University College London

PHAS0102 - High Performance Computing

Project - Stage 1

Solving Poisson Equation Using Iterative Solvers and OpenCL Parrallelisation

Author:
Fahad Chohan

Supervisor:
Dr. Timo Betcke

March 8, 2019

1 Introduction

The aim of this project was to use iterative solvers to solve the Poisson equation below for $(x, y) \in [0, 1] \times [0, 1]$ with boundary conditions $u(x, y) = 0$. This was done using the linear algebra scipy package from which the gmres, bicgstab, minres and cg iterative solvers were used.

$$-\nabla \cdot (\sigma(x, y)\nabla)\, u(x, y) = f(x, y)$$

From this Poisson equation we approximate the first point by using a standard centered finite difference scheme. The left hand side (LHS) operator is approximated by

$$\nabla \cdot (\sigma(x, y)\nabla)\, u \approx A + B$$

$$A = \frac{\left(\sigma_{i+1/2,j} \frac{(u_{i+1,j} - u_{i,j})}{h}\right) - \left(\sigma_{i-1/2,j} \frac{(u_{i,j} - u_{i-1,j})}{h}\right)}{h}$$

$$B = \frac{\left(\sigma_{i,j+1/2} \frac{(u_{i,j+1} - u_{i,j})}{h}\right) - \left(\sigma_{i,j-1/2} \frac{(u_{i,j} - u_{i,j-1})}{h}\right)}{h}$$

where we approximate sigma by $\sigma_{i+1/2,j} \approx \frac{1}{2}(\sigma_{i+1,j} + \sigma_{i,j})$ and h is the spacing between matrix elements.

For $f(x, y)$ the equation $f(x, y) = x^2 + 2y^2$ is used. For $\sigma(x, y)$ the equations $\sigma(x, y) = 1 + x^2 + y^2$ and $\sigma(x, y) = e^{-S(x,y)}$ are used when comparing with FEniCS and running experiments respectively (where $S(x, y)$ is a field of normally distributed random numbers).

2 Code Implementation

The code starts off with a brief introduction followed by initialisation which imports functions, creates counter, variable and timer classes and creates $\sigma(x, y)$ and $f(x, y)$.

The Timer class is used to time how long the iterative solvers take. The gmres_iteration_counter and other_iteration_counter classes are used via the callback parameter in the solvers to count the number of iterations and to

calculate residuals for each iteration. Two seperate classes were created because the gmres solver directly outputs residual values whereas the other solvers (bicgstab, minres and cg) output the solution vector after each iteration. For the bicgstab, minres and cg solvers, $r_k = f(x, y) - x_k$ is used to calculate the residual after each iteration where $f(x, y)$, x_k and r_k are the RHS function, solution vector and residual after each iteration respectively. These are then normalised.

The next section creates the openCL matrix vector function which takes in a vector u (flattened $u(x, y)$ using ravel() function) as a parameter and applies the LHS operator in a matrix free way to return a vector Au. Boundary conditions are preserved by an if statement in the kernel. Please note that the iterative solvers by default use a vector of zeros as an initial guess, if the user would like to provide an initial guess to the solvers then he must make sure that the initial guess vector boundaries are consistent (zeros in this case).

The openCL implementation uses a workgroup size of 1 and applies the matrix operation on all the elements simultaneously as they do not need to be synchronised within each solver iteration. When the iterative solvers apply the next iteration all the values of the vector are up to date, ensured by the queue.finish() command.

Then a LinearOperator object is created which uses this openCL function and can be used in the gmres, bicgstab, minres and cg solvers. Subsequently, these four solvers are used to compute a solution and this is then visualised using the scipy matshow() function and the solvers residuals against iterations are plotted.

The final part of the code solves the Poisson equation using FEniCS and is used for comparison to the iterative solvers in the previous section.

3 Experiments and Results

3.1 FEniCS Comparison

To test the iterative solver code a comparison was made with a FEniCS solution (defualt solve parameters) for $f(x, y) = x^2 + y^2$ and $\sigma = 1 + x^2 + y^2$. The

values were compared by exporting to paraview and from the array, both solutions agreed with insignificant differences. The colour map representations can be seen in Fig.1 below.

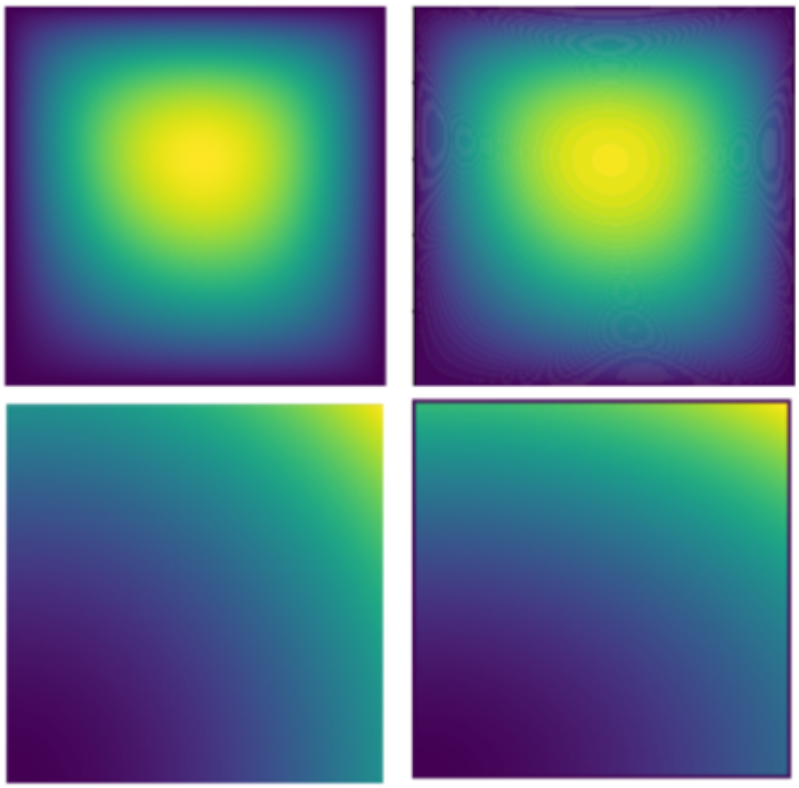

Figure 1: (top left) Solution from gmres iterative solver. (top right) Solution generated by FEniCS. (bottom left) $\sigma(x, y)$. (bottom right) $f(x, y)$.

3.2 Residuals

The change of residual against iteration count was calculated and plotted for each of the four solvers using $(100, 100)$ element matrix domains, where the solvers terminated after achieving a tolerance of 10^{-5}. The same $f(x, y)$ was used but $\sigma(x, y)$ was different. Instead, $\sigma(x, y) = e^{-S(x,y)}$, was used where $S(x, y)$ is a normal distribution of numbers (shown in Fig.2 below)

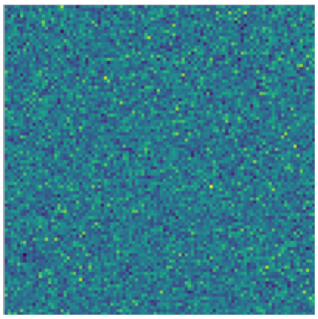

Figure 2: $\sigma(x, y)$ with $S(x, y)$ as a random normal distribution field.

The cg solver residuals seemed to decrease quickest followed by bicgstab, minres and gmres respectively, this can be seen in Fig.3

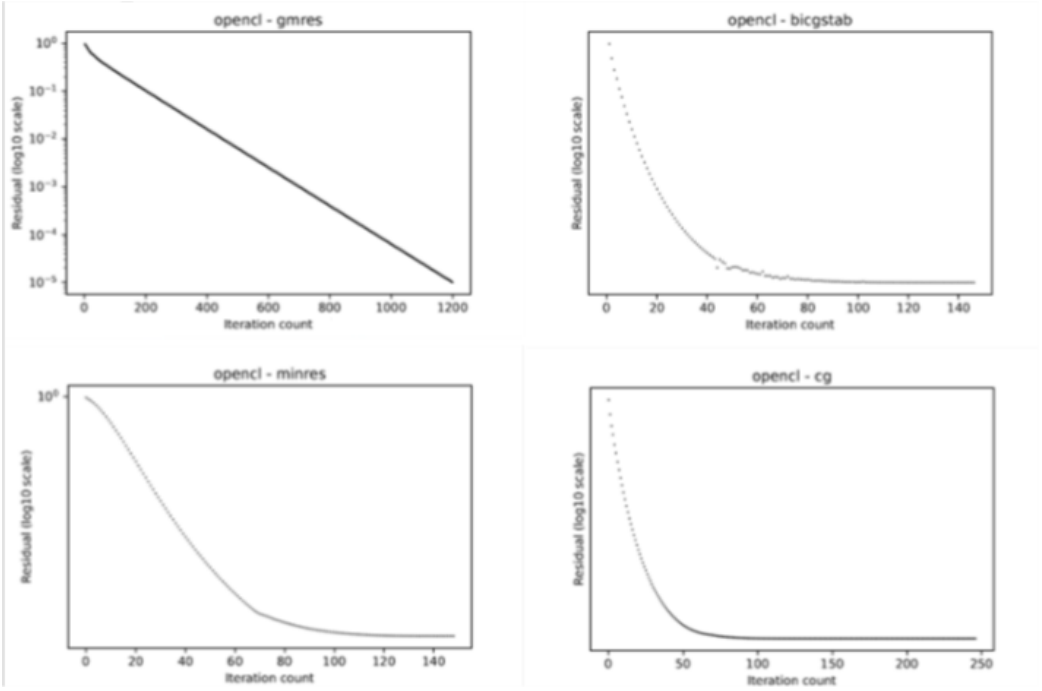

Figure 3: Residuals for gmres, bicgstab, minres and cg solvers changing with each iteration.

3.3 Dimension vs Time and Number of Iterations

The total time and number of iterations taken for the iterative solvers to converge were compared against number of matrix elements, Fig.4 and Fig.5 show the results.

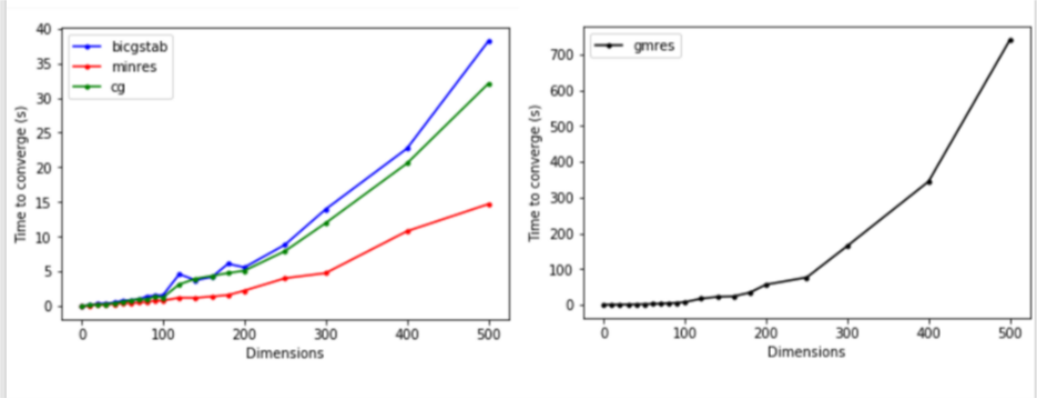

Figure 4: Total time against number of matrix elements.

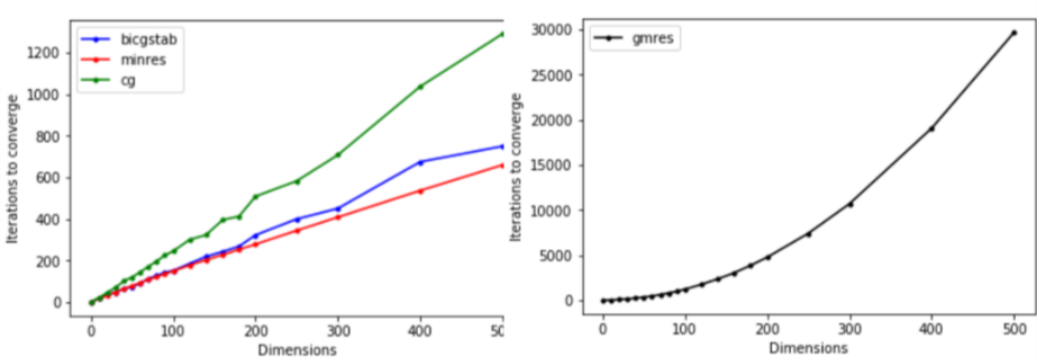

Figure 5: Number of iterations against number of matrix elements.

The increase in number of matrix elements had the strongest effect on the gmres solver which seemed to show a $y = x^2$ dependance for both total time taken to solve and number of iterations. The other solvers also had the same type of relationship for number of matrix elements against total time, however for the number of iterations and number of matrix elements they seemed to have a linear relationship.