ECE 588 Advance Computer Architecture 2

Sukrut Kelkar

Language used: C

Parallel programs using POSIX threads (Pthreads)

Pi Calculation:

Inscribed a circle in a square, randomly generate points in the square, determined the number of points in the square that are also in the circle, r are the number of points in the circle divided by the number of points in the square. PI = 4.0*circle_count/npoints. Circle count is number of points inside the circle and npoint is the number of sample points considered.

More the points generated, the better the approximation.

A function inside the code checks if a particular sample point is inside the circle. If it is inside the circle, circle_count is incremented. Pi is calculated depending on the total points in the circle and the total number of samples. The number of sample points are divided depending upon the number of processors used for computation. Depending on this number i.e. total points divided by the number of processors, multiple threads are created of the function mentioned above and parallel computation of a range of sample points is calculated. Each thread has mutex lock which locks the thread after it's done and adds the circle count to the global circle count. The circle count computed for all the sample points is used to calculate Pi at the end.

Speed up for 1 to 16 processors is calculated and stored in output2.txt

How to run?

On Linux command line:

```
cc -lpthread –lrt piCal.c -o PiCal
```

```
./PiCal 1000000 16        (./PiCal <Max points> <# of Proc>)
```

Heat Equation Calculations:

Grid size = 1000 x 1000, spanning all points in the square between coordinates (1, 1) and (1000, 1000).

Initial condition: All center points in the region (200, 200) to (800, 800) have a temperature of 500 degrees, and all other points have a temperature of zero. Points outside the grid (i.e., neighbors of points on the boundary) always have a temperature of zero that does not change.

At each time step t, the temperature of a point at coordinates (x, y) is computed from the temperatures of the neighboring points in the previous time step (t-1) according to the following equation:

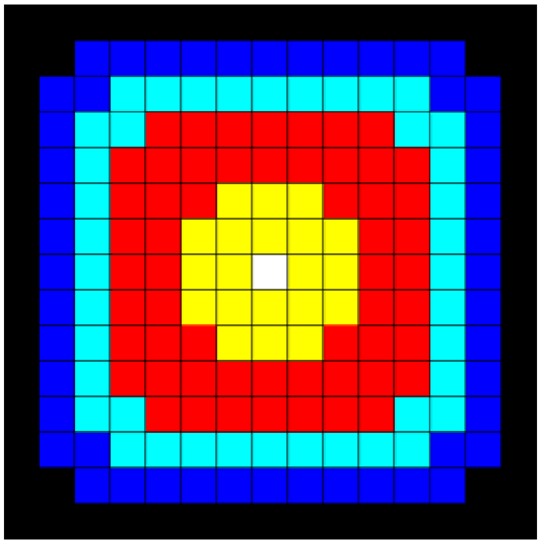
```
u2(ix, iy) =  u1(ix, iy)  +
    cx * (u1(ix+1,iy) + u1(ix-1,iy) - 2.*u1(ix,iy)) +
    cy * (u1(ix,iy+1) + u1(ix,iy-1) - 2.*u1(ix,iy))
```

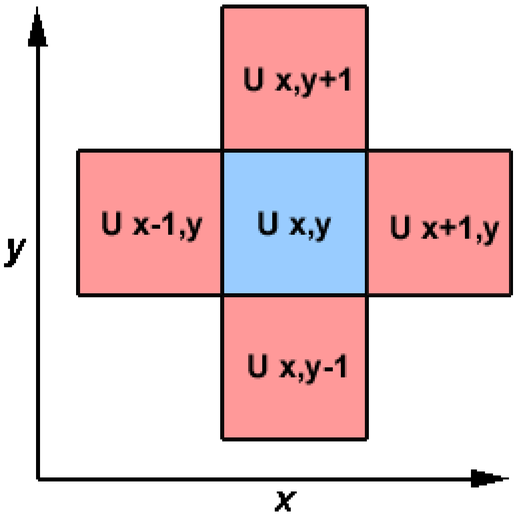
Where Cx =0.12 and Cy =0.1

The program is run for 6000 time steps. After each 200 time steps, the temperatures of the following points are printed: (1, 1), (150,150), (400, 400), (500, 500), (750, 750), and (900,900).

The algorithm is parallelized for computation of the matrix for an individual time step and then run for 6000 time steps to get a high parallel speedup.

The heat equation describes the temperature change over time, given initial temperature distribution and boundary conditions. A finite differencing scheme is employed to solve the heat equation numerically on a square region. The elements of a 2-dimensional array represent the temperature at points on the square. The initial temperature is zero on the boundaries and high in the middle. The boundary temperature is held at zero. A time stepping algorithm is used. The calculation of an element is dependent upon neighbor element values.





Speed up for 1 to 16 processors is calculated and stored in speedup_SK.txt

How to run?

On Linux command line:

```
cc -lpthread –lrt HeatCal.c -o HeatCal
```

```
./HeatCal 16      (./HeatCal <# of Proc>)
```

Or a script file is also provided which will calculated the time for 1 to 16 processors and create 16 text files. It can be ran as follows:

```
Source ScriptHeat.sh
```

Reference:

[1] Professor Alaa R. Alameldeen documents

[2] https://computing.llnl.gov/tutorials/parallel_comp/

[3] https://computing.llnl.gov/tutorials/pthreads/