, coords, &proc1);

                                    MPI_Send(ABlock + nforDB*(nforDB + 2) + 1, 1, rowType, proc1, 1, comm2d);
MPI_Recv(ABlock + (nforDB + 1)*(nforDB + 2) + 1, 1, rowType, proc1, 1, comm2d, &status);

coords[row] = MyCoords[row];    coords[col] = MyCoords[col] - 1;
                                    MPI_Cart_rank(comm2d, coords, &proc3);

                                    MPI_Send(ABlock + (nforDB + 2) + 1, 1, columnType, proc3, 0, comm2d);
                                    MPI_Recv(ABlock + (nforDB + 2) + 0, 1, columnType, proc3, 0, comm2d, &status);

                                    for (i = 0; i < nforDB; i++)
      for (j = 0; j < nforDB - 1; j++)
                                            {
                                                    sum = beta*ABlock[(i + 1)*(nforDB + 2) + j + 1];
                                            ia = i - 1; ja = j;
                                                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                                    ia = i + 1; ja = j;
                                                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                                    ia = i; ja = j - 1;
                                                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                                                    ia = i; ja = j + 1;
                                                      sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];

                                                    ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = sum;
                                            }
                            }
                            else
                            {
```

```
                                        /*  rightmost bottom */
                coords[row] = MyCoords[row] - 1;
coords[col] = MyCoords[col];

                                        MPI_Cart_rank(comm2d, coords, &proc);


                                        MPI_Send(ABlock + (nforDB + 2) + 1, 1, rowType, proc, 1, comm2d);
                                        MPI_Recv(ABlock + 1, 1, rowType, proc, 1, comm2d, &status);


                                        /* rightmost top */
coords[row] = MyCoords[row] + 1;
coords[col] = MyCoords[col];

                                        MPI_Cart_rank(comm2d, coords, &proc1);


                                        MPI_Send(ABlock + nforDB*(nforDB + 2) + 1, 1, rowType, proc1, 1, comm2d);
  MPI_Recv(ABlock + (nforDB + 1)*(nforDB + 2) + 1, 1, rowType, proc1, 1, comm2d, &status);


  coords[row] = MyCoords[row];    coords[col] = MyCoords[col] - 1;

                                        MPI_Cart_rank(comm2d, coords, &proc3);


                                        MPI_Send(ABlock + (nforDB + 2) + 1, 1, columnType, proc3, 0, comm2d);
                                MPI_Recv(ABlock + (nforDB + 2) + 0, 1, columnType, proc3, 0, comm2d, &status);



                                        coords[row] = MyCoords[row];
                                        coords[col] = MyCoords[col] + 1;
                                        MPI_Cart_rank(comm2d, coords, &proc2);
                                        MPI_Send(ABlock + (nforDB + 2) + nforDB + 0, 1, columnType, proc2, 0, comm2d);
                                        MPI_Recv(ABlock + (nforDB + 2) + nforDB + 1, 1, columnType, proc2, 0, comm2d, &status);


                                        /* updating inputs for next iteration */
                                        for (i = 0; i < nforDB; i++)

                                                for (j = 0; j < nforDB; j++)
                                                {
}

                    sum = beta*ABlock[(i + 1)*(nforDB + 2) + j + 1];
                    ia = i - 1; ja = j;
                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                    ia = i + 1; ja = j;
                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                    ia = i; ja = j - 1;
                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];
                    ia = i; ja = j + 1;
                    sum += alpha*ABlock[(ia + 1)*(nforDB + 2) + ja + 1];

                    ABlockNew[(i + 1)*(nforDB + 2) + j + 1] = sum;
                }
            }

            MPI_Barrier(comm2d);

            ALocalTemp = ABlock;
            ABlock = ABlockNew;
            ABlockNew = ALocalTemp;

            dt++;
        }

        if (my2drank != 0) {
            MPI_Send(ABlock + nforDB + 3, 1, blockTypeLocal, 0, 1, comm2d);
        }
        if (my2drank == 0) {        for (i = 0; i <
        nforDB; i++)                for (j = 0; j <
        nforDB; j++)

                                    a[i*n + j] = ABlock[(i + 1)*(nforDB + 2) + j + 1];

            for (i = 1; i < numtasks; i++)
MPI_Recv(a + disps[i], 1, blockTypeGlobal, i, 1, comm2d, &status);
        }
        MPI_Barrier(comm2d);

        if (my2drank == 0)
end_time = MPI_Wtime();

        if (my2drank == 0)
                printf("Wallclock time: %f sec\n", end_time - start_time);

        if (my2drank == 0)
        {
                outputFile = fopen(outFileName, "w");
        if (outputFile == NULL)

                {
                        MPI_Finalize();
                        exit(1);
                }

                for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)

                                fprintf(outputFile, "%10.6f ", a[i * n + j]);
                fprintf(outputFile, "\n");

                }
                fprintf(outputFile, "\n");
                fclose(outputFile);

                outputFile = fopen(outputFileTime, "w");
                if (outputFile == NULL)

                {
                        exit(1);
                }
```

```
 fprintf(outputFile, "Time: %f", (double)(end_time - start_time));


                    fprintf(outputFile, "\n");
fclose(outputFile);

          }


          if (my2drank == 0)
printf("Stop\n");



          MPI_Finalize();


          return 0;
}
```