High Performance Computing

Solving the 2D Heat Equation to get the steady state temperature (Gauss-Seidel Algorithm)

Amarnath Karthi Chahak Mehta

November 2017

2D Heat Equation

- Steady State Heat Equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

- The above equation when combined with the boundary temperatures provided form the Dirichlet problem for Laplace equation to find the steady state temperature of a rectangular plate at any point.

Essentially, the previous equation tries to solve the Laplace equation to obtain the steady state temperature at any point on a rectangular plate which gives a result visualized by this image:

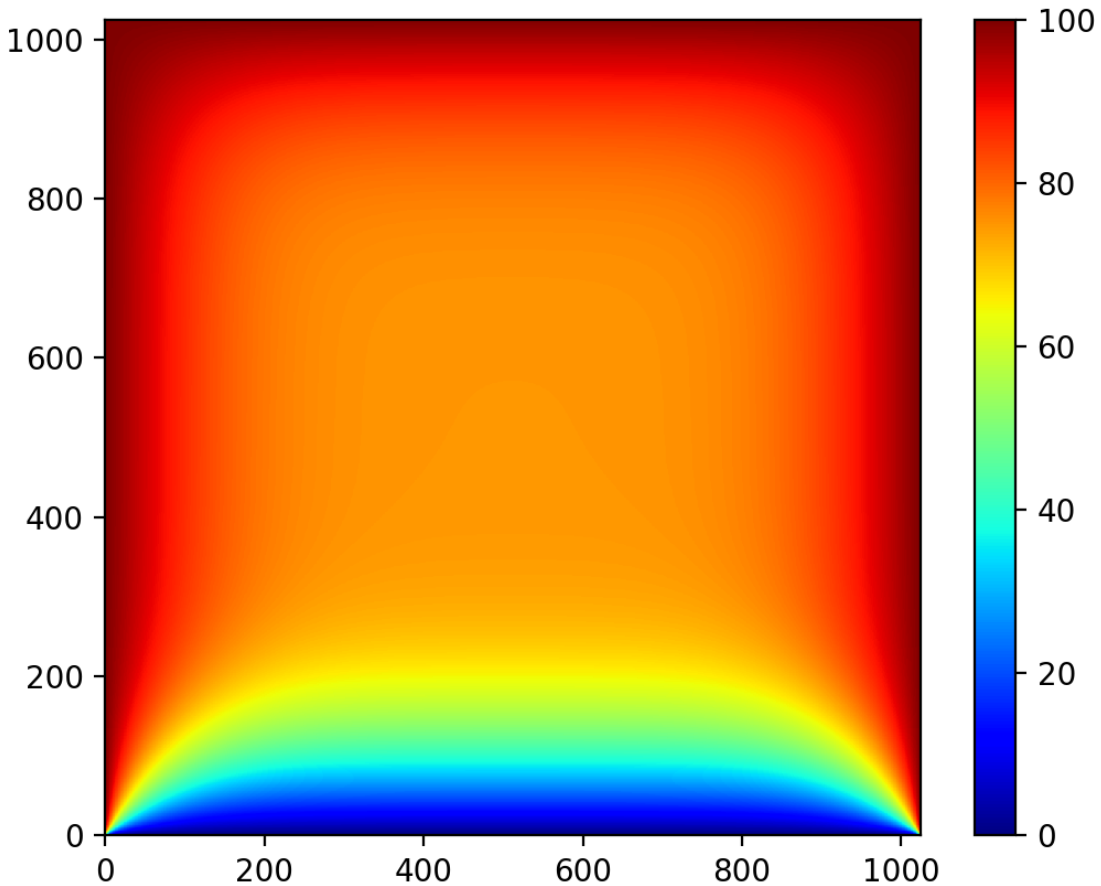

Figure: Steady state temperature of the plate

Gauss-Seidel Algorithm

- Gauss-Seidel Algorithm is an iterative method which can be used to find the solution to a system of linear equations $Ax = B$ using the iterative formula:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

- This algorithm can be used to solve the Laplace equation on the previous slide by changing it to a difference equation of the form:

$$\nabla^2 T \approx (T_n + T_s + T_e + T_w - 4T_c)/h^2$$

$$T_c \approx \frac{T_n + T_s + T_e + T_w}{4}$$

Computational Complexity

- Naive Gaussian elimination algorithm - $O((mn)^2)$ space complexity and $O((mn)^3)$ time complexity - computationally too expensive
- Gauss-Seidel algorithm - better solution for sparse matrices (as is the case here)

Parallelizing the algorithm

- The Gauss-Seidel algorithm doesn't have natural parallelism since each iteration depends on the values of the previous iterations and neighbours and hence the iterative loop cannot be parallelized by just using **#pragma omp for** in natural order traversal.
- Initialize the solution temperature grid with the mean values of the boundary conditions rather than random values so that the algorithm converges faster.

Decomposition schemes

- To maintain concurrency and adhere to the data dependencies, following decomposition schemes can be used:
  - Red-black decomposition scheme: decompose the matrix into red and black nodes - hence can perform multiple iterations at the same time.
  - Wavefront decomposition scheme: divide the matrix into independent *"wavefronts"*

Speedup Curves

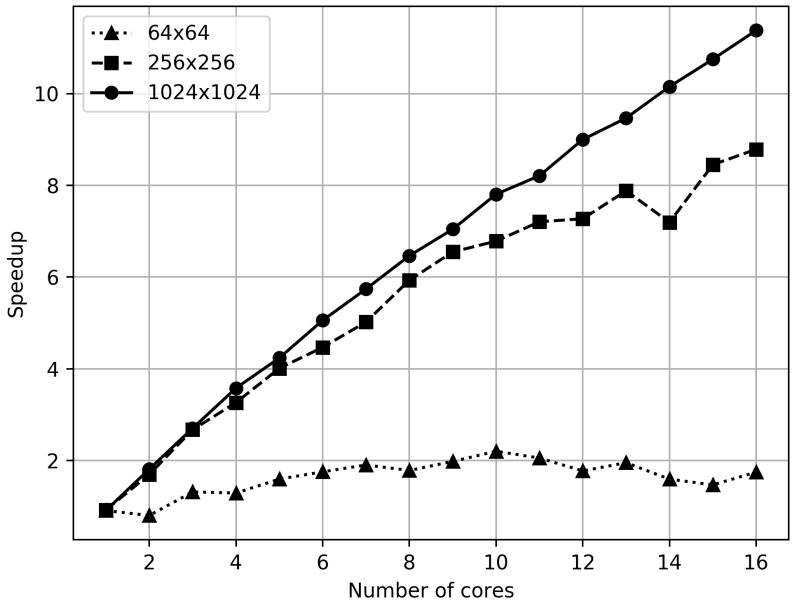

Figure: Speedup vs Number of cores

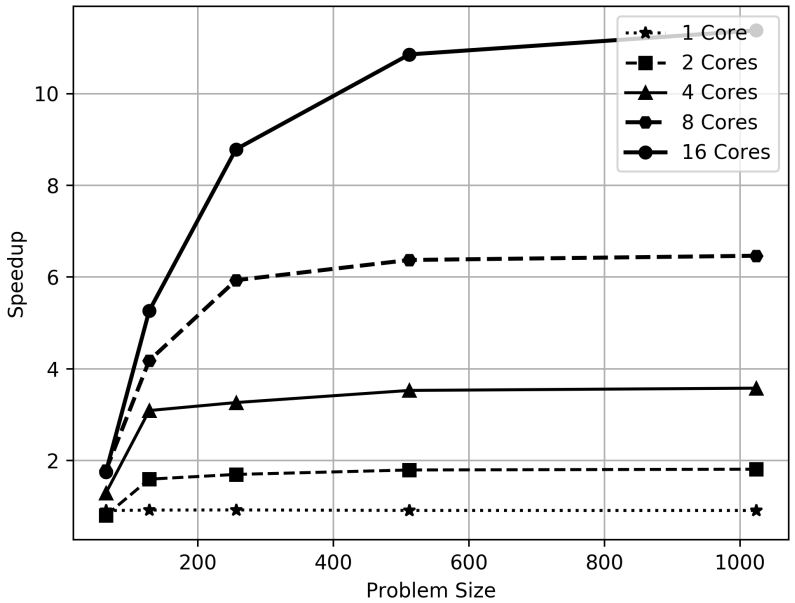

Figure: Speedup vs Problem size

Performance Analysis

- For small problem size, the parallel overhead is more than that of the time saved by the parallelization and hence we can see very less speedup.
- For larger problem size, we can see more speedup. This is because the compute time is very high for large problems and the speedup compensates for the parallel overhead that occurs because of the communication between the processors.
- The algorithm speedup doesn't scale linearly because of serialization bottlenecks for larger number of cores and because of the fact that even though the computation power increases, cache, memory bandwidth and other resources do not increase linearly.

Karp-Flatt Analysis

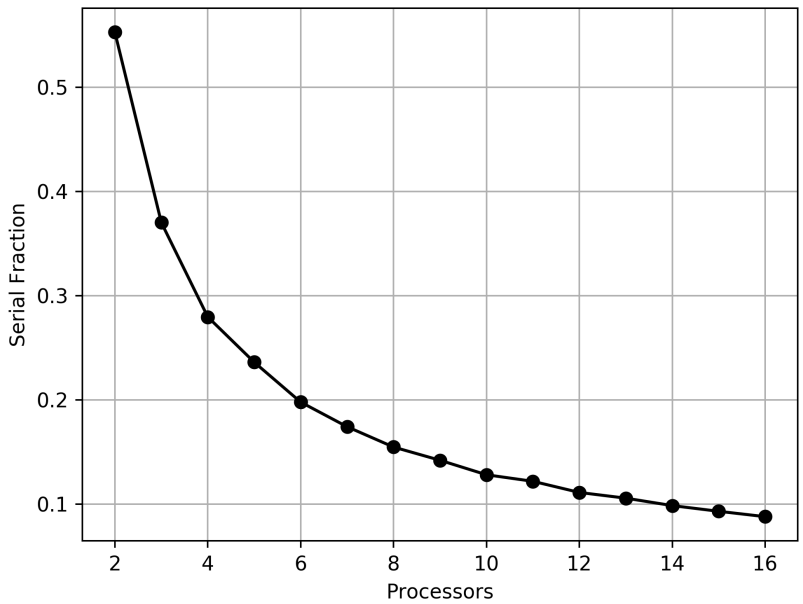

Figure: Serial Fraction using Karp-Flatt metric

- We can see in the above image that the serial fraction of the algorithm converges to 0 and hasn't become constant yet. This shows that there is a scope of better performance by using higher number of cores.

Any questions?

References

References:

1. Parallel SOR Iterative Algorithms and Performance Evaluation on a Linux Cluster:
   http://www.dtic.mil/dtic/tr/fulltext/u2/a449212.pdf
2. Numerical Solution of Laplace Equation, Gilberto E. Urroz, October 2004
   http://ocw.usu.edu/Civil_and_Environmental_Engineering/\Numerical_Methods_in_Civil_Engineering/LaplaceDirichletTemperature.pdf