**HPC project Tanner Salvage**

**Iterative Solvers for Diffusion Processes**

**Part 1**

The Poisson Equation is a partial differential equation (PDE) with many applications including modeling the heat equation, gravity and gas diffusion. Its relevance in many fields means that an efficient way of solving the equation is constantly sought after with a popular method being the use of iterative solvers such as the finite difference method. With this in mind, this study discusses the finite-difference-method (FDM) to solve the Poisson equation given in (1) and in particular, compares the performance of two popular algorithms BiCGstab (biconjugate gradient stabilized) and GMRES (generalized minimal residual) from the python library SciPy.

The 2D Poisson Equation can be written as

(1)

For with boundary conditions .

**Method**

To solve (1), the PDE must be written in terms of a linear system in the form where A is an N x N matrix of coefficients, x is a vector of unknowns and b the results vector. The resulting matrix is so large and sparse that it is impractical to store the full matrix and therefore finite difference methods are enforced to solve the simultaneous equations in a more efficient way.

Here, the centred finite difference method was used to discretize (1) by means of equation (2).

(2)

Where and and similarly for other values of . Here, is the solution on a given grid point and h is the step size.

The N x N matrix u was converted to a N^2 vector and the same was done for and and these vectors were subsequently fed into an OpenCL implementation to discretize the problem in parallel. Once this was achieved, the function was converted into a linear operator class to allow the use of bicgstab and gmres solvers.

By discretizing the problem in this way, we are left with a sparse matrix with no more than five non-zero elements in each row, i.e. the matrix has a bandwidth of five. Using the scipy.sparse.linalg allows us to store only the non zero elements of the matrix, therefore drastically reducing storage.

**Validation**

To test the correctness of the code, the solution u (x, y) was compared to the solution given from a FEniCS implementation. Here, the operator sigma was defined as and for simplicity with results shown in Figures 1 and 2.

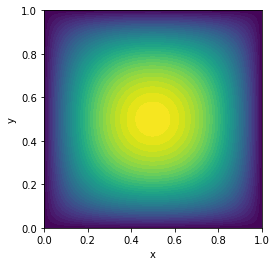
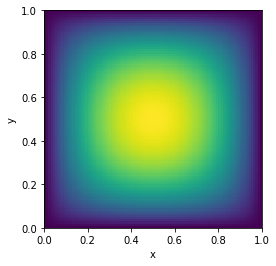


Figure 2 - FEniCS

Figure - OpenCL with bicgstab

The graphs show good agreement between both solutions, validating the results from the OpenCL implementation. With this confirmed, we move on to the experiments.

**Experiments and results**

The solvers bicgstab and gmres were implemented using parameters specified in table 1.

Table

|  |  |
| --- | --- |
| Parameter | Value |
|  | Where is a field of normally distributed random numbers. |
| Mean | 0 |
| Standard deviation | 0.1 |
|  | 1 |

To test for convergence, the residual was calculated at each iteration for both methods and plotted in figure 3 where the quicker convergence rate of bicgstab is immediately obvious. From the figure we can conclude that whilst both methods converge, bicgstab makes more progress towards the solution per iteration making it the more efficient solver in this case.

By varying N we can further examine the efficiency of the solvers. Looking at figures 4 and 5, we can see that when N is small (<40), gmres is more efficient, preforming a similar number of iterations in around half the time of bicgstab, however this drastically changes as N increases: the number of iterations for gmres grows quadratically with N as seen in figure 3 as opposed to the linear relationship seen in bicgstab.

To formalize this relationship we define the convergence rate as and state that for gmres and for bicgstab. Further more, by noting that = , we can derive the relationship between convergence rate and step size as for gmres and for bicgstab.

From figure 5, we deduce that the cost per iteration is higher for bicgstab. This is explained by the fact that bicgstab preforms two matrix-vector calculations per iteration whilst gmres only preforms one **[1]**. The advantage of this extra cost though is clear from the discussions above. Despite the cheaper cost of gmres, its quadratic growth leads to the much slower convergence seen in figure 3 and this effect is only increased for larger systems, which in practice are more common and often much larger than this simulation (N >> 100). In this respect, bicgstab out preforms gmres and is more suitable for “real world” applications with these parameters.

From the evidence presented in this section, we conclude that bicgstab is a more suitable solver for this problem.

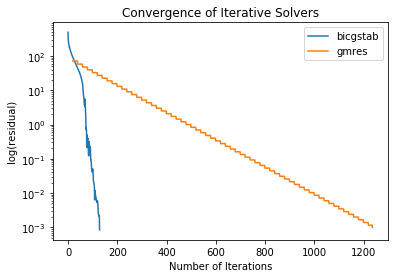


Figure 3

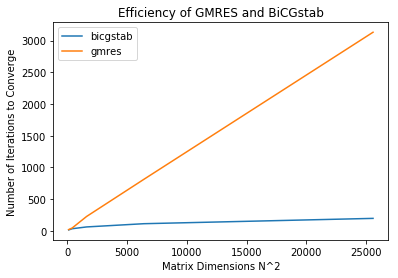


Figure 4

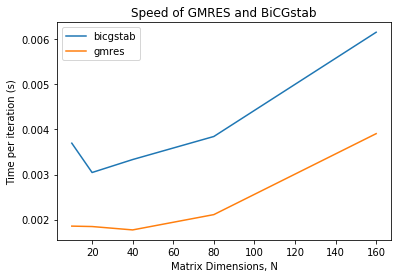


Figure 5

As a final test, the solvers were applied with the right hand side with N = 100 and all other parameters the same as previously. The results in table 2 indicate that bicgstab preformed well with a short computing time and acceptable number of iterations whilst gmres was unable to solve the solution within 100,000 iterations as clearly seen in the incomplete graph. Again, this only reinforces the prior discussion that the extra cost in bicgstab is beneficial for fast convergence and accurate solutions.

Table

|  |  |  |
| --- | --- | --- |
| Method | bicgstab | gmres |
| Time (s) | 0.8580 | 241.9 |
| Iterations | 188 | 100,000 |
| Graph | Macintosh HD:Users:tanner:Desktop:MSc_computing:HPC:sin.jpg | Macintosh HD:Users:tanner:Desktop:MSc_computing:HPC:sin.jpg |

**Conclusion**

To conclude, the Poisson equation was solved using a centred finite difference scheme with solvers bicgstab and gmres and validated using FEniCS. It was found that bicgstab, although more expensive, is a more efficient algorithm for most systems with a convergence rate of and should be chosen for large systems with sparse matrices. Additionally, gmres was shown to outperform bicgstab only for small systems of N < 40 due to the convergence rate of . In future studies, the solvers should be tested with different functions for to explore how the initial matrix effects the performance of the solvers.

**Part 2**

The parabolic equation in (3) can be interpreted as the 2D diffusion equation with diffusion coefficient of 1. The diffusion equation has many applications but is most commonly used to model the distribution of heat over time, which can have important implications such as estimating the age of the Earth [2] as well as more practical applications like designing household appliances including fridges and air-conditioning systems. This part of the study investigates heat diffusion using a forward difference approximation in OpenCL and specifically examines the influence of the time step and initial conditions on the solution.

The parabolic equation can be written as:

(3)

For the same space domain and operator described in part 1 with initial conditions and on the boundary .

**Method**

The finite difference method is applied to (3) by discretizing the right hand side as outlined in part 1 and the left hand side using the forward difference scheme in (4). By some rearrangement, equation 5 is produced describing the heat profile at the next time step.

(4)

(5)

To solve (5), the NxN matrices and were converted to N^2 vectors and fed into an OpenCL implementation to solve on the entire domain in parallel for a single time step. The function was then called for every time step in a loop. For the given boundary conditions, the problem can be physically interpreted as heat diffusion on a thin (approximately 2D), square metal sheet with boundaries held at 0 °C.

**Experiments and results**

**Experiment 1 – Step size**

It is well known that the time step has a direct impact on the accuracy and stability of the method. Here, the effect of the time step on the solution was investigated by varying and solving for the final time t = 0.005 s with results in figure (6).

The figures clearly show that for (figures 6b and c) the scheme is unstable and grows to extreme values of up to 1e^48 whereas for (figures 6d and e) the correct solution is produced with the heat from the source diffusing outwards and no growth in the maximum value.

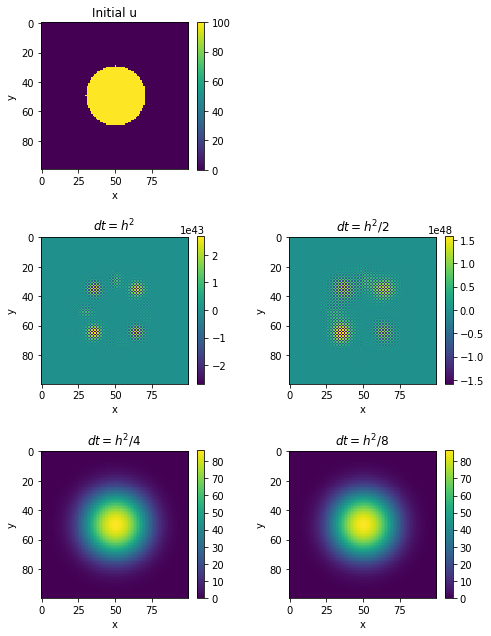


Figure 6 – diffusion after 0.005 seconds for various time steps. (top left (a), middle left(b), middle right(c), bottom left(d), bottom right (e)

These results can be explained by carrying out Von Neumann stability analysis [3]. Consider the Forward Euler scheme in (6).

(6)

Assume there exists a solution of the form where the term is known as the growth factor and is associated with time dependence and the exponential with space. Substituting these terms into (6) and rewriting in terms of sine and cosine gives:

(7)

For stability, we require , therefore we rearrange for to determine the stability condition

(8)

to minimize the cosine terms, take giving

(9)

(10)

We choose so as to maximize the left hand side, giving

(11)

Since we are solving on a unit square, we have giving the maximum time step as , as demonstrated in the experiment.

Notice that mesh size cannot be reduced independently from the step size, i.e. if then . This means that for a more accurate solution, we require not only a finer mesh but also a smaller time step, which in turn would require more iterations to arrive at a final time step. In this way, there is a compromise between accuracy and efficiency and therefore must be chosen carefully to reflect the desired accuracy as well as efficiency of the solution.

**Experiment 2 – Initial conditions**

The initial heating of the metal plate also plays a role in the diffusion of temperature and therefore, the heat profile of the metal plate was examined after 100 time steps with N = 100 and (maximum step size) for 3 different initial conditions.

* Condition A is a uniform diamond shaped heat source centred at (0.5, 0.5) with a radius of 0.4, width of 0.2 and temperature of 100 °C.
* Condition B is a source point, centred at (0.5, 0.5) with a radius of 0.195 (carefully chosen to give the same area as the heat source in A) and a temperature of 100 °C.
* Condition C is the equation with a maximum temperature of 100 °C, centred at (0.5, 0.5).

Looking at A and B in figure 7 we observe that A diffuses noticeably more than B with a maximum final temperature of 81 °C compared to 98 °C. This is because whilst the overall area of the heat sources are equal, the perimeter of A is much larger than B meaning there is a larger area for diffusion to take place and due to its shape, it can occur on both the inner and outer borders of the source whilst B can only diffuse from the edge of the circle.

C also has a lower maximum temperature than B at 95 °C, however this is not necessarily because it has diffused more. Whilst the maximum initial temperatures are equal, the temperature in C drops quickly as you move away from the center and comparing the initial temperature at the edge of the circle in B to the corresponding position in C, we find that C is only 67°C compared to 100°C. Newton’s law of cooling states that the rate of heat transfer is directly proportional to the temperature difference [4] therefore the it is reasonable that the final temperature of B is higher since there is no temperature difference within the circle, thus diffusion must take place on the edges and work its way to the center whereas C is immediately able to diffuse into the surrounding areas(albeit at a slow rate).

A visual comparison of B and C suggests that B has diffused more than C despite the higher temperature and this is also supported by Newton’s law of cooling; since there is a large temperature difference between the source and the rest of the metal plate in B, diffusion can take place at a faster rate than C. By running the experiment for 500 time steps we confirm that this is the case, with maximum temperatures of 53 °C and 78 °C for B and C respectively.

In all three cases, diffusion takes place from hot to cold and the maximum temperature at the final time is lower than the initial as expected.

Looking at the heat sources in A and B, we notice that the boundaries are not perfect due to the discretization of the problem. Using a finer mesh could improve this, but as previously discussed the small time step required would place a considerable time constraint on the problem. For a simple comparison of diffusion over time, this added accuracy and longer computing time is unnecessary. As an added remark, we note that since the mesh is finite the boundaries will never be perfect.

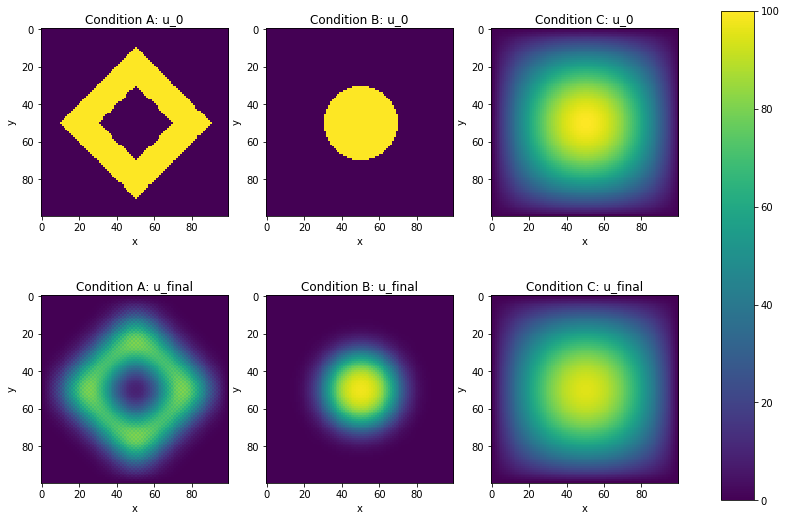


Figure 7 – Diffusion after 2.5 ms

**Experiment 3 – Performance**

Finally, we evaluate the performance of the code in terms of speed. The time to compute 10 time steps with OpenCL and NumPy was tested with condition B with results in table 3 (given to one significant figure due to the variance in computing time). Whilst the speeds for both methods are of the same order, OpenCL outperforms NumPy for larger systems. In practice, systems are often much larger than N=1000 therefore OpenCL will certainly offer an advantage however it’s performance can be improved by writing it in such a way to reduce global memory access, utilising local and private memory instead.

Table

|  |  |  |
| --- | --- | --- |
| Dimension (N) | Method | Time (ms) |
| 100 | OpenCl | 5 |
| Numpy | 4 |
| 1000 | OpenCL | 20 |
| Numpy | 70 |

**Conclusion**

In summary, a forward difference approximation to the 2D diffusion equation was simulated and it was found that for stability, the time step should be no larger than This time step was tested with different initial conditions to demonstrate how heat transfer depends on surface area and the temperature of the surroundings. Finally it was shown that OpenCL preforms better than NumPy for large systems but offers little advantage for smaller trivial problems.

**References**

[1] Amritkar, A., de Sturler, E., Świrydowicz, K., Tafti, D. and Ahuja,(2015). Recycling Krylov subspaces for CFD applications and a new hybrid recycling solver. Journal of Computational Physics, 303, pp.222-237.

[2] A. Stinner, "Calculating the Age of the Earth and the Sun", Institute of Physics, 2019.

[3] B. Peterson, "von Neumann Stability Analysis", Physics.byu.edu, 2019. [Online]. Available: https://www.physics.byu.edu/faculty/petersonb/Phys430/DiffusionStability.pdf. [Accessed: 29- Mar- 2019].

[4] M. Vollmer, "Newton's law of cooling revisited", European Journal of Physics, vol. 30, no. 5, pp. 1063-1084, 2009.