Appendix A

Grid Structs and Partitioning

A.1 Data structures

The fields and quantities in our PiC model is discretized on a threedimensional grid and comes to use several places in the method. In the multigrid calculation we also have a use for several grids of varying spatial coarseness. So it will be useful for us to organize the data so we have the grid stored as an independent structure available for the program, while the multigrid part uses an extended version where it also has access to the different coarser subgrids. There will also be several types of grid, all with the same specifications, but storing different quantities, so we will use a grid template that all the grids of the same size shares, while we will have a seperate struct containing the quantities in the grid and the specifications.

A.2 Domain partitioning

As earlier discussed, in **??** the physical domain covered by our model will be divided onto several processors that each take care of a subdomain. The subdomains are dependent on each other and we need some communication between them, which we solve by letting each subdomain also store the edge of the neighboring subdomain. Depending on the boundary conditions it could also be useful to store an extra set of values on the outer domain boundary as well, which will be called ghost points, N_G. The extra grid points due the the overlap between the subdomains we will call overlap points, N_O. Let us for simplicity sake consider a regular domain, with equal extent in all dimensions, with N grid points per dimension, d and consider how many grid values we need to store as a singular domain and the grid values needed when it is divided amongst several processors, a 2 dimensional case is depicted in fig. A.1.

A.2.1 Singular domain

In the case where the whole domain is worked on by one process we need N^d to store the values on the grid representing the physical problem, in addition we see that we also need to store values for the ghost points along the domain boundary. Given that we have one layer of ghost points on all the boundaries, and there is 2 boundaries per dimension, the total number of ghost points is given by $N_G = 2dN$. Since there is only 1 domain we don't need to account for any overlap between subdomains and the total grid points we need to store is:

$$N_{Tot} = N^d + N_G + N_O = N^d + 2dN^{d-1} \tag{A.1}$$

For the 2 dimensional case, in fig. A.1, that adds up to $N_{Tot} = 8^2 + 2 \times 4 \times 8 = 128$.

A.2.2 Several subdomains

In the case where we introduce several subdomain, in addition to storing the grid values and the ghost points we also need to store an overlap between the subdomains. If we take our whole domain Ω and divide in up into several small domains Ω_S, the smaller domains only takes a subsection of the grid points. For simplicity case, and equal load on processors, we let the subdomains as well be regular, with the whole domain being a multiple of the subdomains. Our whole domain has N grid points in each direction, if we then divide that domain into $\#\Omega$ domains, then each of those subdomains will have $N_S = N^d/\#\Omega$ grid points. Each of those subdomains will also need values representing the ghost points and overlap from the neighboring nodes. A boundary of a subdomain will either have overlap points, or ghost points, not both at the same time so for each boundary we need to 1 layer, N_S^{d-1}. Each subdomain will have 2 boundaries per dimension since we have regular subdomains. The total number of grid points needed per subdomain is then

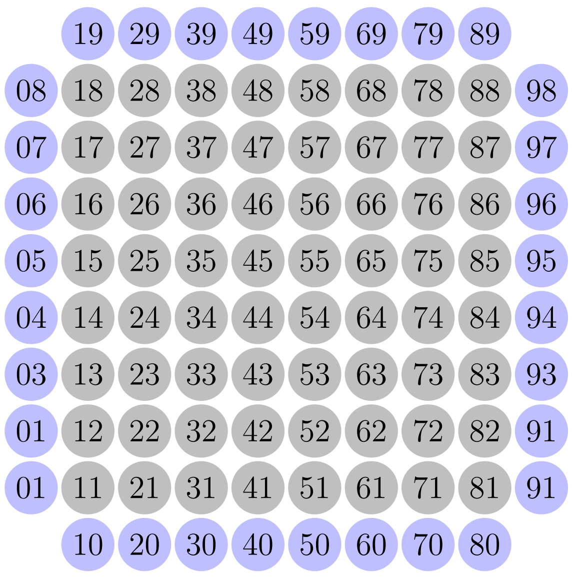
$$N_{Tot,S} = N_S^d + (N_G + N_O) = N_S^d + 2dN_S^{d-1} \tag{A.2}$$

while the total number of grid points is

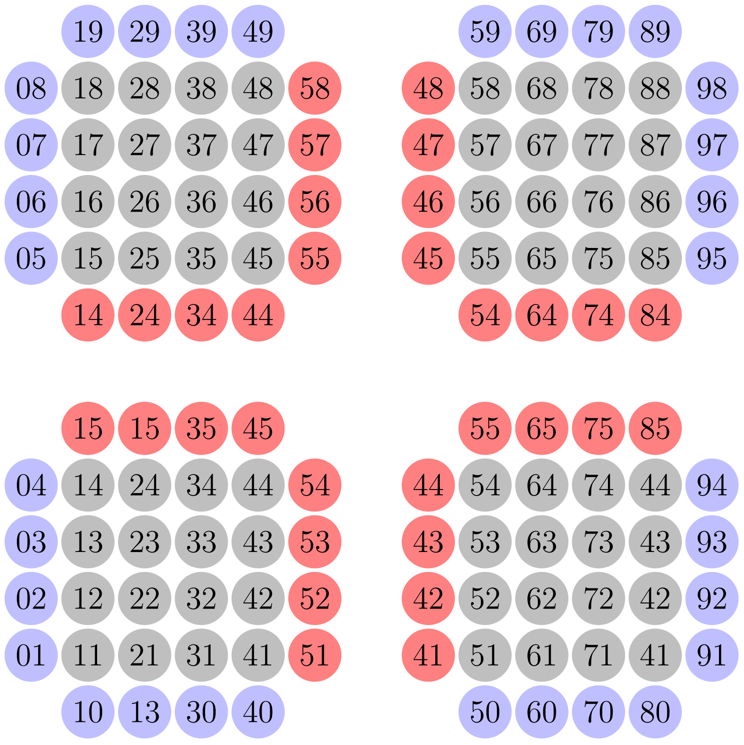
$$N_{Tot} = \#\Omega N_{Tot,S} \tag{A.3}$$

For the 2 dimensional case discussed earlier we need $N_{Tot,S} = 4^2 + 2 \times 2 \times 4^1 = 32$.

Since the effect of the subdomain boundaries increase the coarser the grid is we should not let the coarsest multigrid level be to small. We also don't need the spatial extent of the grid to be equal on all sides, but it was done here to keep the computations simple.



(a) The grid points needed for an 8 × 8 domain.



(b) The 8 × 8 grid divided into 4 subdomains

Figure A.1: Each circle in the figures represents 1 grid point, and the first number is the column while the second is the row. The grey colour represents physical space the computational node works on, the blue color is the outer grid points for boundary conditions and the red colour is the overlapping grid points.

Appendix B

Verification and Testing

The multigrid method has several different steps in the algorithm, as a developmental help and to ensure that the program works correctly during as many different conditions as possible we want to test the whole code, as well as the constituent parts where possible. The method is quite modular and several parts of it can be tested alone. The GS-RB, used for smoothing, can be independently tested, since on it's own it converges to a solution, just at a higher computational cost than the multigrid method. To test it we will use an initial density field with a length between the grid steps that results in an exact answer.