Techniques of High Performance Computing - Final Project

Liam Eloie

May 2017

1 Introduction

Solving partial differential equations can be a difficult task and can require a huge amount of computational power and time. Many of these problems are near impossible to solve on a single computer, making it necessary to distribute and utilise parallelisation over a whole cluster of computers. As a result, numerical methods for splitting, distributing and solving large matrices in parallel have been developed to do exactly this. In this report, I will be discussing, analysing and comparing Schwarz decomposition methods for solving the diffusion equation.

2 Domain Decomposition Strategy

2.1 Multiplicative Schwarz Decomposition

The first decomposition method that will be discussed is the multiplicative Schwarz decomposition method. The main idea behind this method is to partition a mesh and solve the partial differential equation over smaller 'subdomains' in an alternating fashion. Suppose there exists a domain Ω, which can be split into three overlapping subdomains Ω_1, Ω_2, and Ω_3 as seen in Figure 12, the multiplicative method works as follows:

1. Choose an initial guess, u, for the solution of the global domain.
2. Solve the partial differential equation in Ω_1.
3. Solve the partial differential equation in Ω_2 and Ω_3, taking into account the updated values of u in the boundaries shared with Ω_1.
4. Solve in Ω_1 again, but taking into account the updated u values in the shared boundaries.
5. Repeat until a desired residual threshold is found.

In terms of pseudocode, the procedure would become [4]:

```
Choose an initial guess $u$ to the solution;
Until convergence Do ;
for $i$=1,...,$s$ do
    Solve $\Delta u$ = f in $\Omega_i$ with $u = u_{ij}$ in $\Gamma_{ij}$;
    Update u values on $\Gamma_{ji}$, $\forall j$
end
```

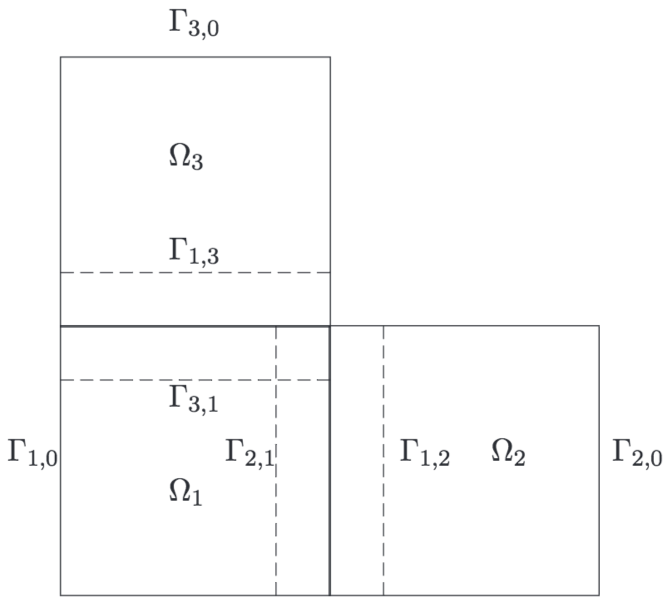


Figure 1: An example of a domain split into three overlapping subdomains [4].

2.2 Restriction Operators

In order to retrieve the information of each subdomain, it is necessary to compute restriction operators that allow us to map from the global stiffness matrix to the subdomain systems. Restriction operators are matrices describing whether vertices exist in a certain region of the mesh and are made up of 0s and 1s.

For example, if there are eight nodes in our mesh and a partition vertically straight down the middle, as seen in Figure 2, the restriction operators corresponding to regions 1 (left) and region 2 (right) can be written as the following two matrices. These are n_i by n matrices, where n_i is the number of nodes in region i and n are the total number of nodes in the mesh.

$$R_1 = \begin{pmatrix} 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{pmatrix}$$

$$R_2 = \begin{pmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&0&1 \end{pmatrix}$$

Using these, the global stiffness matrix can be mapped onto subdomains performing the computation in Equation 1

$$A_i = R_i A R_i^T \tag{1}$$

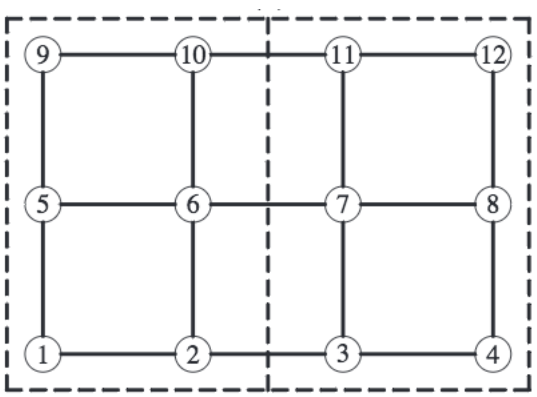


Figure 2: An example of a partition splitting nodes into two regions [4].

Equation 1 makes it possible to compress the multiplicative Schwarz procedure into a simple update equation.

```
Choose an initial guess $u$ to the solution;
Until convergence Do ;
for $i$=1,...,$s$ do
    $u := u + R_i^T A_i^{-1} R_i(b - Au)$;
end
```

2.3 Additive Schwarz Decomposition

The big difference between additive and multiplicative Schwarz decomposition is that multiplicative solves by alternating through the subdomains and adjusting the global solution u accordingly whilst additive solves all the subdomains first and then sums the contributions to the global solution u. This procedure can be easily implemented as the following pseudocode.

```
Choose an initial guess $u$ to the solution;
Until convergence Do;
for $i$=1,...,$s$ do
    Compute $\delta_i = R_i^T A_i^{-1} R_i(b - Ax)$;
end
$u_{\text{new}} = u + \sum_{i=1}^{s} \delta_i$
```

A benefit for doing additive Schwarz decomposition, is that it is possible to easily parallelise the procedure. This can be done by sending each subdomain to a separate processor, solving the subdomain, and then collecting the residuals on a root node to update the solution.

2.4 Schwarz Preconditioning

Preconditioners are used to transform hard to solve linear equations such as

$$A\vec{x} = \vec{b}$$

into easier to solve equations,

$$M^{-1}A\vec{x} = M^{-1}\vec{b}$$

Fortunately, as well as being used to solve systems, Schwarz decomposition methods can be used to create preconditioners that can be used in other types of solvers, such a Krylov subspace solvers.

2.4.1 Multiplicative Schwarz Preconditioner

The way in which a preconditioner can be obtained from the multiplicative Schwarz method is by the iterative strategy in Equation 2.

$$M_i = M_{i-1} + R_i^T A_i^{-1} R_i (I - AM_{i-1}) \tag{2}$$

2.4.2 Additive Schwarz Preconditioner

Additive Schwarz differs from multiplicative Schwarz in the sense that it consists of updating all subdomains from the same residual. Therefore, the preconditioner is calculated in a slightly different way – seen in Equation 3.

$$M_i^{-1} = \sum_{i=1}^{s} R_i^T A_i^{-1} R_i \tag{3}$$

3 Automatic Non-overlapping Partitions

Choosing partitions can be a complicated task. Other than manually placing partitions in a grid, it is possible to use certain algorithms to decide where the divisions should lie. These methods generally try to minimise the number of edges cut by the partitions - in a way to minimise the error introduced by cutting through elements.

There are a couple of tools that can be used to do this. The first one is a Python package called NetworkX which is used for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks [2]. NetworkX has a method that creates a directed graph from an adjacency matrix of a mesh. This graph can then be parsed into a package called METIS [3], which uses specialised algorithms to create partitions by minimising the number of edges cut.

Once the partitions have been created, it is easy to segregate the nodes in the mesh into each region, which can then be passed into a Schwarz decomposition solver.

4 Parallelisation Strategy

Parallel computing is the idea of solving a problem by distributing independent parts of the problem to a number of processes to be solved. This allows multiple tasks to be run at once, which would otherwise run in series on a single processor.

Since the nature of additive Schwarz decomposition is to solve each subdomain independently of every other subdomain, it makes sense that this task can be parallelised. Consider splitting up a domain into N parts, it is possible that these N domains can be sent to M processors which will independently solve the the PDE in those subdomains. The M processors then send their contributions for each subdomain they solved to the global solution to a root node, which adds them all together and produces a solution to the PDE.

Message Passing Interface (MPI) [1] protocol was used to implement this procedure. First, the mesh must be partitioned into a number of subdomains. The subdomain matrices and restriction matrices for each one is found and stored in an array called subdomain_pairs. Using a Python wrapper for MPI, mpi4py, the subdomain_pairs are put into chunks and are distributed over a number of specified processors from the root node.

```
if(rank == 0):
    subdomain_pairs = subdomain_pairs
    chunks = [[] for _ in range(number_of_processors)]
    for i, chunk in enumerate(subdomain_pairs):
        chunks[i % size].append(chunk)
else:
    subdomain_pairs = None
    chunks = None

subdomain_pairs = comm.scatter(chunks, root=0)
```

Each processor now has access to a subset of the total subdomains. For each subset of subdomains in each processor, additive Schwarz decomposition is executed and the residuals are stored into an array called subdomain_residuals. After all processors are finished calculating the all the residuals for their subdomains, the residuals are gathered back into the root node.

```
subdomain_residuals = []
for subdomain_pair in subdomain_pairs:
    residuals = np.zeros(rhs.shape)

    subdomain_matrix = subdomain_pair[0]
    restriction_matrix = subdomain_pair[1]

    residuals = residuals + ... (calculate residuals)
    subdomain_residuals.append(residuals)

comm.Barrier()

subdomain_residuals = comm.gather(subdomain_residuals, root=0)
```

The root node then adds up all the contributions of the residuals to the initial solution to give an updated solution. This is done efficiently by using the reduce function on a lambda expression that adds up all the residuals from the subdomain_residuals array.

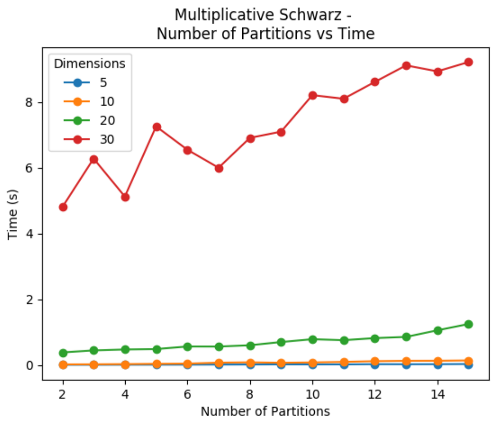
```
if(rank == 0):
     initial_solution = initial_solution +
     reduce(lambda x, y: x + y,
                 [item for sublist in subdomain_residuals for item in sublist]
           )
```

The subdomains are then solved again, using the new updated solution. This procedure is repeated until some convergence criteria.

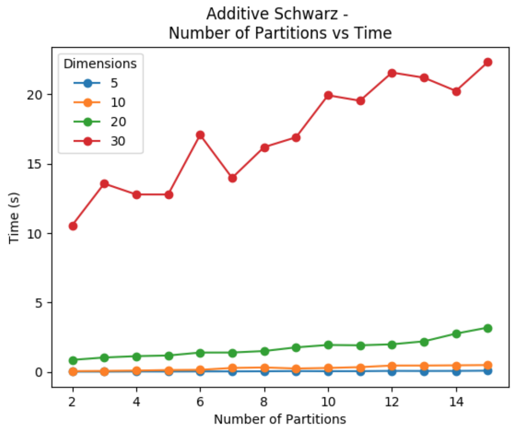
5 Analysis

5.1 Increasing Partitions

A mesh can be broken up into as many partitions as there are elements, and the subdomains can be made really simple to solve. An experiment was set up in order to see the behaviours of multiplicative and additive Schwarz decomposition when the number of partitions increase. The number of partitions tested was between 2 and 15 and they were both fairly tested on unit square



(a) Multiplicative Schwarz

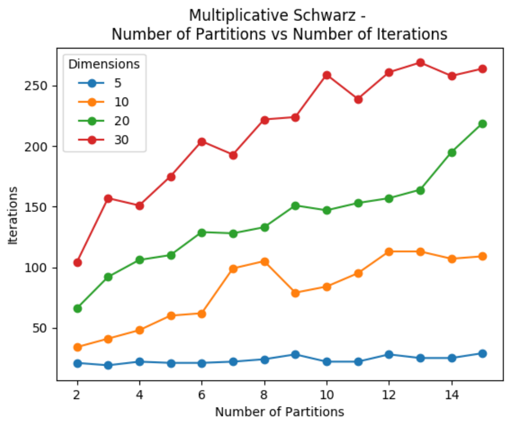


(b) Additive Schwarz

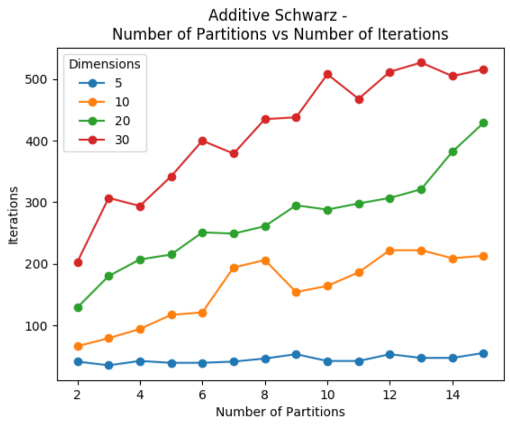
Figure 4: The time to solve increases as the number of partitions increase.

grids with dimensions 5, 10, 20, and 30. Their stopping criteria was a residual of 10^{-10}. From this experiment, it can be seen that it is undesirable to break the mesh into too many partitions. Figure 3a and 3b suggest that the increase in partitions increases the number of iterations required for both multiplicative and additive Schwarz to effectively solve. As expected, it can be seen that additive Schwarz takes more iterations than multiplicative Schwarz. This is because the discrepancies in the subdomain solutions at the boundaries are more abrupt, and therefore take longer to iron out.

Perhaps a natural extension from the increase of iterations due to the increase of partitions, is the increase in time taken to solve the system. This can be seen in Figure 4a and 4b for both multiplicative and additive Schwarz decomposition. In general additive Schwarz tends to take longer than multiplicative Schwarz. This is due to the fact that additive Schwarz has to calculate through a lot more iterations to converge.



(a) Multiplicative Schwarz



(b) Additive Schwarz

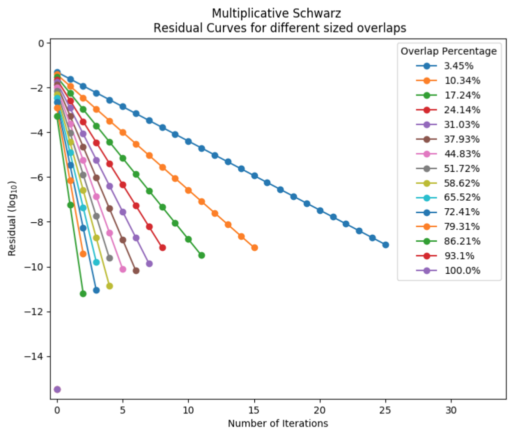
Figure 3: The number of iterations increases as the number of partitions increase.

5.2 Increasing Overlap

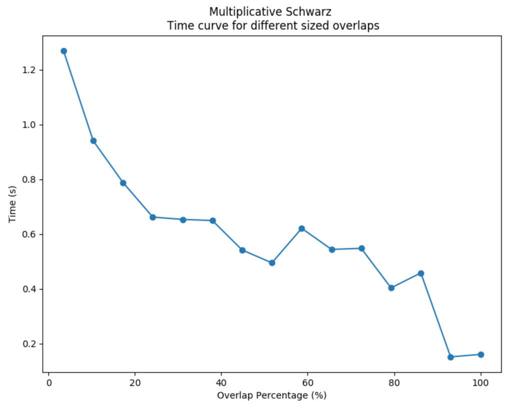
Having overlaps between the subdomains can be a way of more smoothly merging the solutions of each subdomain together. To test the behaviour of both multiplicative and additive Schwarz decomposition methods with increasing overlap size, an experiment was set up.

The experiment consists of a 30 by 30 square grid with one vertical partition down the middle. The overlaps are chosen such that the boundaries of the partitions don't cut through any elements. This is important, as cutting through elements will cause bad approximations for the integral over the elements being cut. Additionally, the overlaps are symmetric about the central partition. The overlaps are expressed in terms of total area of the mesh covered and go from the minimum overlap possible to the maximum overlap possible.

Figure 5a clearly demonstrates that an increase in overlap coverage decreases the amount of iterations the solver has to go through for the multiplicative Schwarz solver. This make sense, as the larger the overlap, the more the whole system is solved after an iteration. Additionally, it can be seen in Figure 5b that the time taken to solve the system decreases as the overlap increases.



(a) Residual curves for different sized overlaps.



(b) Time curve for different sized overlaps.

Figure 5: The residual curves and time curve for increasing the overlap for multiplicative Schwarz decomposition.

There is a different story when it comes to having an overlap with additive Schwarz decomposition. The iterative procedure of calculating the solution in all subdomains first and then adding them together all at once doesn't take into account what happens in the overlaps between subdomains. Because of this, when an overlap is introduced, additive Schwarz fails to converge to a proper solution. If the overlap were to increase, the area of discrepancy between iterations increases – owing to a solution with a worse convergence. This behaviour is demonstrated in Figure 6 where the smallest overlap has the smaller residual, and the largest overlap has the largest residual.

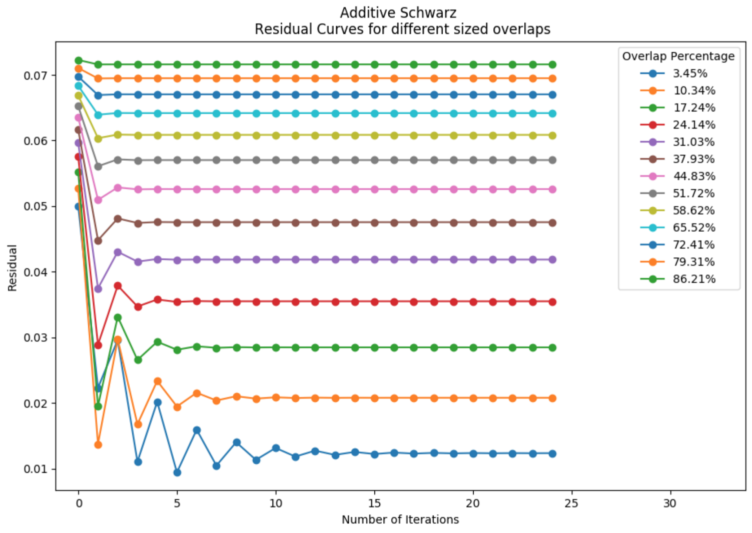


Figure 6: Residuals curves for additive Schwarz solver for increasing overlap.

5.2.1 Parallelisation

Mentioned earlier, a strategy can be devised to distribute subdomains in order to solve a system using additive Schwarz decomposition in parallel. In order to test this, a unit square 5 by 5 grid was set up for many different number of partitions $\{2, 5, 10, 15, 20, 25\}$. Note that the minimum amount of partitions is two and the maximum for a 5 by 5 grid is 25. These grids were solved using additive Schwarz decomposition for one processor and two processors. Figure 7 demonstrates the significant speed up gained when using the parallel strategy. On average, the parallel strategy introduces an increase of 40% in speed over non-parallel additive Schwarz.

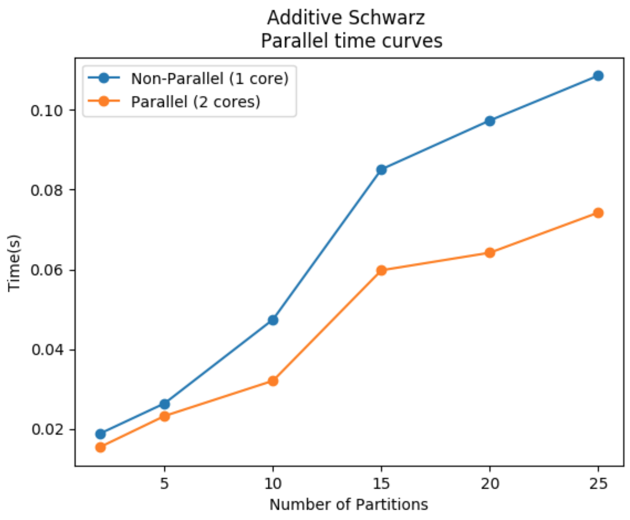


Figure 7: Time comparisons for additive Schwarz decomposition using one and two processors.

5.2.2 Pure Schwarz methods vs Preconditioned GMRES

Earlier, it was mentioned that Schwarz decomposition methods can be used as preconditioners for Krylov subspace solvers. Figure 8 is an example of the comparison between the residuals of pure Schwarz methods and their preconditioned versions using GMRES as a solver. It is evident that the preconditioned versions of GMRES perform much better than the pure Schwarz methods and GMRES alone.

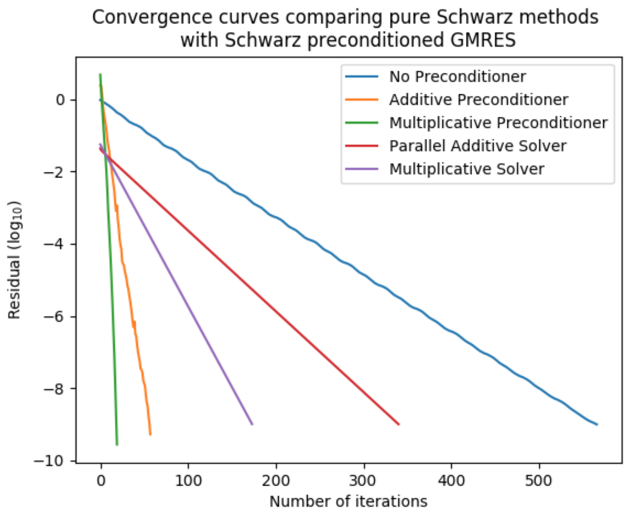


Figure 8: Convergence curves comparing pure Schwarz methods with preconditioned Schwarz for GMRES solver.

6 Krylov and Multigrid Subdomain Solvers

During Schwarz decomposition methods, the subdomain can be solved using an array of different methods. In the standard version, a direct solver is used – requiring the inverse of the subdomain matrix to be taken. This becomes a problem when the matrices become really large, since the inverse of large matrices take a very long time to compute. To get around this, the subdomains can be solved using iterative solvers such as Krylov subspace solver or multigrid methods. To perform these iterative solvers on the subdomains, the global initial solution was mapped into the subdomains by using the restriction operators. Using the subdomain stiffness matrices, right-hand-side and initial solution, a iterative solver can be used to solve for the subdomain solution.

```
# Map global initial solution to local initial solution.
local_initial_solution = restrictions[j] * np.matrix(initial_solution)

# Map global rhs to local rhs initial solution.
local_rhs = restrictions[j] * np.matrix(rhs)

if(solver_type == "gmres"):
    solution, _ = gmres(subdomains[j], local_rhs, x0=local_initial_solution)
elif(solver_type == "cg"):
    solution, _ = cg(subdomains[j], local_rhs, x0=local_initial_solution)
elif(solver_type == "minres"):
    solution, _ = minres(subdomains[j], local_rhs, x0=local_initial_solution)
```

```
elif(solver_type == "amg"):
    solution = ruge_stuben_solver(subdomains[j])
    .solve(local_rhs, x0=local_initial_solution)
```

The solution of each subdomain can then be mapped back from the local domain to the global domain to be contributed to the update global solution.

```
updated_solution = updated_solution + restrictions[j].T * np.matrix(solution).T
```

This process continues until some convergence critera, or a maximum amount of iterations have been performed.

All these methods were used to solve a 20 by 20 square grid with a constant diffusion field, partitioned into four subdomains. The solutions acquired by all these methods can be seen in Figure 9.

As you can see from this solution, the boundaries are very obvious between the subdomains. The initial solution was chosen to be zeros to highlight this discrepancy. The cause of this originates from the partitioning and solving of the subdomains. When iteraitve solvers are used over a subdomain the local residuals in the subdomain are minimised up to a convergence criteria. These residuals don't take into account the elements within the boundaries of the partitions, and so information about the diffusion is lost between them. This loss of information is carried through to the global system when mapped from the subdomains. The reason this doesn't occur in the direct solver is because the residuals are calculated over the entire system and then mapped into a subdomain, so there are no loss in information between nodes. Therefore, even though the subdomains are solved with low residuals – no matter the tolerance set on the iterative solvers, the global solution will never be able to converge to an acceptable solution. Therefore, if accuracy is desired over speed, direct solvers should be used. However, iterative solvers could possibly be used to minimise the residuals of the subdomains to an acceptable level. Then direct solves can be used to find a more accurate solution.

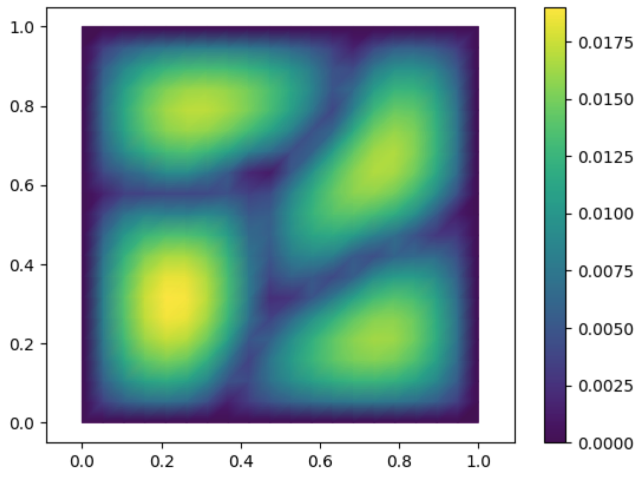


Figure 9: Diffusion equation solution acquired by iterative subspace solvers.

7 Further Examples

Although the bulk of analysis was done on square meshes with varying dimensions for fair testing, these methods can be easily extended to abnormally shaped grids with varying diffusion constants and right-hand-side functions. Below, are some examples of these more extravagant systems.

7.1 Square with a hole in the centre

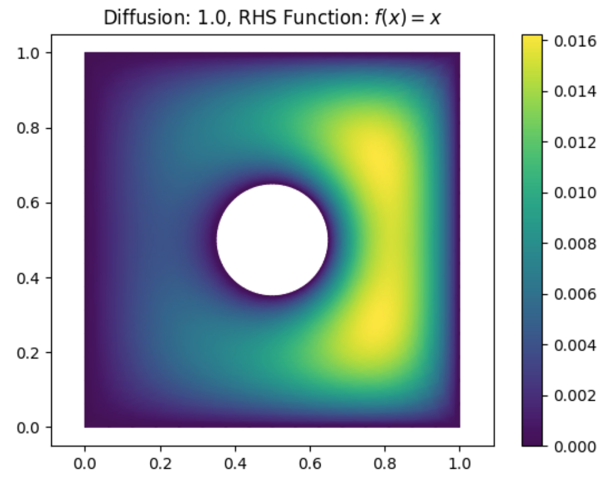


Figure 10: A square grid with a hole in the centre.

7.2 L grid

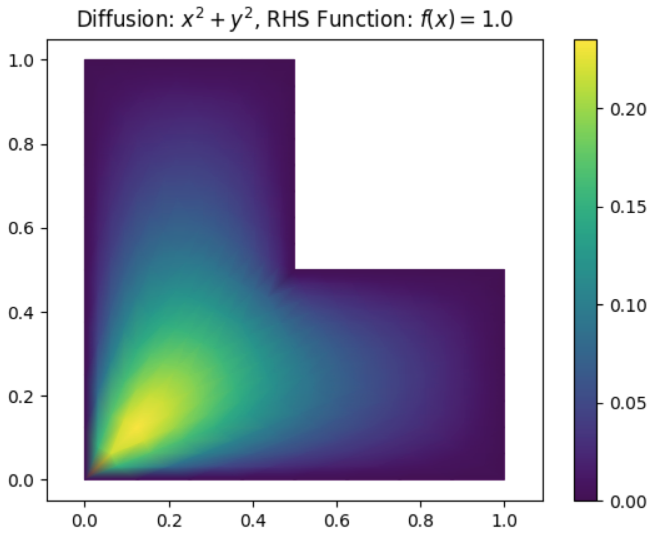


Figure 11: A L shaped grid.

7.3 Tear drop grid

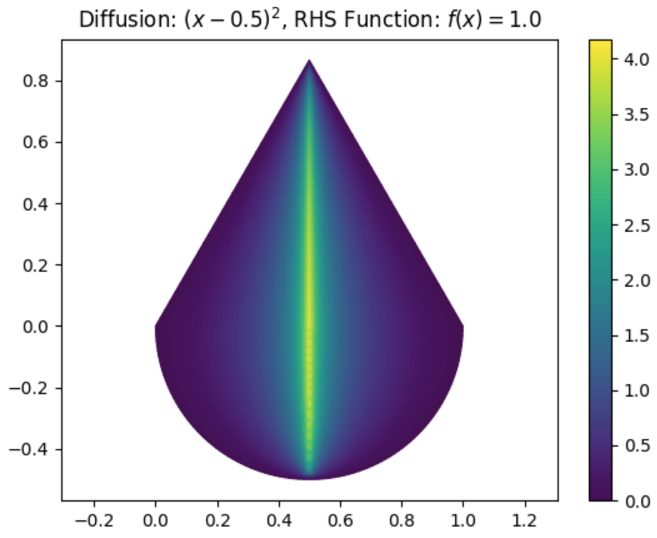


Figure 12: A tear drop grid.

8 Conclusion

Schwarz decomposition methods were successfully used to solve the diffusion equation over large and complicated grids by decomposing into and solving smaller subdomain problems. Two types were tested, multiplicative and additive, differing by how they are updated upon iterations. In general, a pure multiplicative Schwarz solver solved a problem in less iterations and time than an pure additive Schwarz solver. However, additive Schwarz solver were easily parallelised, giving an immediate increase in speed. Additionally, both pure Schwarz solvers were tested with increasing overlap sizes between a partition. It was found that multiplicative is able to converge to a solution much quicker with larger overlaps. Conversely, additive Schwarz is unable to converge when an overlap is introduced – the larger the overlap, the larger the final residual. Using non-overlapping partitions, it was also found that both multiplicative and additive Schwarz methods take more iterations and more time to solve. Both Schwarz methods can be used as a preconditioner for other solvers, such as Krylov subspace solvers. GMRES was preconditioned with both multiplicative and additive Schwarz methods, and it was found that the preconditioned versions performed much better than the non-preconditioned version of GMRES. Specifically, a multiplicative preconditioned GMRES outperforms an additive preconditioned GMRES. These preconditioned GMRES solvers were also found to take less iterations than any of the parallel Schwarz methods. Finally, iterative solvers were used to solve the subdomains during Schwarz decomposition methods. It was found that these were unable to successfully converge to a true global solution due to the information lost in the boundaries of the partitions. The most accurate solution is found with the direct solvers, as you lose no information. However it can be extremely computationally expensive to solve large grids, even when divided into smaller subdomains.

References

[1] Message P Forum. Mpi: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[2] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[3] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[4] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.