E517/317 – Introduction to High Performance Computing
Fall 2019

Assignment 5

Problem 1. Heat distribution.

Consider a square area that has known temperatures along each of its edges, and a single heat source at one boundary location. The temperature of the interior of an area will depend on the temperatures around it. A question that we would like to answer is:

"How does the distribution of the heat in the area change depending on whether the heat source is near an edge or near the center of the area, and also depending on the temperature of the heat source?"

The approach to solving this problem is to divide the area into a fine mesh of points, `h[i][j]`. Temperatures at an inside point are taken to be the average of temperatures of four neighboring points. The figure below shows the model of the area as a square mesh of points. One of the points is enlarged.

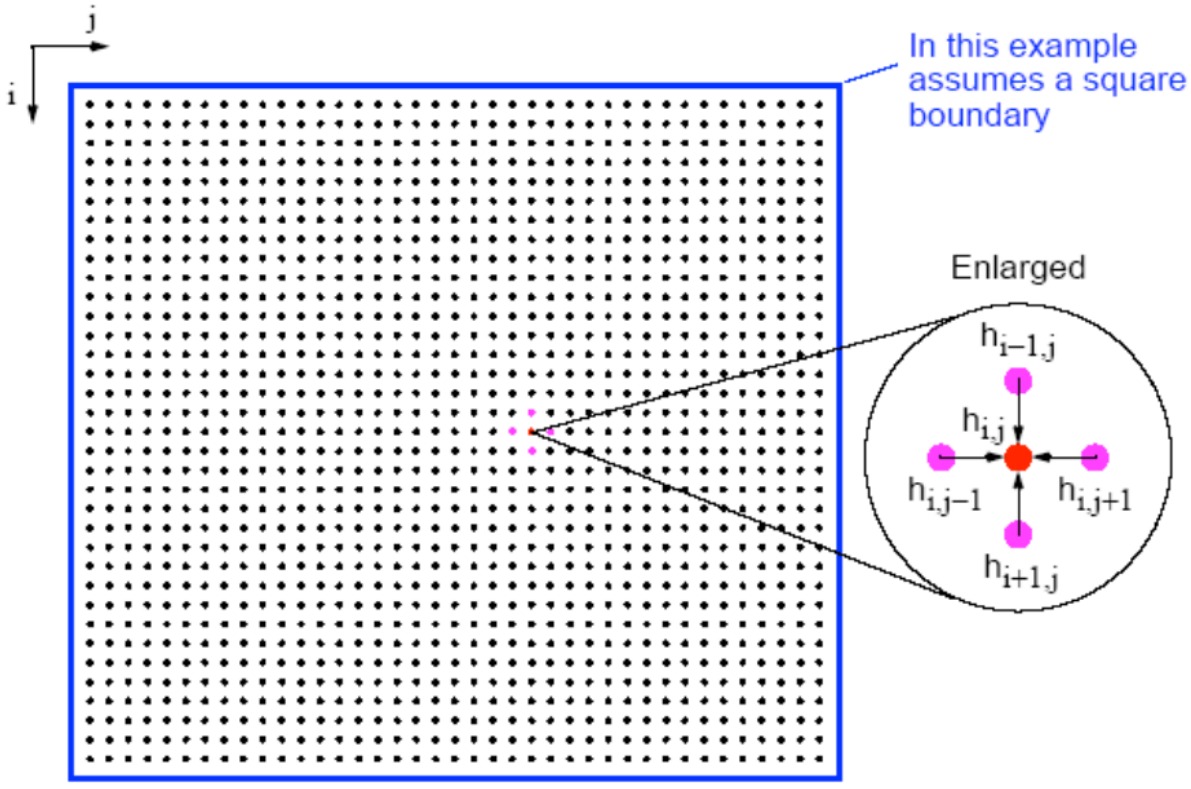

Figure 1: Mathematical Model of Area (Source: Wilkinson, 2004)

A serial implementation that solves this problem has been provided to you. The listing of the code is provided on the next page of this assignment, along with the original source code ("`heat_distribution.c`") available in canvas.

Consider the code snippet provided on the next page. The code operates as follows: two matrices, `a_new` and `a_old`, are allocated and initialized after which a while loop is executed until a desired precision, controlled by a variable `EPS`, is reached. Every iteration of the loop computes new values for the non-boundary points, calculates the maximum difference between all points of the old and new iteration, and proceeds to the next iteration.

This program should declare and compute a mesh of size `10000x10000`. The temperature of the edges should be fixed at `0` degrees Celsius. You should also initialize the interior of the mesh to `0` degrees Celsius, but these will change as the program executes.

The program should declare as input the `x` and `y` coordinates of the heat source (i.e. indices of the matrix), which is anywhere within the mesh or on the edge. The program should check to make sure that the entries are valid. If they are not valid then the program should terminate.

The program should also declare the temperature of the heat source. The location and temperature of the heat source should remain fixed throughout the entire program.

The program should terminate the computation after a certain precision of computation is reached. This should be declared within the code itself.

To summarize, your implementation should be done using MPI, be able to run on `1..64` cores of BigRed 2 with the following parameters:

`COLS=ROWS=10000, EPS=1e-3` ***for full credit***.

Submitting code that works with values less than those mentioned (e.g. your code only works for cases of ROWS=100, or EPS=1e-1) will result in lower grade.

```
40 int main() {
41
42     double **a_old = alloc_matrix(); //allocate memory for the matrices
43     double **a_new = alloc_matrix();
44
45     init_matrix(a_old); //initialize the matrices
46     init_matrix(a_new);
47
48     while (true) {
49
50         if (DEBUG)
51             printf("Performing a new iteration...\n");
52
53         //compute new values and put them into a_new
54         compute_new_values(a_old, a_new);
55
56         if (DEBUG) {
57             printf("a_old is:\n"); //output matrix to screen
58             print_matrix(a_old);
59
60             printf("a_new is:\n");
61             print_matrix(a_new);
62         }
63
64         //calculate the maximum absolute differences among pairwise
65         // differences of old and new matrix elements
66         double max_diff = max_abs(a_old, a_new);
67
68         if (DEBUG)
69             printf("Max diff is: %f\n", max_diff);
70
71         if (max_diff < EPS)
72             break;
73
74         copy_matrix(a_old, a_new); //assign values of a_new to a_old
75
76         if (DEBUG)
77             printf("End of iteration\n\n");
78     }
79
80     print_matrix(a_new);
81
82     return 0;
83 }
```

For the Problem statement above:

1. Generate an MPI program to perform the above 2-D Heat Distribution on 4 cores [50pts]
2. Provide a generic solution that works for a variable number of processors. [20pts]

3. Provide scaling results for 1..64 cores on BigRed 2. Only do this after successfully completing item 2. [10pts]
4. Using your favorite visualization software, visualize the resulting heat distribution (you might want to use gnuplot, matlab or similar) [10pts]
5. Document the code and provide details in the report. Be succinct and precise [10 pts]

Hints:

a) If you are using gnu compilers (on your own linux machine or outside bigred 2) and encounter an error like this " error: 'for' loop initial declaration used outside C99 mode", use the '–std=c99' flag. If you are writing/debugging your code on bigred2, cc works just fine.
b) Use small values for ROWS and COLS when debugging your parallel code. ROWS=COLS=16 would probably be a good case as you can easily split your domain between 4 processes to begin with. Remember though that the final solution should work with 10Kx10K matrix.
c) For problem 2, when thinking of a generic solution with variable number of cores, think of a way to distribute data for any number of processes. Also, think what kind of communication is required on the boundaries of subdomains allocated to different processes (i.e. residing in distributed memory).
d) Defining custom MPI datatypes might simplify your job of communicating information on the boundaries of subdomains. This is not a requirement, however.