INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

# Spring Semester, 2020
# Introduction to Parallel Scientific Computing
# Course code: CS504

**HW III**
February 21, 2020

Due: 12.03.20 **Instructor: Dr. Pawan Kumar** Maximum Marks: 47

**INSTRUCTIONS:**

**The codes must be written in either C or C++.**

1. **(OpenMP, MPI and CUDA)** The objective is to model the static heat distribution of a room. We will simulate the room in 2 dimensions. The room is 10 feet wide and 10 feet long with a fireplace along one wall as depicted in Figure 1. The fireplace is 4 feet wide and is centered along one wall (it takes up 40% of the wall, with 30% of the walls on either side). The fireplace emits 100 C of heat (although in reality a fire is much hotter). The walls are considered to be 20 C. The boundary values (the fireplace and the walls) are considered to be fixed temperature. We can find the temperature distribution by dividing the area into a fine mesh of points, $h_{i,j}$. The temperature at an inside point can be taken average of the temperatures of the four neighboring points, as illustrated in Figure 2. [4+4+4+2]

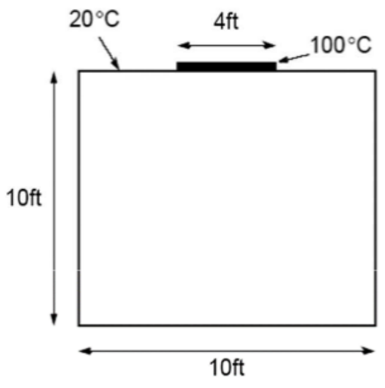


*Figure 1: 10 x 10 room with a fireplace*

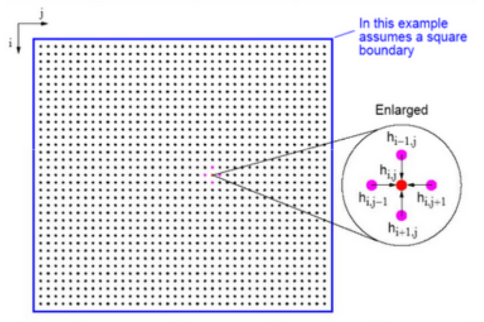


*Figure 2: Determining Heat Distribution by a finite difference method*

We can compute the temperature of each point by iterating the equation:

$$h_{i,j} = \frac{h_{i-1,j} + h_{i+1,j} + h_{i,j-1} + h_{i,j+1}}{4}$$

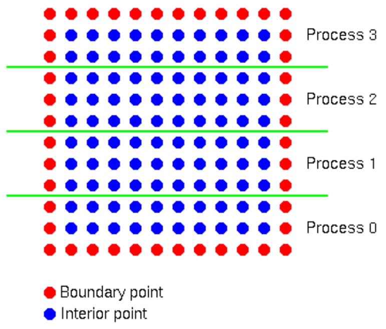
Suppose the temperature of each point is held in an array $h[i][j]$, and the boundary points $h[0][x], h[x][0], h[n][x]$, and $h[x][n]$ $(0 \times n)$ have been initialized to the edge temperatures. The calculation as sequential code could be

```
for (iteration = 0; iteration < limit; iteration++) {
   for (i = 1; i < n; i++)
      for (j = 1; j < n; j++)
         g[i][j] = 0.25 * (h[i-1][j] + h[i+1][j] + h[i][j-1] + h[i][j+1]);

   for (i = 1; i < n; i++)      /* update points */
      for (j = 1; j < n; j++)
         h[i][j] = g[i][j];
}
```

Do the following tasks:

1. Write an OpenMP program that will model this static heat distribution of the room. Model the room as $1000 \times 1000$ matrix and consequently, the fireplace of 400 points.
2. Write an MPI program that will model this static heat distribution of the room. [Hint: Decompose the problem in one-dimension by dividing processes as shown in Figure]



3. Write a CUDA program that will model this static heat distribution of the room.
4. Compare the results and times of the above OpenMP, MPI and CUDA programs, and make a table for each of these showing time in seconds versus the number of processes (for MPI) or threads used (for OpenMP and CUDA).

2. **(OpenMP)** Describe and correct what is wrong with the following code when run with OMP_NUM_THREADS = 4 if the output is expected to match the serial program output [3]

```
main(...){

   int i,j,k,l;
   #pragma omp parallel
   {
   #pragma omp for collapse(3)
   for (i = 0 ; i < 100 ; i++)
      for (j = 0 ; j < 100 ; j++)
         for (k = 10 ; k < 100 ; k++)
            for (l = 10 ; l < 100 ; l++)
               arr[i][j][k][l] = i + (j*2) + arr [i][j][k-10][l];
   }

}
```

3. **(OpenMP, MPI and CUDA)** Implement Bitonic Sort using OpenMP and OpenMPI and CUDA. Compare the serial and parallel implementation. Following is pseudo code - [4+4+4+2]

```
void bitonic_sort(int start, int end, int *arr){
    int no_of_elements = start-end+1;
    for(int j = 2; j <= no_of_elements ; j = j * 2){
        for(int i = 0; i < no_of_elements ; i = i + j){
            if( ( i / j ) % 2 == 0) merge( i , i + j - 1 , 1 , arr);
            else merge( i , i + j - 1 , 0 , arr);
}

void merge(int start, int end, int dir, int *arr){
    int no_of_elements = start-end+1;
    for(int j = no_of_elements/2; j > 0 ; j = j / 2)
        for(int i = start; i + j <= last ; i++)
            if(dir == (arr[i] > arr[i+j]) swap( arr[i], arr[i+j]);
}
```

Your code should take $N$ (no of elements to sort) as input and $M$ (no of threads in case of OpenMP and no of processes in case of OpenMPI) as input, generate an array of $N$ random integers and sort them using parallel as well as serial implemented code and check the correctness. [Hint: For an array of size != $2^n$ , you can pad 0s]

4. **(OpenMP or MPI)** Solve $Ax = b$ for a $n \times n$ dense matrix using LU factorization. Check your solution by computing $||b - Ax||_2$, it should be close to zero (in double precision arithmetic). Implement using OpenMP or OpenMPI.Compare the serial and parallel implementation. [4+2]

   Your code should take $N$ (dimension of $A$) as input and $M$ (no of threads in case of OpenMP and no of processes in case of OpenMPI) as input, and generate a random matrix $A$ and vector $b$, solve them using parallel and serial implemented code and check for correctness.

5. **(CUDA)** Write a CUDA program that, given an $N$-element vector, finds the [1+1+2+2]

   1. The maximum element in the vector
   2. The minimum element in the vector
   3. The arithmetic mean of the vector
   4. The standard deviation of the values in the vector

   Your solution should take as input $N = 10^8$ and generate a randomized vector $V$ of length $N$. It should then compute the above four values of $V$ on the CPU and on the GPU. The program should output the computed values as well as the time taken to find each value.
   [Hint: Rather than creating one kernel for each of the statistics, try to compute as many of the statistics as possible in a single kernel. Computing the statistics concurrently can significantly improve performance, because the overhead of launching a kernel is non-negligible]

6. **(CUDA)** Write a CUDA program to detect edges in an image by applying the method as follows - [4]

   Assuming that the image to be operated is $I$:

   1. We calculate two derivatives:

Question 6 continues on the next page. . .

(a) **Horizontal changes:** This is computed by convolving $I$ with a kernel $G_x$ with odd size. For example for a kernel size of 3, $G_x$ would be computed as:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

(b) **Vertical changes:** This is computed by convolving $I$ with a kernel $G_y$ with odd size. For example for a kernel size of 3, $G_y$ would be computed as:

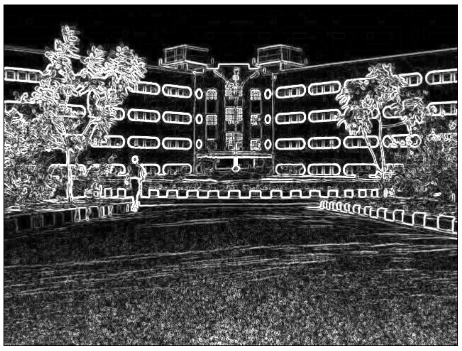
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

2. At each point of the image we calculate an approximation of the gradient in that point by combining both results above:

$$G = \sqrt{G_x^2 + G_y^2}$$

Consider the following input image:

Here is the output of applying our basic detector to the image obh.jpg:



You may look at the serial implementation in the edgeDetection.cpp file
Follow the sequence of steps

1. Reserve one of the compute nodes in ADA as follows:
`sinteractive -c 10`

Question 6 continues on the next page. . .

2. Load the opencv module as follows
   `module load opencv/4.1.2`
3. To compile do the following:
   ```
   g++ `pkg-config --cflags opencv` edgeDetection.cpp `pkg-config --libs opencv`
   -o edgeDetection
   ```
   See the link here: `https://stackoverflow.com/questions/13904117/compiling-and-linking-openc`
4. To run, do the following:
   `./edgeDetection obh.jpg`