Chapter 1

Verification and Performance

In this chapter we will go through different methods we used to verify the multigrid solver, as well as scaling measurements. Modular parts of the solver is tested with unittests where feasible. In addition the whole solver is tested with both analytically solvable test cases and randomly generated fields.

1.0.1 Error Quantification

In order to evaluate solutions we will use two different measurements, the error, $\bar{E}$, based on the average deviation from a correct solution and the residual, $\bar{r}$ based on the average residual. Both methods is independent of the problemsize so differently sized problems can be compared.

$$\bar{E} = \frac{1}{N^d \bar{\phi}} \left(\sum_i \hat{\phi}_i - \phi_i \right) \tag{1.1}$$

$$\bar{r} = \frac{1}{N^d} \left(\sum_i \nabla^2 \hat{\phi}_i + \rho_i \right) \tag{1.2}$$

1.1 Multigrid Solver

To test the solver itself we employ a couple different techniques. First we create a charge distribution by differentiating a known potential, and then running the solver and check if the resulting potential was equal to the original known potential.

For the second test we use a charge distribution with a known analytical solution, and we then check that the solver reproduces the known analytical solution.

A third method we use to verify it is to produce a random charge potential and then check that the potential converges, or in other words that the residual

goes toward zero.

Then lastly we use the solver on identical charge distributions with the domain divided up into different subdomains and check that the solver produces the same potential.

1.1.1 Predetermined Potential

In this section we first decide which potential we want, then numerically construct a corresponding charge potential by derivating. Then we compare the result with the original potential.

1.1.2 Analytical Solutions

$$f(x) = \tag{1.3}$$

1.1.3 Convergence of Residual

1.1.4 Different Domain divisions

1.2 Unittests

Unittests are small tests that is used to check that the single pieces of the code work as they should. This serves a dual purpose in developing a software project. When a part of the code is developed it serves as a framework to create a standardized test of the piece of code that can easily be repeated. It also helps when developing the higher level algorithms, in that the unittests ensures that the problem lies in the higher level algorithm and not in the lower level pieces it uses. When implementing wider changes, for example datastructures, the unittests can help making sure that the changes are not causing any unintended bugs. For information of how to use the unittests see the documentation, **documentation**

1.2.1 Prolongation and Restriction

The prolongation and restriction operators with the earlier proposed stencils will average out the grid points when applied. So the idea here is to set up a system with a constant charge density, $\rho(\mathbf{r}) = C$, and then apply a restriction. After performing the restriction we can check that the grid points values are preserved. Then we can do the same with the prolongation. While this does not completely verify that the operators work as wanted, it gives an indication that we have not lost any grid points and the total mass of the charge density should be conserved.

1.2.2 Finite difference

The finite difference operators is tested by setting up a test field based on a polynomial on which the operator should give an exact answer for. For example if we have a quantity $f(x) = 3x$, then a first order finite difference scheme will give $\hat{\nabla} f(x) = 3$.

1.2.3 Multigrid and Grid structure

We want the basic grid to be available through a grid datastructure and the stack of grids stored in the multigrid structure. To ensure that this will still work through changes in the the structs there is a simple unittest that uses a grid struct to set up a field, then it is changed in the multigrid struct. Then it confirms that the values in the grid struct is also changed.

1.2.4 Edge Operations

In the communication between the subdomains, as well as in the treatment of boundary conditions, there is a group of functions dealing with slice operations. These are tested by putting assigning each subdomain different constant values, then different slice operations is performed.

1.3 Scaling

In this section we investigate the performance of the solver and different scaling measurements. We are interested in both how well the solver performs on a larger number of processors, as well as the perfomance impact of the different parameters in the solver.

We want to obtain a better understanding of how well the