PARALLELIZATION OF THE STEADY STATE HEAT EQUATION

CS 301 : High Performance Computing

Abhin Kakkad	201501419
Manthan Mehta	201501420

Brief Description of the Problem

- The problem involves the parallelization of the Heat Diffusion in 2D Square Plate.
- Method used is central average method.
- Finding out the steady state of a system is useful in physical as well as chemical processes.
- E.g. Finding out the steady state in chemical reactions is a very important part of the process because we need to know at what point will the system come to a stable state. Here, we have tried to find out the stable state of a 2-dimensional plate using the steady state equation.

Complexity of the Algorithm

- Considering the plate has dimensions M x N.
- The time required to traverse all the cells of a plate would be of the order O(M*N).
- The boundary points don’t need to be calculated since they have a fixed temperature, but if we take a very large plate then those points are very few in number so the time complexity for determining the temperature for all points would be O(M*N).
- For determining the temperature of all points we need to solve a second order differential equation which we would do so using the finite difference method.

Real World Applications

1) **Thermal Diffusivity in Polymers**
2) **Modeling of Options**
3) **Particle Diffusion**
4) **Other Applications**
   a) Pressure diffusion in a porous medium.
   b) Image analysis and in machine-learning as the driving theory behind Laplacian methods.
   c) An abstract form of heat equation provides a major approach to Atiyah-Singer index theorem and has led to much further work on heat equations in Riemannian geometry.

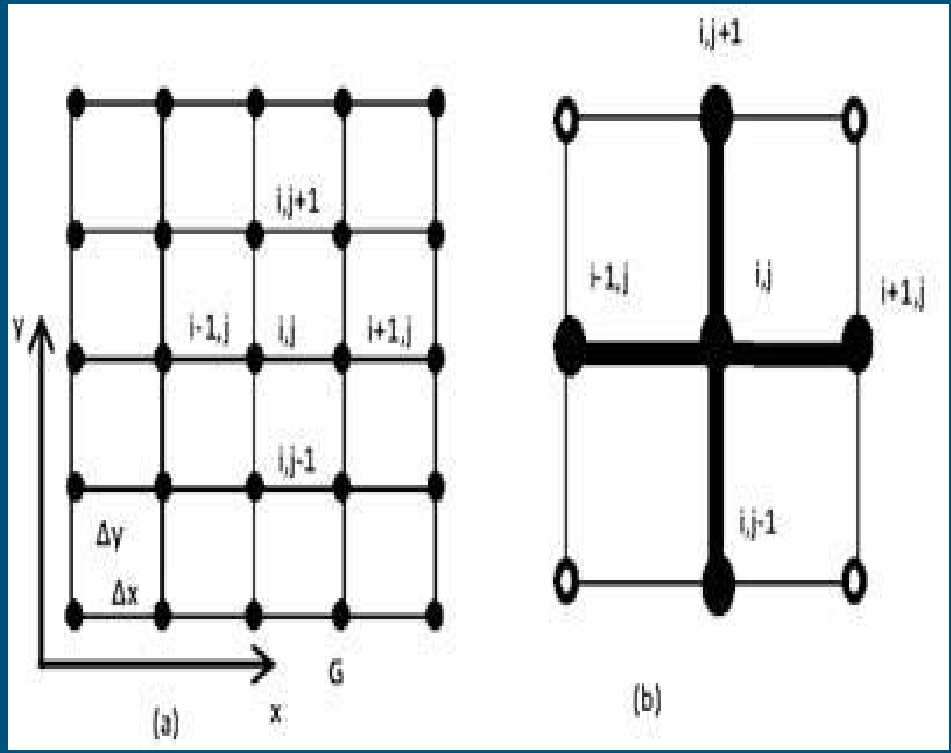
Algorithm

- Initially the plate has a certain temperature on three of its edges. Thus we have an initial condition set for the plate, also we would have the boundary conditions for the plate. This will provide us with a definite average temperature for the interior points of the plate.
- For each of the interior points there would be nine neighbouring points except for the boundary points. For calculating the temperature of any point we calculate the average of the four points around it, i.e. the points to its north, south, east and west. The method used here is the central mean theorem.

$$W[Central] = (1/4) * (W[North] + W[South] + W[East] + W[West])$$

Where W[i,j] is the temperature at (i,j) th point in the M*N grid.

- For every iteration we then update the value of the temperature values at that point. We keep increasing the iterations until the temperature difference in the current iteration and the previous iteration isn't less than or equal to the error tolerance that was set by us.
- When such a point is reached we can say that the system has reached a steady state and there would be no heat transfer between the individual points on the grids and more or less all the points on the grid would have the same temperature.



Profiling Information and Possible SpeedUp

- From the call graph function we saw that the central( ) part of the code took 49.7% of the time of the total execution.
- From Amdahl’s Law the maximum speedup that we get can be obtained for 12 threads comes out to be 1.85. We got a value greater than that in experimental results.

Parallel Pseudo Code

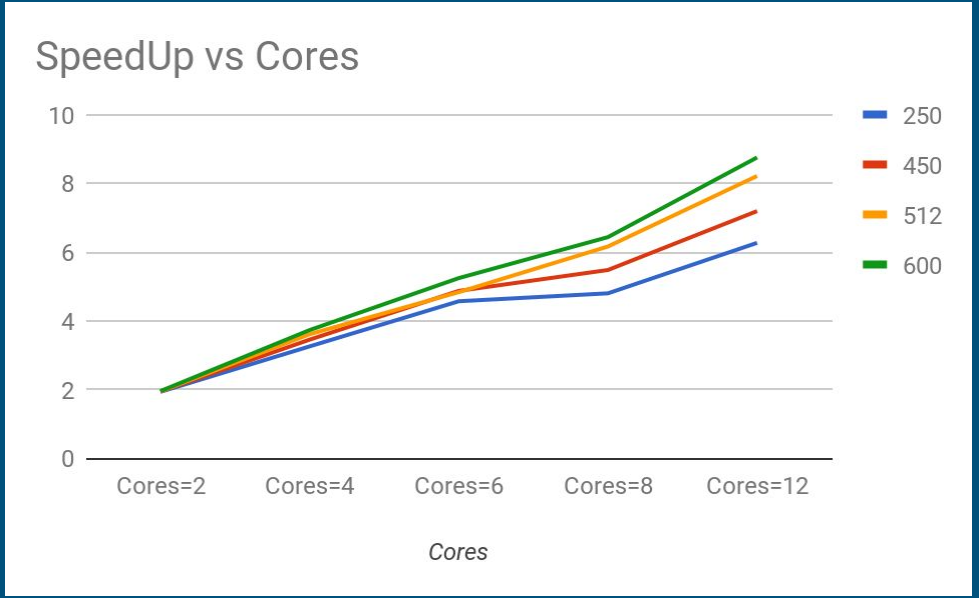
1) Set the boundary conditions such that the interior points have a reasonable value to start with.
2) Calculate the average for the interior points.
3) **Distribute amongst p threads.**
4) Iterate the loop for all the M*N points such that the older value of that point doesn't differ from the new value by more than the error tolerance.
    a) This we do by selecting each and every square and calculating the mean temperature (similar to the method followed in image processing) amongst its neighbour points.
    b) We check the error tolerance in respect to the grid point which has the maximum difference in temperature as compared to the previous value.
    c) We then update the previous value of the temperature of each grid point with the new value for the next iteration.

Optimization Strategy

- There aren’t many functions which require parallelization. The main time that was spent was in the count function which calculated the average. Applying the pragma for as shown below we were able to achieve the speedup as expected.
- The average calculation is an iterative method and with each iteration the values are updated for all the grid points. This results in a lot of time taken by each iteration. Hence all the loops have been parallelized.

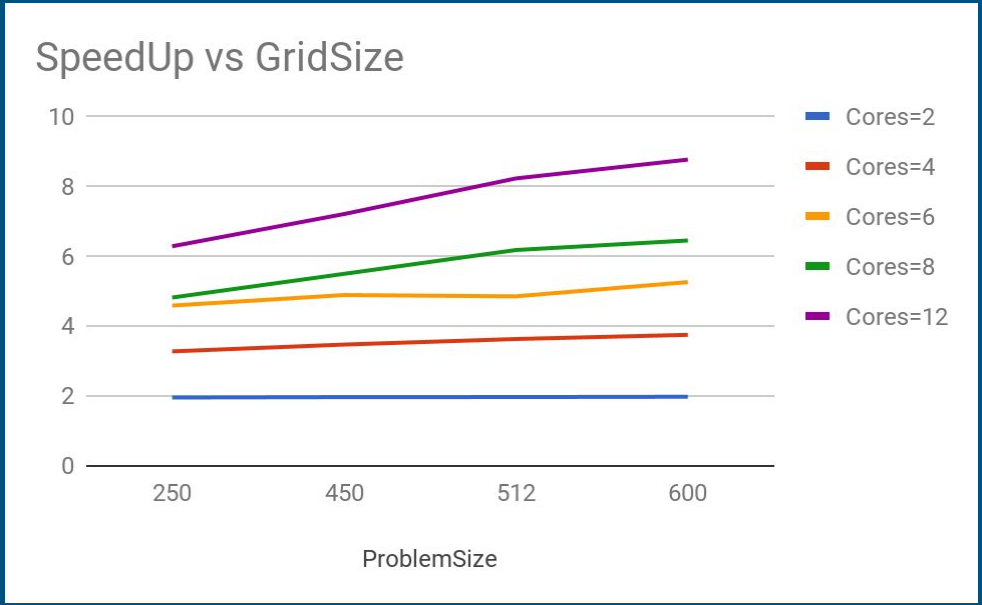
Speedup vs Cores

- With increase in the number of threads, speedup increases.
- Speedup increases as we increase the problem size.
- After 6 cores the speedup tends to flatten out.



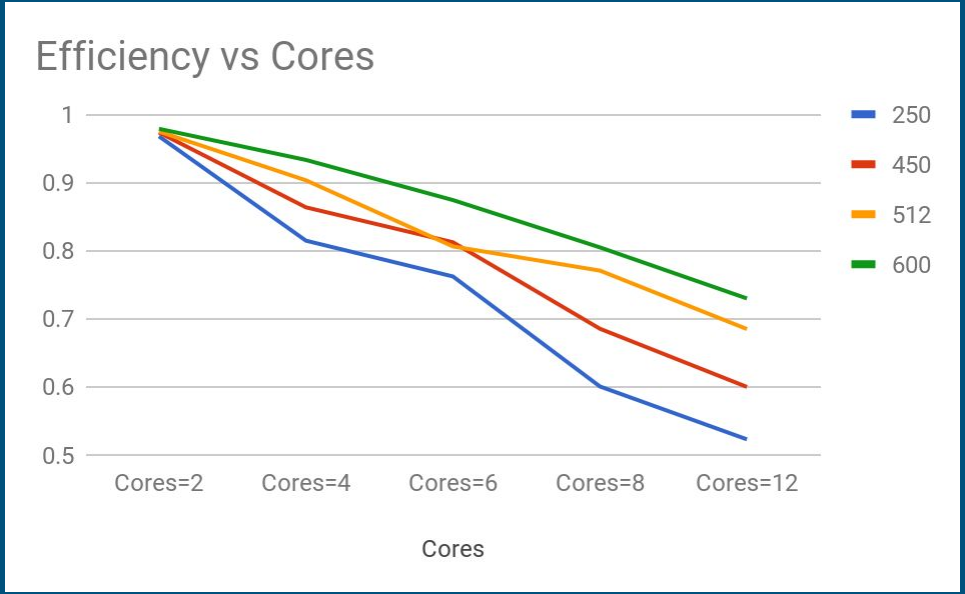
Speedup vs ProblemSize

- Speedup increases with the increase in number of cores. The speedup obtained is nearly equal to the number of cores (as expected) up till 6 cores. After that the speedup is high but not equal to the number of cores.



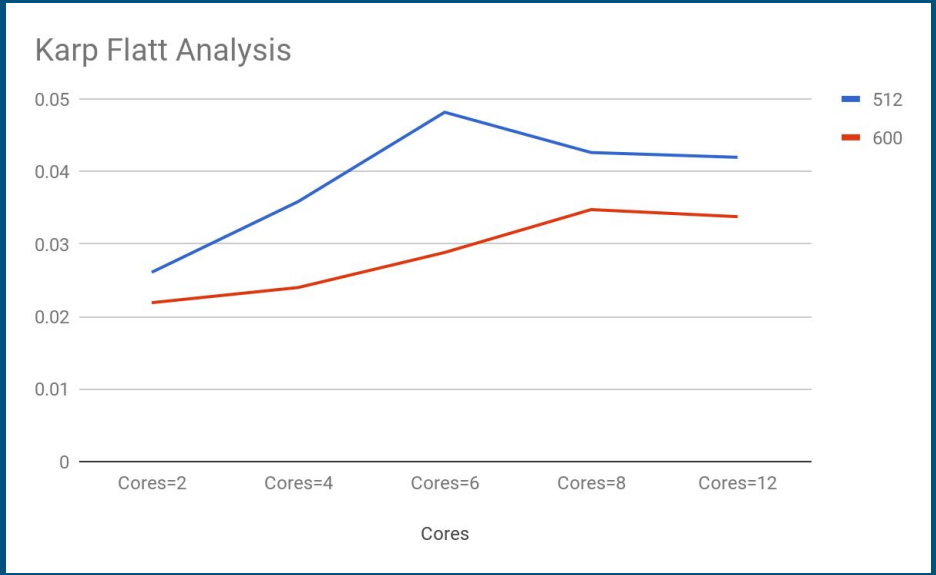
Efficiency vs No. Of Cores

- The efficiency is decreasing as the number of cores increase.
- As the problem size increases, the efficiency increases.
- We can conclude that the problem is weakly scalable.



Karp Flatt Analysis

- The Karp Flatt Metric for higher problem size is as shown above.
- Karp Flatt Metric shows that the parallelization is as per the requirement. Also we get the speedup that we expect according to the number of cores.



Conclusion

- Parallelised the two dimensional heat equation for a steady state condition using Open MP API.
- The gprof profiler helped a lot in understanding the points where we needed to parallelize the code.
- The speedup curve increases up till 6 threads and after that it flattens out.
- The reason for the same could be the hardware used.

Future Scope

- The parallelized version has been run on a 12 core machine only.
- We can run it on larger machines having up to 100 processors for larger input sizes. This would help us know the real life scenarios better.
- We can use the MPI libraries instead of the OpenMP ones for the increased number of processors.
- We have tried to implement a serial version considering the actual heat equation and all the related constants. That version could also be parallelized and better results could be found out using that equation.