Master Thesis

A Parallel Multigrid Solver for Beam Dynamics

ETH Zurich

Yves Ineichen (iyves@student.ethz.ch)
Supervisor: Andreas Adelmann (andreas.adelmann@psi.ch)
Supervising Professor: Prof. Dr. P. Arbenz (arbenz@inf.ethz.ch)

August 17, 2008

Abstract

The calculation of space-charges is the computationally most expensive task in current state-of-the-art particle accelerator simulation codes such as OPAL, the Object Oriented Parallel Accelerator Library, at PSI. Most notably direct solvers (FFT) and iterative solvers are used to solve the anisotropic Poisson's equation for the electrostatic potential on irregular bounded three-dimensional domains. In this thesis we implemented a smoothed aggregation based algebraic multigrid preconditioner for use with iterative solvers like CG or BiCGstab.

The innovation of this thesis lies in discretizing the problem for quite arbitrary irregular domains in three space dimensions and still be able to apply a scalable fast iterative solver. The domains can be specified with common design software. A preconditioned iterative solver has been integrated into the OPAL tracking code. It can reuse solutions of previous time-steps to improve convergence. We show that it is possible to take arbitrary geometries of beam pipe elements into account when computing space-charge forces. Furthermore the proposed and implemented work-flow, from design to actual simulation, is very convenient for the user.

Contents

1 Introduction and Motivation **1**
- 1.1 Related Work . . . 2
- 1.2 Thesis Outline . . . 2

2 Solver **4**
- 2.1 Physical Model . . . 4
- 2.2 Conjugate Gradient Algorithm . . . 6
- 2.3 A Short Introduction to Multigrid Methods . . . 7
- 2.4 A Multigrid Preconditioner for Space Charge Calculations . . . 8

3 Boundary Conditions **13**
- 3.1 Boundary Conditions on Rectangular Domains . . . 14
- 3.2 O(h) Approach . . . 14
- 3.3 Shortley-Weller . . . 15
- 3.4 Elliptic Beam Pipes . . . 16
- 3.5 Arbitrary Geometries as Beam Pipes . . . 17
- 3.6 Implementation . . . 19

4 Numerical Results **22**
- 4.1 Elliptic Irregular Domains . . . 23
- 4.2 Arbitrary Irregular Domains . . . 26

5 Conclusion **29**
- 5.1 Problems . . . 30
- 5.2 Further Work . . . 30

References **32**

Appendices

A Proof Sketch: Symmetrize Anisotropic Discretization on a Rectangular Domain **34**
- A.1 3D Poisson Discretization with Kronecker Products . . . 34

1 Introduction and Motivation

Object Oriented Particle Accelerator Library (OPAL[1]) is a C++ framework for general parallel particle accelerator simulations. It includes various beam line element descriptions and methods for single and multiparticle optics.

The simulation framework also takes into account self or space-charge effects by solving the arising anisotropic Poisson partial differential equation (PDE) for the electrostatic potential. Currently OPAL calculates space-charge fields in the rest frame by a convolution of the charge density with the appropriate Green function in every time-step. The convolutions are implemented with means of fast Fourier transforms (FFT) using an integrated Green function as described in [1]. This approach (being a direct solver) has the "disadvantage" of throwing away all information on the potential computed in previous time-steps which is assumed to be most of the time very close to the solution of the current time-step. When using a Multigrid method (in fact any iterative method could be applied) the information of previous time-steps can be reused as an initial guess for the solution to improve the convergence of the iteration process.
Space-charge calculations in OPAL are responsible for roughly 50% of the total simulation time. With the new improvements our hopes are to reduce the time to solution for space-charge field calculations with the help of a Multigrid solver. Another important facet is scalability. For very detailed simulations OPAL can make use of parallel computing environments to speedup the simulation. Therefore it is important that our Multigrid solver scale well.
The Multigrid solver will be implemented with the aid of Trilinos and its additional packages. This library provides us with a fully parallel Multigrid preconditioner and CG solver. The resulting solver will be integrated in OPAL .

Another improvement to the current situation (FFT space-charge solver with open boundary conditions) concerns the boundary conditions. We hope to improve the accuracy of space-charge simulation by applying a perfect electric conductor (PEC) boundary conditions on arbitrary beam pipe geometries. To our knowledge the only other simulation code that can handle real geometries as boundary of the beam pipe is called WARP [2]. It will be central to offer the OPAL user a seamless integration of design and actual execution of the simulation. Therefore beam pipe geometries for specific beam elements (i.e. solenoid) can be preprocessed and passed to OPAL directly from STEP (a common industry standard for CAD like design tools) files.
The straight forward discretization of these irregular domains results in a non-symmetric system. We tried to circumvent having to solve non-symmetric systems by scaling the non-symmetric system.

[1] http://amas.web.psi.ch/tools/OPAL/index.html

1.1 Related Work

The master thesis [3] describes a similar space-charge solver for elliptical beam pipes using Multigrid methods. To solve the non-symmetric system a BiCGStab solver is applied. We extend this work by allowing arbitrary beam pipe geometries as domain boundaries. Furthermore we show results for applying a conjugate gradient (CG) solver to solve this non-symmetric problem.

In another relevant paper [4] the Poisson problem is solved on an irregular bounded three dimensional domain. They approximate discrete Laplacian operator with quadratic Shortley-Weller extrapolation. It is shown that the error in the gradient of the solution is of second order when applying Shortley-Weller. In contrast to our work, they use adaptive mesh refinement methods to further reduce the operator size.

1.2 Thesis Outline

We are giving in Chapter 2 an overview of the components of the proposed preconditioned CG solver. The chapter is concluded by a summary of parameters for ML and implementation details concerning the solver.

In Chapter 3 we introduce various discretization approaches and boundary condition for regular and irregular domains. Additionally we focus on the question if the discretization matrix can be constructed to be symmetric for all discretized boundaries. In the last section some implementation details are mentioned.

For all discretization methods we provide a comparison of the numerical solution with the analytical, convergence studies and speedup analysis in Chapter 4. Finally this thesis is concluded in Chapter 5 also providing an insight into possible extensions.

Acknowledgments

This master thesis would never have been possible without the constant support of many people. Much credit goes to:

> Andreas Adelmann, my supervisor at PSI, who always could motivate me during rough times and provided many fruitful discussions about just anything related to this thesis.
> Peter Arbenz, my supervising professor at ETH, who always found time to answer my questions in a very competent and in-depth way.
> Timothy Stitt, from the CSCS in Manno, who was very competent and helpful with all the problems we encountered regarding compiling OPAL , Trilinos etc. on the cluster.
> Benedikt Oswald and Achim Gsell, for contributing H5FED . All members of the AMAS group, who contributed to a very stimulated work environment at PSI.

Since this thesis marks the end of a very interesting and intense time at ETH I also would like to express my gratitude to people who helped me along the way:

Tomas Dikk, Mark Green, Boris Jenni, Roman Kerscher, Daniel Kuettel and Simon Meier, who made life besides Computer Science equally interesting and worth living.
My parents Eva and Beat for providing me with the possibility and constant support to study at ETH.
My brother Christian, who always listened to my problems and provided helpful support in any possible way.

2 Solver

In this chapter we will start by giving an overview of the underlaying physical model and computational method. In consecutive sections we will introduce the solver used to tackle the problem. The last section of this chapter will cover implementation details about the solvers implementation and integration in OPAL .

2.1 Physical Model

Particle tracking calculations are performed within OPAL framework by considering M macro- or even real particles with mass m and charge q. Their trajectories are computed by means of the relativistic equations of motion given by:

$$\begin{aligned} \frac{\gamma_i \mathbf{v}_i}{dt} &= \frac{q}{m}(\mathbf{E}_i + \mathbf{v}_i \times \mathbf{B}_i) \\ \frac{\mathbf{x}_i}{dt} = \mathbf{v}_i &= \frac{\gamma_i \mathbf{v}_i}{\sqrt{\gamma_i \mathbf{v}_i^2/c + 1}}, \ \forall i = 1 \dots M. \end{aligned} \tag{2.1.1}$$

For each of the i particles initial position $\mathbf{x}_i^0$, velocity $\mathbf{v}_i^0$ must be given together with initial field values $\mathbf{E}_i^0$ and $\mathbf{B}_i^0$. The electric and magnetic fields of OPAL are a superposition of external, self and wake fields:

$$\begin{aligned} \mathbf{E} &= \mathbf{E}_{ext} + \mathbf{E}_{self} + \mathbf{E}_{wake} \\ \mathbf{B} &= \mathbf{B}_{ext} + \mathbf{B}_{self} + \mathbf{B}_{wake} \end{aligned} \tag{2.1.2}$$

where the subscript i has been omitted for readability.

The evolution of beam's distribution function $f(\mathbf{x}, \mathbf{v}, t)$ can be expressed by the collisionless Vlasov equation:

$$\frac{df}{dt} = \partial_t f + \mathbf{v} \cdot \nabla_x f + \frac{q}{m}(\mathbf{E}(\mathbf{x}, t) + \mathbf{v} \times \mathbf{B}(\mathbf{x}, t)) \cdot \nabla_v f = 0, \tag{2.1.3}$$

where $\mathbf{E}$ and $\mathbf{B}$ are defined in (2.1.2). When combining (2.1.3) with Maxwell equations we get the collisionless Valsov-Maxwell equation.

Self or space-charge fields of an accelerated bunch have both an electric and a magnetic field component. An intuitive picture is that the electric part is caused by Coulomb repulsion, where the magnetic part is caused by the fact that moving particles represent a current. A convenient numerical method to calculate these fields is to determine them in a frame traveling with the same velocity as the charged particle bunch (rest frame). In this frame, the magnetic fields are negligible as long as the velocity differences are below the speed of light.

For the numerical solution of this problem we note that due to the Lorentz transformation one of the dimensions (z) is stretched by the relativistic factor γ. This anisotropy has impact of the numerical scheme as we will see later.

2.1.1 Electrostatic Case

In case of non-relativistic motion in the bunch frame, the full set of Maxwell equations are reduced to:

$$\nabla \times \mathbf{E} + \frac{\partial \mathbf{B}}{\partial \mathbf{t}} = \mathbf{0}, \tag{2.1.4}$$

$$\nabla \cdot \mathbf{B} = 0. \tag{2.1.5}$$

The self-consistent Coulomb potential $\phi(\mathbf{x}; t)$ can be expressed in terms of the charge density $\rho(\mathbf{x}; t)$, which is proportional to the particle density

$$n(\mathbf{x}; t) = \int d\mathbf{v} f(\mathbf{x}, \mathbf{v}; t), \quad \rho(\mathbf{x}, \mathbf{v}; t) = en(\mathbf{x}; t) \tag{2.1.6}$$

and we can write:

$$\phi(\mathbf{x}; t) = \int_\Omega d\mathbf{x}' \frac{\rho(\mathbf{x}'; \mathbf{t})}{|\mathbf{x} - \mathbf{x}'|}. \tag{2.1.7}$$

The self-fields due to space-charge are coupled with Poisson's equation

$$\nabla \cdot \mathbf{E} = \int f(\mathbf{x}, \mathbf{v}; t) d\mathbf{v}. \tag{2.1.8}$$

The resulting electric field in the rest frame is described by the electrostatic potential ϕ in the form of $\mathbf{E} = -\nabla\phi$. This electric field can be transformed to yield both the electric and the magnetic fields required in (2.1.1) by means of a Lorentz transformation.

Computational Method

Widely used methods for the calculation of these space-charge fields are the particle-mesh method and the particle-particle method. The particle-mesh method, based on solving Poisson's equation for the electrostatic potential, is typically much faster than the particle-particle method. Furthermore, it provides better numerical results for sufficiently 'smooth' distributed particles.

Lets denote $\Omega \subset \mathbb{R}^3$ the simply connected computational domain and $\Gamma = \Gamma_1 \cup \Gamma_2$, the boundary of Ω. On the two disjoint boundary surfaces: $\Gamma_1 \cap \Gamma_2 = \emptyset$ we normally apply weighted combination of Dirichlet boundary conditions and Neumann boundary conditions, the so called Robin (open) boundary conditions. and Dirichlet (perfect electric conductor PEC) boundary conditions. We now can state the equation to solve:

$$\begin{aligned} \nabla^2\phi = -\frac{\rho}{\varepsilon_0}, \text{ in } \Omega \subset \mathbb{R}^3, \\ \phi = 0, \text{ on } \Gamma_1 \\ \frac{\partial\phi}{\partial\mathbf{n}} + \frac{1}{d}\phi = 0, \text{ on } \Gamma_2, \end{aligned} \tag{2.1.9}$$

with ε_0 denotes the dielectric constant and d is the distance of the bunches centroid to the boundary.

We can solve (2.1.9) by using a second order finite difference scheme which leads to a set of linear equations,

$$Lu = f, \tag{2.1.10}$$

where u denotes the vector of unknown values of the potential and f the vector of the given charge density at the grid points and the operator L is the discretization of the Laplacian. Further discretization discussions follow in Chapter 3.

2.2 Conjugate Gradient Algorithm

For symmetric positive definite systems (spd), the Conjugate Gradient (CG) [5] method provides a fast and memory efficient solver. For a n-dimensional problem the CG scheme converges in n iterations steps (apart from numerical instabilities and roundoff errors).

The basic idea is to reformulates the problem as a minimization problem of a quadratic functional. Once in this form, the problem can be solved by an iterative search for the minimum in a given direction. For the CG method these directions are pairwise chosen to be conjugated (or A-orthogonal). In each iteration step a new search directions "orthogonal" to all previous search directions is created.

A proper implementation of a CG solver only needs memory for the spd matrix and 4 additional vectors of length n.

2.2.1 Preconditioning

To improve the convergence properties of CG (basically any iterative scheme) methods one can try to improve the condition number of the matrix by preconditioning the system. This is due to the fact that usually the convergence of an iterative solver degrades when the condition number of a system increases. A preconditioned system has the following form

$$\underbrace{\mathbf{M}^{-1}\mathbf{A}}_{\overline{\mathbf{A}}}\mathbf{x} = \underbrace{\mathbf{M}^{-1}\mathbf{b}}_{\overline{\mathbf{b}}},$$

where $\mathbf{M}^{-1}$ is called preconditioner. When the preconditioner is well chosen the new system has a lower condition number and is still symmetric positive definite.

2.2.2 Iterative Solvers for Non-Symmetric Systems

In the case that the discretized system is non-symmetric or symmetric and indefinite but still sparse we can apply other (more or less efficient) iterative schemes.
The non-symmetry property implies that the problem at hand cannot be reformulated as a minimization of a quadratic functional. For non-symmetric problems we therefore apply the stabilized Biconjugate Gradient (BiCGStab) method [6]. Basically BiCGStab is the combination of the Biconjugate Gradient method (BiCG) and

the GMRES method [7]. Repeated application of the GMRES method minimize the residual locally resulting in a smoother convergence behaviour of the method compared to CGS [8] and BiCG. On the downside combining BiCG with GMRES introduces additional breakdown possibilities, namely if GMRES does not converge.

In general both BiCGStab and CGS converge equally fast, although there are problems where one of the two performs better than the other. When considering computational costs we note BiCGStab requires two matrix-vector products in contrast to CG using only one.

2.3 A Short Introduction to Multigrid Methods

In this section we will give a short overview of Multigrid methods. Additional information can be found in [9] and [10].
Multigrid methods are based on the observation that a smooth error on a fine grid can be well approximated on a coarser grid. When this coarser grid is chosen to be a sufficient factor smaller (say at leaste by 2 in every direction) than the fine grid the resulting problem is smaller and thus cheaper to solve. Obviously one can continue to coarsen the grid until it is small enough to be able to cheaply solve the problem with a direct solver. This observation suggests an algorithm similar to the one shown in Algorithm below.

Algorithm 1 Multigrid V-cycle Algorithm

1: **procedure** MultiGridSolve(A_l, b_l, x_l, l)
2: **if** l = maxLevel-1 **then**
3: DirectSolve $A_l\mathbf{x}_l = \mathbf{b}_l$
4: **else**
5: $\mathbf{x}_l \leftarrow S_l^{pre}(A_l, \mathbf{b}_l, 0)$
6: $\mathbf{r}_l \leftarrow \mathbf{b}_l$ - $A_l\mathbf{x}_l$ {calculate residual}
7: $\mathbf{b}_{l+1} \leftarrow R_l\mathbf{r}_l$ {Restriction}
8: $\mathbf{v}_{l+1} \leftarrow \mathbf{0}$
9: MultiGridSolve(A_{l+1}, $\mathbf{b}_{l+1}$, $\mathbf{v}_{l+1}$, $l+1$)
10: $\mathbf{x}_l \leftarrow \mathbf{x}_l + P_l\mathbf{v}_{l+1}$ {coarse grid correction}
11: $\mathbf{x}_l \leftarrow S_l^{post}(A_l, \mathbf{b}_l, \mathbf{x}_l)$
12: **end if**
13: **end procedure**

The procedure starts on the finest level ($l = 0$) and repeatedly coarsens the grid until the coarsest level is reached (maxLevel) where the direct coarse level solver can be used to solve the problem at hand. In all other levels l the algorithm starts by presmoothing S_l^{pre} the problem to damp high frequency components of the error (line 5). Subsequently the fine grid on level l can be restricted with the restriction operator R_l to a coarser grid on level $l+1$ (line 7). This essentially "transfers" the low frequency components on the fine grid to high frequency components on the coarse grid. After the recursion has reached the coarsest level and used the direct solver to solve the

coarse level problem the solution can be prolongated back to a finer grid. This is achieved with the prolongation operator P_l (line 10). Often a postsmoother S_l^{post} is used to remove artifacts caused by the prolongation operator. Usually these operators (for every level l) are defined in a setup phase preceding the execution of the actual Multigrid algorithm. Lastly A_l denotes the matrix of the discretized system in level l.

The described Multigrid algorithm has order $O(n)$ and its convergence should be independent of the grid-spacings. Multigrid methods work well for symmetric positive definite systems obtained from elliptic PDE's.

2.4 A Multigrid Preconditioner for Space Charge Calculations

As described in [11], Multigrid methods provide a less sensitive convergence behaviour to parameter changes when used as a preconditioner applied to an iterative solver. Additionally the algebraic Multigrid (AMG) performs better when used as preconditioner opposed the application as stand-alone solver. This provides a sufficient motivation to use an AMG (described in this section) as preconditioner for an iterative solver (CG, BiCGStab).

Independent of the application of Multigrid methods the performance profoundly depends on the choices of the operators (introduced in the previous section) and their interplay. To ensure that the resulting preconditioner is symmetric we use the same pre- and postsmoother S_l and the restriction operator is chosen to be the transpose of the prolongation operator $R_l = P_l^T$. This leaves us with two operators (P_l and S_l) that have to be defined for every level.

The prolongation and restriction operator can be defined in various ways. When using a Geometric Multigrid (GMG) the restriction and prolongation operator is defined by using geometric information about the grid. Another approach is to solely base these operators on the algebraic information contained in the system of equations without considering the underlaying grid hierarchy. This describes the basic property of operators utilized in AMG's. The independence of the underlaying geometric grid makes the AMG very robust in the presence of complicated structured or unstructured grids. Since we are dealing with an anisotropic grid this property makes the AMG a valuable choice, saving us a great deal of trouble and additional effort. Normally when applying a GMG to an anisotropic grid special transfer operators are needed (i.e. semi-coarsening). Aside the more costly transfer operators we will experience that the standard smoothers applied to an anisotropic problem will perform badly. The reason for the bad performance is that the smoother has no averaging effect in directions that are anisotropic (or extremely stretched). Therefore the smoothing effect in these direction is inexistent or very weak causing the Multigrid method to converge very slowly. In contrast AMG's work efficiently on all error components. In comparison the convergence rate of an AMG is about the same as for a GMG.

Prolongation Operator P_l Aggregation based methods cluster the fine grid unknowns to aggregates (of a specific form, size, etc.) as representation for the unknowns on the coarse grid. First the discretization matrix A_l is converted into a graph G_l. Each vertex of G_l is then assigned to one aggregate of the disjoint aggregate set. In a next step a tentative prolongation operator matrix is formed where matrix rows correspond to vertices and matrix columns to aggregates. A matrix entry (i, j) has a value of 1 if the it^{th} vertex is contained in j^{th} aggregate and 0 otherwise. This prolongation operator basically corresponds to a piecewise constant interpolation operation. To improve the robustness one can additionally smooth (normally with a damped Jacobi smoother) the tentative prolongation operator. In general applying the smoother results in better interpolation properties opposed to the piecewise constant polynomials. The application of the smoother is beneficial for spd problems arising from elliptic problems improving convergence. In [12] various strategies are proposed how this process can be parallelized. One of the simplest parallel method is to let each processor aggregate its portion of the grid. This method is called "decoupled" since every processor act independently of others. Usually the aggregates have a size of 3^d in d dimensions. For our 3D problems aggregates therefore have a size of $3 \times 3 \times 3 = 27$ forming small cubic aggregates with no overlapping elements.

Smoothing Operator S_l As advised in [13] we choose a Chebyshev polynomial smoother. The choice is motivated by the result that polynomial smoothers perform better than (i.e.) Gauss-Seidel smoothers for a parallel solver. Advantages are i.e. that polynomial smoother do not need special matrix kernels and formats for optimal performance and generally polynomial methods can profit of architecture optimized matrix vector products. Additionally polynomial smoothers are much easier to parallelize than i.e. Gauss-Seidel methods.

Coarse Level Solver The coarse level equation is solved by a direct LU based solver. In particular Trilinos (the software framework employed) provides a package (Amesos) offering various direct solvers. The employed solver (KLU) ships the coarse level problem to node 0 and solves it by means of a LU decomposition. Once the solution has been calculated it is broadcast to all nodes.

Implementation

The Multigrid preconditioner and solver is implemented with help of the Trilinos [14] library. Trilinos provides state-of-the-art tools for numerical computation in various packages. Amongst others there is a package called ML [15] providing a Multigrid preconditioner. We utilized this package to create the AMG preconditioner using a smoothed aggregation-based transfer operator. A summary of the essential AMG preconditioner parameters discussed in the previous section are presented in Table 2.1. Currently the solver is parallelized in z direction.

For the embedding in the physical simulation code we utilized Independent Parallel

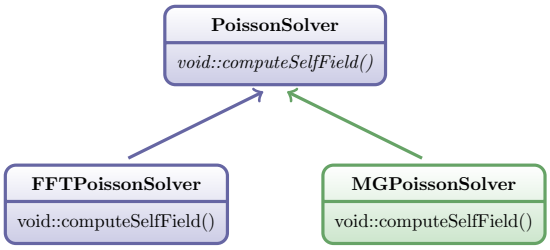

Figure 2.4.1: Inheritance diagram for the solver

Particle Layer[1] (IPPL). This library is an object-oriented framework for particle based applications in computational science requiring high-performance parallel computers, hence it is used by OPAL to a great extent to handle particles, fields, operators on grids and fields and communication. In the context of this thesis IPPL is only relevant because OPAL also uses IPPL to represent the grid with the interpolated charges of the particles.

OPAL 's object oriented approach makes the integration of the new solver as easy as possible. As shown in Figure 2.4.1 (in greenish color) the Multigrid solver could simply be placed in a new class MGPoissonSolver that is derived from PoissonSolver. The interface defined by PoissonSolver provides an abstract method for calculating the field induced by space-charge effects. This is the central method, that had to be implemented to enable OPAL to use the MG based solver. An overview of the various steps the solver performs is given in Figure 2.4.2. We start by redistributing the solution of the last time-step (if there was a preceding time-step) because the particles could have been redistributed in the mean time (OPAL redistributes particles if needed after a time-step has been simulated). After we ensured, that the solver and IPPL have the same domain decomposition the solver builds the discretization matrix with the

[1] http://amas.web.psi.ch/tools/IPPL

name	value
max levels	5
increasing or decreasing	decreasing
prec type	MGW
aggregation: type	Uncoupled
smoother: type	Chebyshev
smoother: sweeps	3
smoother: pre or post	both
coarse: type	Amesos-KLU

Table 2.1: Parameters for multilevel preconditioner

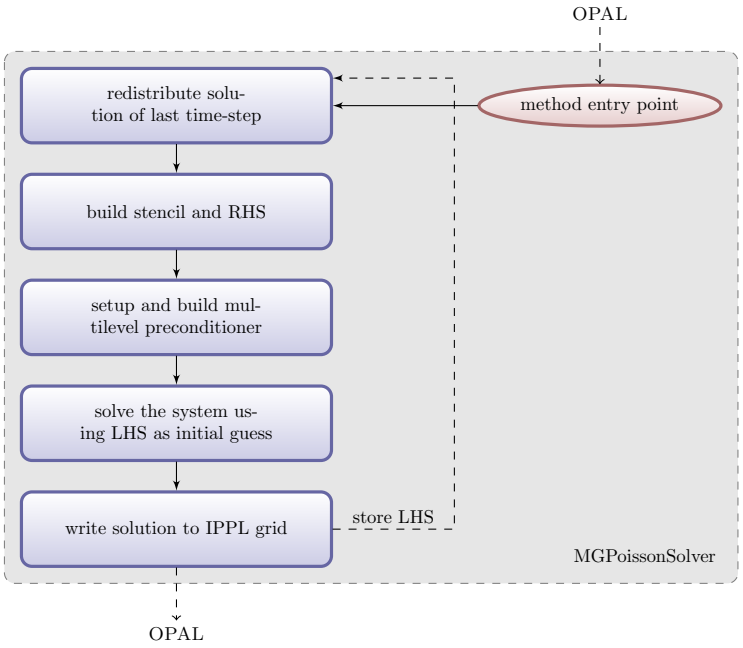

Figure 2.4.2: Flow diagram for the MGPoissonSolver

desired boundary conditions and stores the right hand side into an Epetra_Vector. After the ML package has setup and created a preconditioner we finally are able to solve the preconditioned system with Aztecoo's solvers. The last step consists of writing the solution back to the IPPL grid. The transfer operations between the IPPL grid and an Epetra_Vector is discussed in the next section.

Interface Between OPAL and the Solver

From the OPAL point of view the solver can be interpreted as a black box (see Figure 2.4.2). This black box requires a distributed scalar field providing the on the grid interpolated charges as input and returns the electric potential as output. On one end of the interface to the solver we have an OPAL grid which is implemented with the help of the IPPL and on the other end Trilinos expects an Epetra_Vector. The interface between OPAL and the solver therefore consists of a method that converts an IPPL field to an Epetra_Vector and vice versa. A minor difficulty is to assign the same domain decomposition to the Epetra_Vector as present in the IPPL (IPPL balances the grid) grid to prevent any data communication between nodes. Trilinos handles the domain decomposition with an Epetra_Map. IPPL provides us with local indices and domain lengths for every dimension. This makes the creation of such an Epetra_Map very comfortable: we traverse all local indices on a given CPU and store for each

index triple (x, y, z) the global index in the Epetra_Map.

In Listing 2.4 we show such a interface for a simple rectangular domains. Local index sets are provided by IPPL through the index set localidx passed to the function. This set the is used to calculate the total number of elements on the current processor and we can loop over all the local indices creating the mapping between local indices to global indices.
For elliptic and arbitrary domains the corresponding methods look similar although the calculation of the global index (in line 11) is a bit more challenging since in general the internal in general are non-regularly numbered, depending on the boundary geometry.

2.4 IPPL to Epetra_Map

```
1  inline Epetra_Map* IPPLToMap3D(NDIndex<3> localidx)
2  {
3
4    int NumMyElements = localidx[0].length()*localidx[1].length()*localidx[2].length
         ();
5    vector<int> MyGlobalElements(NumMyElements);
6    int idx = 0;
7
8    for(int x = localidx[0].first(); x <= localidx[0].last(); x++)
9      for(int y = localidx[1].first(); y <= localidx[1].last(); y++)
10       for(int z = localidx[2].first(); z <= localidx[2].last(); z++)
11         MyGlobalElements[idx++] = x + y * nr_m[0] + z * (nr_m[0] * nr_m[1]);
12
13   return(new Epetra_Map (-1, NumMyElements, &MyGlobalElements[0], 0, Comm));
14
15 }
```

3 Boundary Conditions

In this chapter we discuss how boundary conditions are handled. After a short introduction of the boundary conditions for Multigrid methods we will look at problems arising when using non-rectangular domains and how beam pipe geometries can be specified.

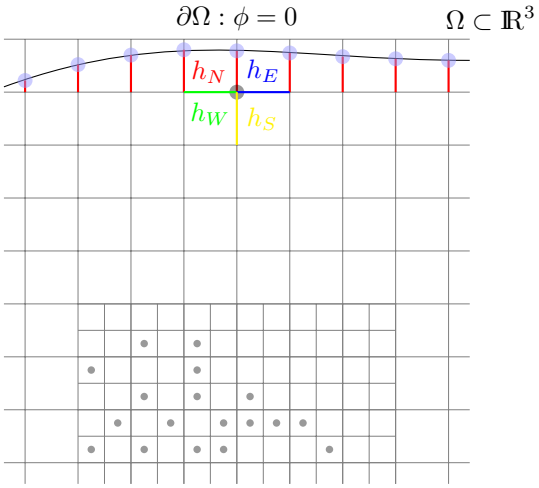

Figure 3.0.1: Irregular bounded domain

Throughout this chapter we will impose a perfect electric conductor (PEC) boundary condition in transversal direction on the surface of the beam pipe and "open" boundary conditions in longitudinal direction. Perfect electric conductors possess the property that the tangential vector component in the dielectric is zero. We can model this by defining the boundary in transverse direction as a Dirichlet boundary condition for the potential ϕ on the beam pipe surface Γ_1,

$$\phi = 0, \text{ on } \Gamma_1.$$

This boundary can be irregular as schematically shown in Figure 3.0.1. We will discuss two different approaches how the problem is discretized near the boundary (see Section

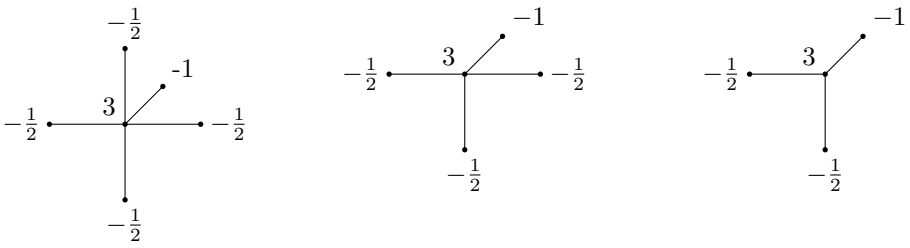

Figure 3.1.1: Discretization for a rectangular domain with Neumann boundary conditions on $z = 0$. LEFT: surface, MIDDLE: edge, RIGHT: corner.

3.2 and 3.3). "Open" boundary conditions

$$\frac{\partial \phi}{\partial \mathbf{n}} + \frac{1}{d}\phi = 0, \text{ on } \Gamma_2$$

are used to simulate the open beam pipe in the longitudinal direction where d denotes the distance of the bunches centroid to the boundary.

As we mentioned in Section 2.2 it would be very profitable to get symmetric positive definite discretization matrices. For every domain that is investigated we try to argue if the symmetry property can be preserved.

3.1 Boundary Conditions on Rectangular Domains

The first boundary condition we implemented was merely to be able to test the solver. To "simulate" the open boundary conditions in transverse direction, we enlarge the grid by a significant factor and then apply the PEC boundary condition on this new boundary. By doing this we try to approximate the 0-potential at infinity, essentially hoping to get a similar boundary behaviour as when solving space-charge forces with the FFT solver. Figure 3.1.1 shows the discretization for surfaces, edges and corners. The Neumann boundary condition is scaled in such a way that the discretization matrix remains symmetric positive definite, enabling us to use the CG method.

3.2 O(h) Approach

The key idea of this approach is to only consider grid points inside the domain neglecting the distance to the domain boundary. The 2D discretization has the following from:

$$(l_w + l_s + l_e + l_n)u_p - l_n u_n - l_w u_w - l_s u_s - l_e u_e = f_p, \tag{3.2.1}$$

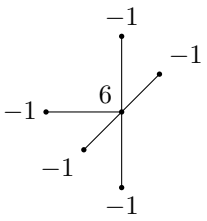

Figure 3.2.1: Discretization for the $O(h)$ condition where the east grid point lies outside of the domain.

where $l_i = h_i^{-1}$. If i.e. the east grid point of the stencil is outside the domain (as shown in Figure 3.2.1) we set

$$u_e = g = 0.$$

The value g on the boundary is then to be added the RHS. In our case these terms vanish. The resulting discretization matrix will still be symmetric. In Appendix A we give a proof that the symmetry is preserved, even for anisotropic grids.

It is clear that this approach has a discretization accuracy of $O(h)$ since we are neglecting the distance to the irregular boundary point which can be at most be h.

3.3 Shortley-Weller

As discussed in [16] the Shortley-Weller approximation provides a good approximation for irregular boundary points with a discretization error of $O(h^2)$. This approximation can be deduced by considering the second derivative of a function u at a point x with

$$-u''(x) \approx \frac{2}{h_w + h_e}\left(\frac{u(x) - u(x - h_w)}{h_w} - \frac{u(x + h_e) - u(x)}{h_e}\right)$$

which can be simplified to

$$-u''(x) \approx -\frac{2}{(h_w + h_e)h_e}u(x + h_e) + \frac{2}{h_w h_e}u(x) - \frac{2}{(h_w + h_e)h_w}u(x - h_w). \quad (3.3.1)$$

For the 2D case the following approximation can be used:

$$\begin{aligned} -u''(x,y) \approx &-\frac{2}{(h_w + h_e)h_e}u(x + h_e) + \frac{2}{h_w h_e}u(x) - \frac{2}{(h_w + h_e)h_w}u(x - h_w) \\ &-\frac{2}{(h_n + h_s)h_n}u(y + h_n) + \frac{2}{h_n h_s}u(y) - \frac{2}{(h_n + h_s)h_s}u(y - h_s) \end{aligned} \quad (3.3.2)$$

3.4 Elliptic Beam Pipes

Elliptic beam pipes can simply be specified by providing the lengths of the semi-major and semi-minor axis. The irregular boundary points then can be easily computed by using the equation for the ellipse and intersect it with x and y grid lines in one plane.

Using the $O(h)$ approach for the discretization of grid points near the domain boundary is straightforward. The symmetry property is preserved as shown in the proof-sketch in Appendix A.

When applying the Shortley-Weller approximation at the boundary the resulting discretization matrix is non-symmetric. At first we thought we could use the same arguments stated in Appendix A after masking out grid points outside of the domain. Admittedly a problem arose concerning the different distance to the boundary for the last grid point in the domain for every grid-line. This can be seen when applying the scaling as proposed in Appendix A to the 2D discretization for the Shortley-Weller approximation given in the previous section.

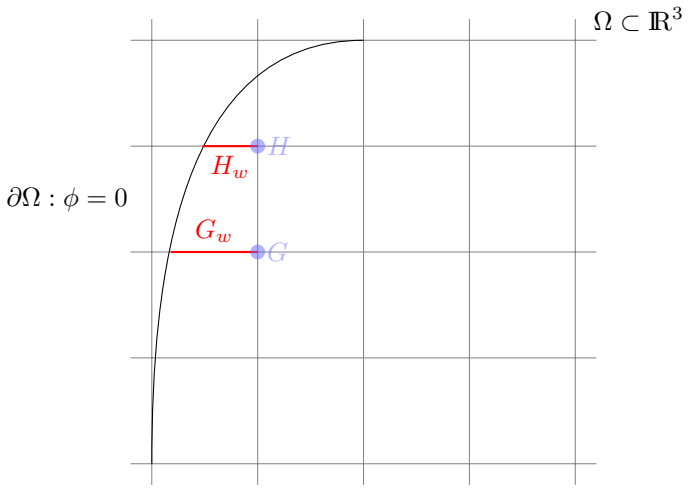

Figure 3.4.1: Shortley-Weller approximation

We will look at the discretization in the two grid points $G(x_i, y_j)$ and $H(x_i, y_{j+1})$ as introduced in Figure 3.4.1. In the following we will inspect the discretization for H

respectively G. For G we get the following discretization

$$\begin{aligned} -u''(x_i, y_i) \approx &-\frac{2(G_n + G_s)}{G_e} u(x_i + G_e) + \frac{2(G_n + G_s)(G_w + G_e)}{h_w h_e} u(x_i) \\ -\frac{2(G_n + G_s)}{G_w} u(x_i - G_w) &- \frac{2(G_w + G_e)}{G_n} u(y_i + G_n) + \frac{2(G_w + G_e)(G_n + G_s)}{h_n h_s} u(y_i) \\ &- \frac{2(G_w + G_e)}{G_s} u(y_i - G_s) \end{aligned} \tag{3.4.1}$$

and for H something similar. In the case where we have an anisotropic rectangular grid this scaling poses no problems because we know that

$$\begin{aligned} G_w &= H_w \\ G_e &= H_e \\ H_n &= H_s. \end{aligned}$$

ensuring that the two stencils are equal (as easily can be checked by substitution in (3.4.1). The problem in the Shortley-Weller case becomes evident in Figure 3.4.1:

$$G_w \neq H_w. \tag{3.4.2}$$

This has the consequence that the discretization of H_s is not equal to G_n effectively rendering the discretization matrix non-symmetric for boundary grid points. The fundamental cause of the problem is caused by the assumption that every D_x (denoting the mesh spacings in x direction) is the same for every grid-line in x direction. With Shortley-Weller unfortunately this is not the case. Two neighboring grid points near the boundary will always have different mesh spacings.

We inspected the eigenvalues of the discretization matrix with the result that the matrix is non-symmetric positive definite (at least for the cases considered). Further investigations are certainly needed to determine which iterative solvers can be applied.

Validation, convergence and speedups are reported in Section 4.1 on page 23.

3.5 Arbitrary Geometries as Beam Pipes

In this section we discuss how we enable the solver to calculate space-charge forces for arbitrary geometries (i.e. complex cavities geometries etc.). To achieve this we need more information about the simulation state, i.e. the distance d we already traveled in the simulation. This information is required to correctly map the correct part of the geometry to the grid. Furthermore we need to define how beam-pipe geometries are specified, stored and loaded. Once the geometry has been loaded it can be used to compute all grid-points on the domain boundary.

Building the discretization matrix is similar to the variants mentioned in Sections 3.4. Additionally we need also to consider intersections in the longitudinal directions.

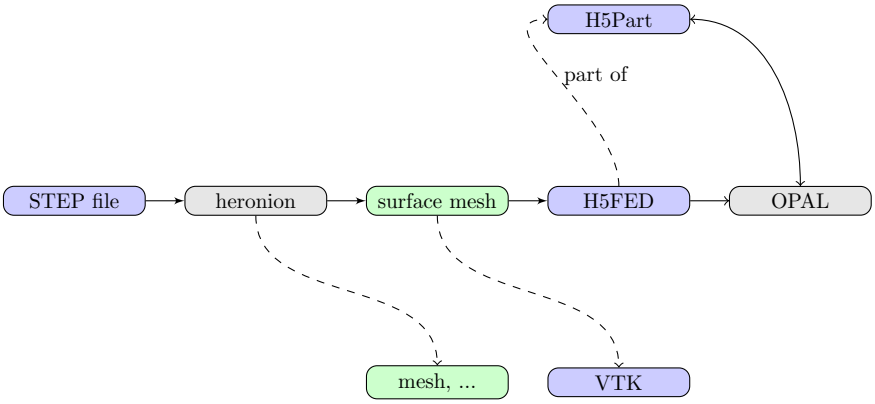

Figure 3.5.1: workflow pipline to use STEP geometries in OPAL. HERONION: physical based mesher developed by Benedikt Oswald, H5FED: HDF5 for finite element data developed by Benedikt Oswald and Achim Gsell.

Specifying Beam Pipe Geometries

It is very important to allow the user to specify the geometries for beam elements in a popular industrial standard. Most CAD systems that are used to design beam elements can export the data to STEP [1] files. The complete work-flow how this is achieved is given in Figure 3.5.1. In the following we will quickly cover all components and interfaces.

H5FED [17] provides an API for storing finite element data on the basis of the HDF5 data storage library. This API supports at the moment reading and writing of hierarchically structured tetrahedral meshes and triangular surface meshes. The API is simple and can easily be integrated into existing codes.

To build the bridge between STEP and H5FED files we could make use of *Heronion*. *Heronion* is a physical based mesher developed by Benedikt Oswald that can generate triangular surfaces and tetraeder meshes from STEP geometries. We extended *Heronion* by a method handling the export of surface meshes to H5FED files.

Efficient Intersection of Grid with Surface Mesh

With the help of the H5FED library OPAL is able to load H5FED files holding the triangular surface mesh. To be able to get the domain boundaries we need to compute the intersection points with our grid. This task should be executed as efficiently as possible since we need to do this in every time-step for our moving window.

Luckily this is a well known problem in computer graphics (i.e. Raytracing, Collision Detection etc.). The ray-triangle intersection is implemented with the algorithm

[1] http://en.wikipedia.org/wiki/ISO_10303

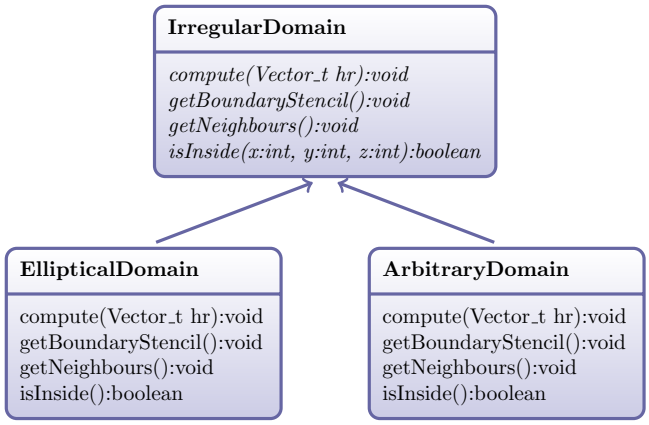

Figure 3.6.1: Inheritance diagram for boundary conditions

described in [18]. This algorithm makes use of the barycentric coordinates, essentially shifting the triangle to the origin and projecting it down. Once the triangle is in this position its very easy to determine if a grid-line intersects the triangle and if so computing the intersection point.

3.6 Implementation

To handle various different irregular domains we introduced new classes that manage internal points and the discretization. The designed class hierarchy is shown in Figure 3.6.1. We introduced a new class called IrregularDomain providing a collection of methods that every irregular domain implementation must provide. Every implementation of a specialized domain is derived from the IrregularDomain class. They all share the same access method to the boundary intersection points used to generate the discretization matrix for the problem. Each derived class has a specialized compute() method that computes all the intersections with the grid and creates a mapping between index and (x, y, z) coordinate needed to build the discretization matrix. Additionally every class provides a method to retrieve the discretization at a global index grid point and its six neighbours global indices. This design enables us to have the solver work in the same way independent of the boundary condition imposed to the system.

The instantiated class starts by computing all intersection points of the geometric object with the grid lines and stores them in lists. STL containers offer a good storage structure for the intersection points, especially the std :: multimap because we are mapping i.e. an integer value (i.e. x identifying the xth grid-line) to multiple intersection double valued points. For symmetric geometries (as i.e. the ellipse) we can reduce the memory for storing the intersection points by only storing a quarter of the ellipse intersections.

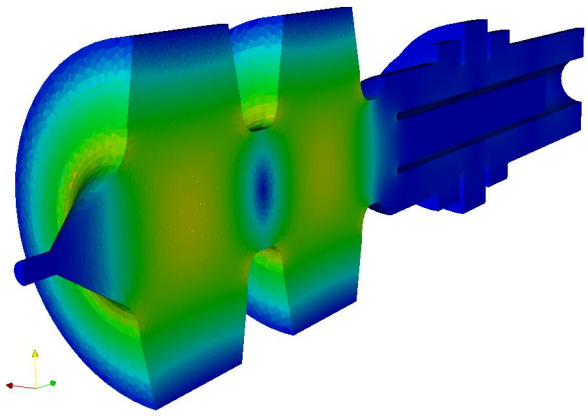

Figure 3.6.2: 2 Frequency LEG Cavity (Based on the design of J.-Y. Raguin (PSI))

In general this map needs

$$\sum_{i=j_{start}}^{j_{end}} numberOfIntersections_i \tag{3.6.1}$$

memory, where j_{start} and j_{end} span the the set of local grid-line in the j-th direction and $numberOfIntersections$ determines the number of intersections of the i-th grid-line in the j-th direction with the domain boundary. For elliptic domains this summation simplifies to

$$2n_j.$$

For arbitrary domains the number of intersections with x and y grid lines will have an average of 2. In the z direction this depends on the geometry to a great extent. This can be seen when considering i.e. the 2 frequency LEG cavity (as shown in Figure 3.6.2 which clearly has a lot more intersections in z direction than 2.

With the help of these maps it then is easy to build up the discretization matrix. Every node simply iterates over all its local indices and queries the currently active domain class about neighbors, stencil values and scale factor for the right hand side.

Complexity

We denote the number of vertices as n, the number of triangles as m, n_x, n_y and n_z for the global number of grid lines in each direction and $local_x$, $local_y$ and $local_z$

for the local number. A summary for the complexity of various operations needed when building the discretization matrix is given in Table 3.1. Clearly we can identify the computation of the intersection with the grid lines as the most computationally heavy task. Another remark concerns the insight that in the current implementation the setup phase (load, intersect geometry and building the index table) for elliptical domains is independent of the processor.

action	Elliptic Domain	Irregular Domain
load geometry	-	$O(m+n)$
index table	$O(n_x n_y)$	$O(local_z n_x n_y)$
intersection with grid	$O(n_x + n_y)$	$O(m(n_x + n_y + local_z))$
build stencil	$O(n_x n_y local_z)$	$O(n_x n_y local_z)$

Table 3.1: complexity for various operations the solver performs

4 Numerical Results

In this chapter we will report results for various discretization approaches discussed in the previous chapter. The following 3 different computational environments were used during this thesis:

Merlin is a small cluster at PSI mostly used for code development and testing with a small number of nodes (≤ 4). The system consists of two Dual Core AMD Opteron(tm) processors with 8 GB RAM.

FELSIM is a new cluster PSI bought. It was installed around mid May. It consists of 8 dual quad-core Intel Xeon processors at 3.0 GHz and has 2 GB memory per node. Currently it is not fully tested and installed.

Buin is a Cray XT4 cluster at the CSCS in Mano used by the swiss weather forecast as main platform for computations. The cluster consists of 468 AMD dual core Opteron at 2.6 GHz, 936 GB DDR RAM, 30 TB Disk and 7.6 GB/s interconnect bandwith.

To convey an idea of the "cost" of the expensive components of the solver a pie chart is presented in Figure 4.0.1. This pie chart shows in percents the contribution of the most computational heavy tasks measured during a run on a single processor on Merlin for rectangular domains.

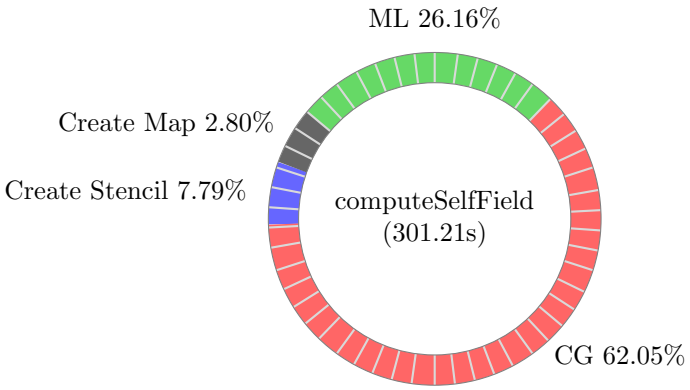

Figure 4.0.1: Solver timings overview for 1 node on a $256 \times 256 \times 256$ grid.

We relinquish further results for rectangular domains since we could not perform measurements on Buin (see Section 5.1) and FELSIM seems to be configured improperly. At the moment one can only observe a "reasonable" speedup behaviour up to 4 nodes.

Most plots shown in this chapter are generated using Gnuplot and Matlab.

4.1 Elliptic Irregular Domains

It was to date not possible to get a working version of OPAL using the Multigrid solver on the Buin cluster at CSCS (see Section 5.1) configuration problems. To overcome this problem we ripped out the solver creating a stand alone solver. This version was used especially for measuring convergence and speedup behaviour but eventually also for the validation.

Validation

To validate the solver for elliptic domains we use the following, to the z axis, rotation symmetric potential function (as mentioned in [3])

$$\rho(r, \phi, z) = (1 - 2r)^3(6r + 1)sin(\pi(z - 0.5)) \tag{4.1.1}$$

where

$$r = \sqrt{(x^2 + y^2)} < \frac{1}{2}.$$

This provides us with a test case where the solution can easily be calculated analytically. The right hand side can be calculated by taking the nabla operator from the potential (4.1.1)

$$b = (96(1 - 4r) + (1 - 2r)^2(6r + 1)\pi^2)(1 - 2r)sin(\pi(z - 0.5)). \tag{4.1.2}$$

In Figure 4.1.1 we see a comparison of the analytical and the numerical solution for the two boundary conditions mentioned. The numerical solution was calculated with OPAL and the analytical with a Matlab script.

It can be deduced that since this discretization only considers grid-points entirely in the domain we lose the circular structure near the boundary. In contrast the difference plot in Figure 4.1.1 for the Shortley-Weller approximation seems to be better in preserving this feature. This also becomes evident when considering the L_2 norm of the error shown in Table 4.1.

Convergence

In Tables 4.2 and 4.3 we see different convergence rates on the various systems for the solver on elliptic domains. ML decided to use 5 levels with aggregates of size $3 \times 3 \times 3$. Since we are using an "decoupled" aggregation scheme we cannot have fewer aggregates than processors. For large number of nodes this may imply that the

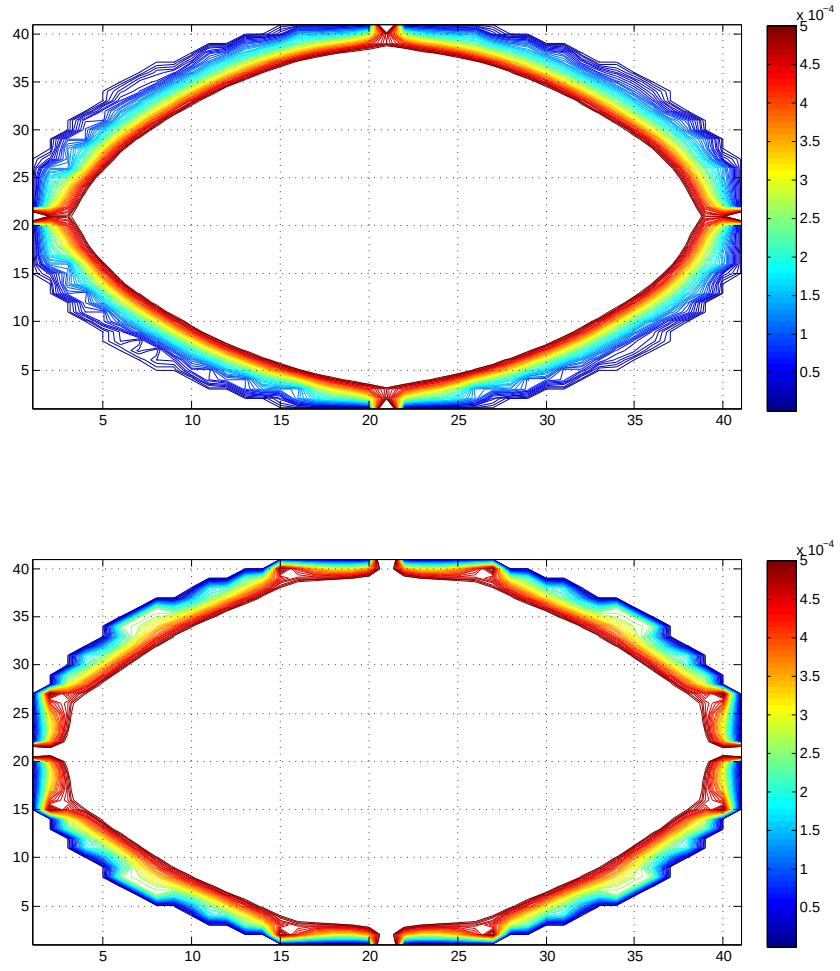

Figure 4.1.1: difference plot near the boundary in xy direction for elliptic irregular domains on a $40 \times 40 \times 40$ grid with mesh spacings of 0.025. UPPER: Shortley-Weller with a L_2 norm of 0.0572 LOWER: $O(h)$ with a L_2 norm of 0.0617

grid size	mesh spacing	L_2 norm of error
$20 \times 20 \times 20$	0.0500	0.1158
$40 \times 40 \times 40$	0.0250	0.0846
$80 \times 80 \times 80$	0.0125	0.0765

Table 4.1: L_2 norm of error

coarsest grid is not really coarse and therefore expensive to solve or slowing down the Multigrid cycle.

The convergence criteria

$$||r^k||_2/||b||_2 < \varepsilon = 10^{-8} \tag{4.1.3}$$

was employed for CG and BiCGStab.

grid size	number of cores	iterations
$256 \times 256 \times 2048$	128	12
$256 \times 256 \times 2048$	256	12
$256 \times 256 \times 2048$	512	11

Table 4.2: convergence for $O(h)$

grid size	number of cores	iterations for CG	iterations for BiCGStab
$256 \times 256 \times 2048$	128	17	10
$256 \times 256 \times 2048$	256	17	10
$256 \times 256 \times 2048$	512	18	11

Table 4.3: convergence for SW

Speedup

Timings and speedup behaviour measured on Buin are shown in Figures 4.1.2 and 4.1.3.

Since BiCGStab uses twice as many matrix-vector product as CG and has a little more than half the number of iterations (compare Table 4.3) BiCGStab actually needs more time to solve the problem. Interestingly CG seems to converge to the same solution (computed with the BiCGStab) for this non-symmetric positive definite problem accrue from the Shortley-Weller discretization.

The speedup is for both discretization approaches nearly optimal when going from 128 to 256 nodes. By doubling the number of nodes again we only gain a small amount

of time ending by an efficiency of around 50%. As discussed in Section 5.2 we hope that this (especially the ML timings) will improve when also parallelize in x and y direction.

4.2 Arbitrary Irregular Domains

In this section we report results for the implementation of arbitrary irregular domains. Again all measurements and tests were conducted on Buin with a stand-alone solver using the arbitrary irregular domain implementation.

Validation

For truly irregular domains the validation is a hard to achieve due to the lack of exact solutions. For now we just generate a cylinder pipe geometry with Heronion and dumped the triangulated surface mesh to a H5FED file. This cylinder then was used as domain boundary followed by conducting the same test (with the same analytical solution) as mentioned in the last section.

L_2 norms are shown in Table 4.4. Difference plots and L_2 norm seems to agree with the results of the last section. This is what we expect since we are also using a cylinder as domain boundary.

grid size	mesh spacing	L_2 norm of error
$20 \times 20 \times 20$	0.0500	0.1145
$40 \times 40 \times 40$	0.0250	0.0838
$80 \times 80 \times 80$	0.0125	0.0755

Table 4.4: L_2 norm of error

Convergence and Speedup

When conducting the same measurements as for elliptic domains the stand-alone solver crashes on Buin when doing a matrix-matrix multiplication while building up the prolongation operator. At the moment it is not quiet clear what is causing the crash but it will be examined in the remaining time.

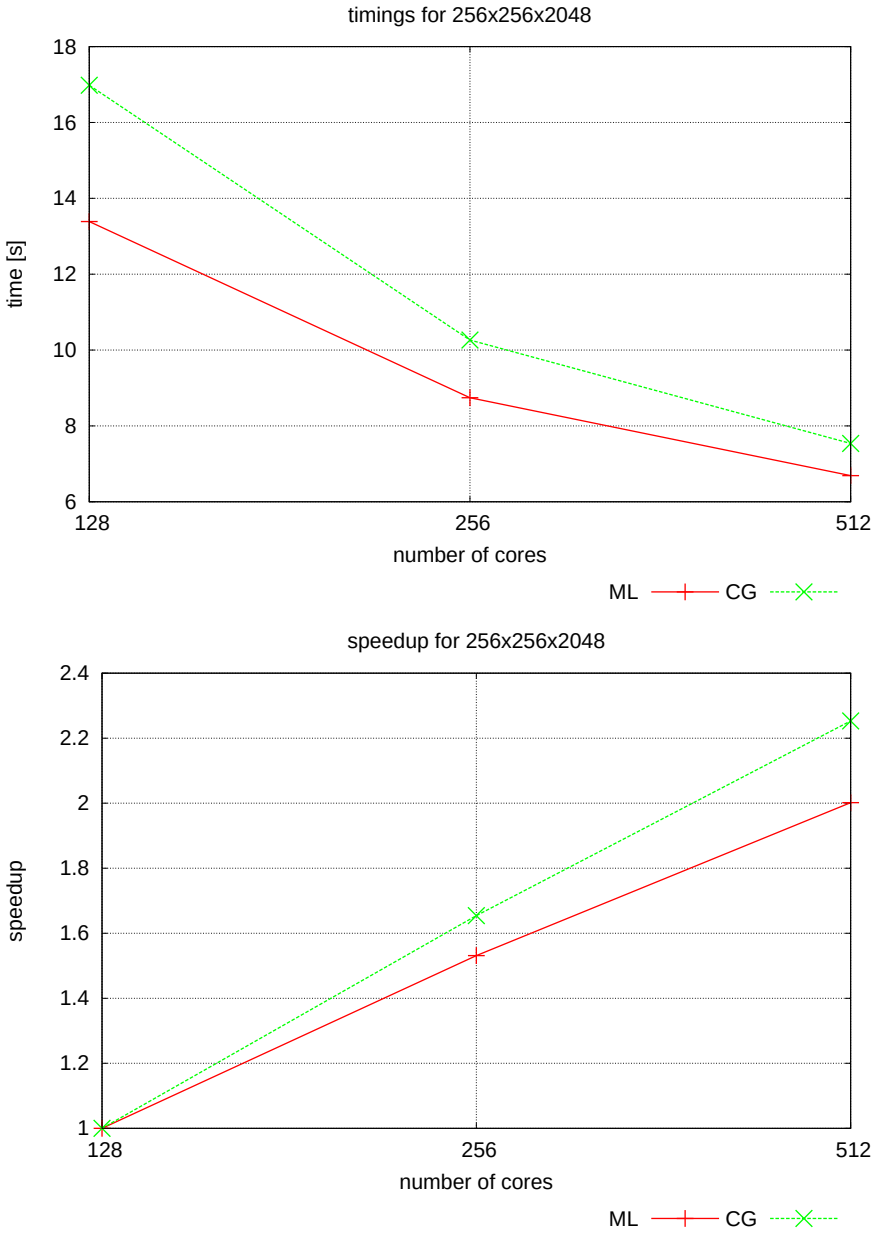

Figure 4.1.2: measurements for the $O(h)$ boundary conditions on Buin. UPPER: timings, LOWER: speedup

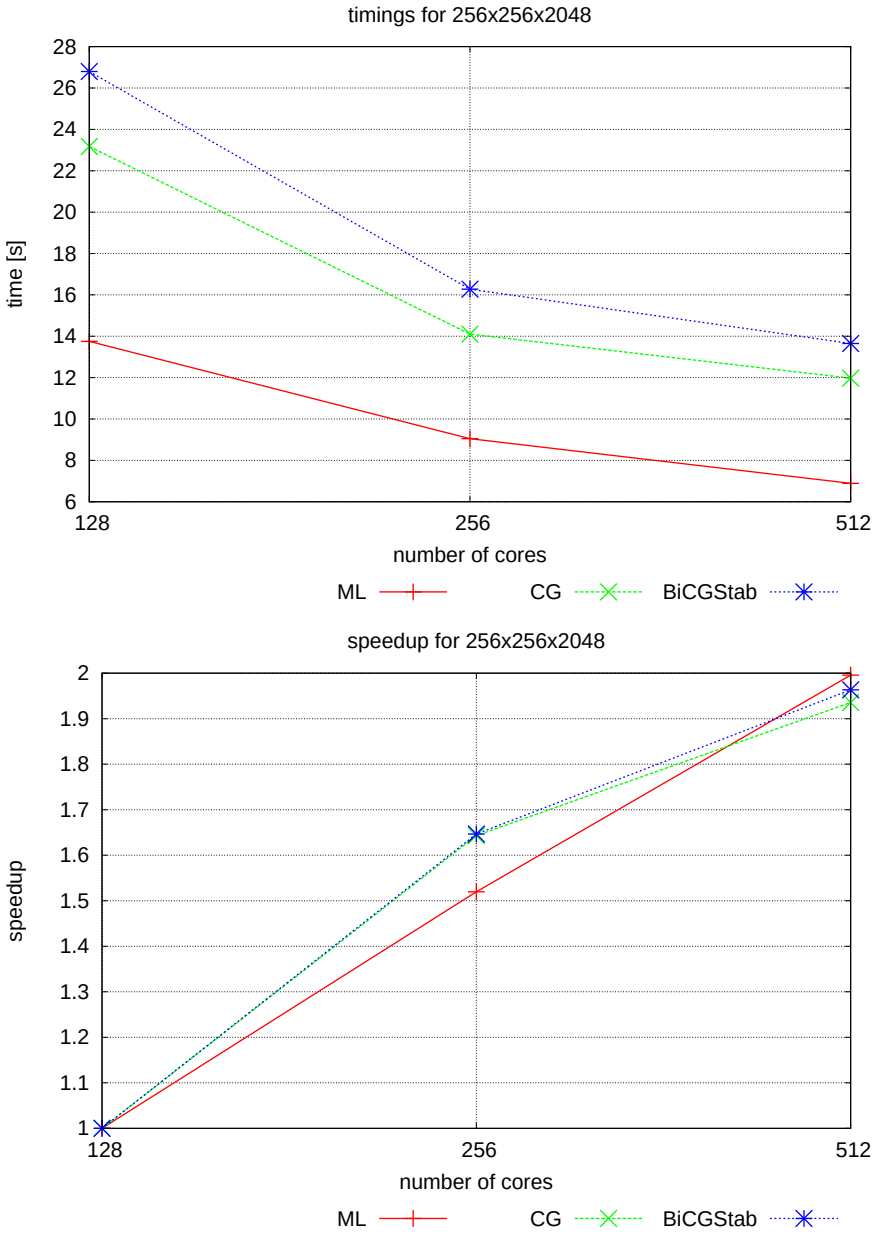

Figure 4.1.3: measurements for the Shortely-Weller boundary condition on Buin. UPPER: timings, LOWER: speedup

5 Conclusion

We presented an algebraic Multigrid with smoothed aggregation as preconditioner for an iterative solver and its implementation with Trilinos. To integrate this iterative solver in OPAL we introduced a method that converts the grid with the charges to an Epetra vector. After completing the integration OPAL can now calculate space-charges with either an iterative solvers or with the already present direct solver. The iterative solver reuses the solution of the last time-step as a good initial guess for the new solution since solutions from consecutive time-steps in general only differ mildly. A first "approximative" test was conducted using rectangular domains to convince us that we get reasonable results.

In a second step we considered an approximation of the beam pipe with elliptic domains. In this step we encountered irregular domains for the first time and we introduced two discretization approaches for grid points near the boundary. For both approaches the numerical solution was validated against an analytical solution, convergence behaviour and parallel performance was measured. The validation revealed that the Shortley-Weller approximation is evidently exacter than the $O(h)$ approach since it does not neglect the distance to the domain boundary for grid-points near the boundary. Interestingly the preconditioned CG solver seemed to converge to the same numerical solution as BiCGStab even tough the system is non-symmetric (Shortley-Weller). The convergence rate is independent of the number of nodes and the grid-size. The BiCGStab solver roughly needs only half of the number of iterations needed by CG but since BiCGStab needs twice as many matrix vector products as CG the time to solution for both methods is approximately equal. The preconditioned solver achieve a parallel efficiency of roughly only 50% when increasing the number of cores from 128 to 512 cores. When doubling the number of cores, going from 128 to 256 cores, parallel efficiency is in the order of 82%.

Finally we have shown that it is possible to take arbitrary geometries of beam pipe elements into account when computing space-charge forces. A simple work-flow has been realized, from STEP files containing the geometry for beamline elements, to H5FED files that can subsequently be used by OPAL . In a next step the geometry is intersected with the grid-lines employing an efficient algorithm. The resulting solver for arbitrary domains seem to match the results achieved with the solver for elliptic domains when run with a triangular surface mesh for a cylinder. Sadly there seem to be problems when we try to measure convergence and parallel efficiency as mentioned in the result section. Since both discretization matrices for elliptical and arbitrary domains are equivalent (aside from the different distance to the boundary) we expect that the convergence rate and parallel efficiency to be similar because we basically solve the same problem with the same solvers.

5.1 Problems

During time of the thesis (and still today) we encounter problems when trying to link OPAL on the Buin cluster in Manno to a binary. Fact is that OPAL can only be built and compiled in the GNU environment (currently no PGI and Pathscale support) and problematically on Buin the GNU environment seems to be not as well supported as the PGI environment. The linking problems seem to be related to Trilinos and dependent libraries (ACML, SuperLU_dist, gfortran a.s.o.). After a time consuming period we could finally link OPAL to a binary (with the help of Timothy Stitt) but the ML package seems to crash while building the preconditioner (this crash is still an open issue). All in all we lost roughly two to three weeks, dealing with all the problems concerning linker and compiler issues on various systems. As mentioned the only way to solve the linker/crash dilemma (on Buin) was to create a stand-alone solver for collecting the results. This stand-alone solver then could be compiled and linked with the PGI compiler. Precisely this problem circumvented us to perform a thorough comparison between the Multigrid and FFT solver.

Another problem we encountered concerns the AMD Core Math Library (ACML[1]). It seemed that when using CG as eigen-analysis type in ML, ACML's routine to calculate all eigenvalues of a symmetric tridiagonal matrix with implicit QL or QR method (DSTEQR) is looping forever. This problem can be avoided by choosing another eigen-analysis type or using the LAPACK[2] DSTEQR routine.

5.2 Further Work

Probably the first thing that should be completed is to change the current parallelization in z direction to a parallelization in all (x, y and z) directions. This is a small task and should not require a lot of changes yet the impact could be measurable. Especially the aggregation process could profit from this new parallelization since it uses cubic blocks.

In the current state the discretization for arbitrary domains is only tested against a read-in cylinder shaped beam pipe geometry. It would be very interesting to see how the space-charge solver handles complicated geometries (i.e. a two-cell cavity) and how big the impact of including exact PEC boundary conditions is on the space-charge forces. This kind of tests will be available when we can fix the outstanding issues with using OPAL on the cluster in Manno (as mentioned in the last section).

An investigation on how stable CG is performing for our non-symmetric positive definite matrices arising when applying the Shortley-Weller approximation would also be appropriate. For simple geometric structures (as i.e. the elliptic beam pipe) it still seems to converge to the correct solution. It would be absorbing to check the convergence and results when having complicated bounded domains, i.e. for the two-cell cavity.

[1] http://www.amd.com/acml
[2] http://www.netlib.org/lapack

On a more global scope one could incorporate adaptive mesh refinements to reduce the computational efforts in regions less interesting or influencing for the solution. This regions could i.e. be the empty space between the bunch and the beam-pipe boundary. This thought would prerequisite further investigation and a thorough analysis of how this "uninteresting" grid-points can be computed and the mesh adaptively refined in the other grid-points.

References

[1] Ji Qiang, Steve Lidia, Robert D. Ryne, and Cecile Limborg-Deprey. Three-dimensional quasistatic model for high brightness beam dynamics simulation. *Acclerators and Beams*, 9(044204), 2006.

[2] David P. Grote, Alex Friedman, Jean-Luc Vay, and Irving Haber. The WARP code: Modeling high intensity ion beams. http://repositories.cdlib.org/lbnl/LBNL-59858, March 2005.

[3] Aleksandar Markovik. Numerical computation of space-charge fields of electron bunches in a beam pipe of elliptical shape. Master's thesis, University of Rostock, 2005.

[4] Peter Mccorquodale, Phillip Colella, David P. Grote, and Jean-Luc Vay. A node-centered local refinement algorithm for Poisson's equation in complex geometries. *Journal of Computational Physics*, 201(1):34–60, 2004.

[5] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[6] H. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13:631–644, 1992.

[7] Youcef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[8] P. Sonneveld. CGS: A fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific Statistical Computing*, 10:36–52, 1989.

[9] Ulrich Trottenberg and Cornelius W. Oosterlee. *Multigrid: Basics, Parallelism and Adaptivity.* Academic Press, 2000.

[10] Wolfgang Hackbusch. *Multi-grid methods and applications.* Springer, 1985.

[11] C.W. Oosterlee and T. Washio. On the use of multigrid as a preconditioner. 1998.

[12] R. Tuminaro and C. Tong. Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines. *SuperComputing*, 2000.

[13] Mark Adams, Marian Brezina, Jonathan Hu, and Ray Tuminaro. Parallel multigrid smoothing: polynomial versus Gauss–Seidel. *Journal of Computational Physics*, 188(2):593–610, July 2003.

[14] Michael Heroux, Roscoe Bartlett, Vicki Howle Robert Hoekstra, Jonathan Hu, Tamara Kolda, Richard Lehoucq, Kevin Long, Roger Pawlowski, Eric Phipps, Andrew Salinger, Heidi Thornquist, Ray Tuminaro, James Willenbring, and Alan Williams. An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

[15] M.W. Gee, C.M. Siefert, J.J. Hu, R.S. Tuminaro, and M.G. Sala. ML 5.0 smoothed aggregation users's guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[16] G.H. Shortley and R. Weller. The numerical solution of Laplace's equation. *Journal of Applied Physic*, 9:334–344, 1938.

[17] B. Oswald and A. Gsell. H5fed. http://h5part.psi.ch, 2008.

[18] Tomas Moeller and Ben Trumbore. Fast, minimum storage ray/triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.

A Proof Sketch: Symmetrize Anisotropic Discretization on a Rectangular Domain

A.1 3D Poisson Discretization with Kronecker Products

On a regular grid we can construct the 3D discretization of a Poisson problem with help of the Kronecker product and 3 1D discretizations.

$$\mathbf{D} = \begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}$$

$$\mathbf{A}_2 = \mathbf{D}_x \otimes \mathbf{I}_{n_y} + \mathbf{I}_{n_x} \otimes \mathbf{D}_y$$

$$\mathbf{A}_3 = \mathbf{A}_2 \otimes \mathbf{I}_{n_z} + \mathbf{I}_{n_x \times n_y} \otimes \mathbf{D}_z$$

where $\mathbf{I}_n$ is the unit matrix of size $n \times n$. $\mathbf{A}_3$ is the common 7 point discretization for a 3D Poisson problem with the size $n_x \times n_y \times n_z$ (see Fig. A.1.1).

Symmetrization of the Discretization Matrix

To symmetrize the discretization matrix on an irregular grid we introduce a new diagonal matrix $\mathbf{D}$. The diagonal entries of this matrix consist of $h_{i,w} + h_{i,e}$ for every

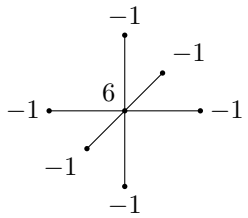

Figure A.1.1: default 7 point discretization for 3D Poisson problems

node i ($h_{i,e}$ stands for the east and $h_{i,w}$ for the west neighbor of node i).
It can easily be seen that when multiplying this matrix $\mathbf{D}_x$ with the 1D discretization matrix $\mathbf{A}_x$ we get a matrix of the form

$$\mathbf{DA}_x = \mathbf{D}_x\mathbf{A}_x = \begin{bmatrix} \ddots & \ddots & & & \\ \ddots & \frac{h_{i-1,w}+h_{i-1,e}}{h_{i-1,w}h_{i-1,e}} & \frac{-1}{h_{i-1,e}} & 0 & \\ & \frac{-1}{h_{i,w}} & \frac{h_{i,w}+h_{i,e}}{h_{i,w}h_{i,e}} & \frac{-1}{h_{i,e}} & \\ & 0 & \frac{-1}{h_{i+1,w}} & \frac{h_{i+1,w}+h_{i+1,e}}{h_{i+1,w}h_{i+1,e}} & \ddots \\ & & \ddots & \ddots & \end{bmatrix}. \tag{A.1.1}$$

THEOREM A.1.1 *For the 1D case the distance from node i to the next node eastwards and the distance of node $i+1$ to the next node westwards is equal.*

$$\forall i: \quad h_{i,e} = h_{i+1,w}$$

Proof sketch (Theorem A.1.1). Node i and $i+1$ are neighbors and therefore connected by only one vertex. □

When applying Theorem A.1.1 to matrix (A.1.1) we see that $\mathbf{DA}_x$ is indeed symmetric positive definite. Clearly this argument holds for the 1D discretization matrix in every direction (x, y and z).
In a next step this 1D discretization matrices are combined with the Kronecker product to get the discretization matrix for the 3D problem. We start by combing the x and y direction. To do this we introduce two lemmas.

LEMMA A.1.1 *Given matrices $\mathbf{A} \in \mathcal{R}^{n,n}$, $\mathbf{B} \in \mathcal{R}^{m,m}$, $\mathbf{X} \in \mathcal{R}^{n,m}$ and $\mathbf{F} \in \mathcal{R}^{n,m}$ the following equivalence holds:*

$$\mathbf{AXB}^T = \mathbf{F} \iff (\mathbf{A} \otimes \mathbf{B})\mathrm{vec}(\mathbf{X}) = \mathrm{vec}(\mathbf{F}) \tag{A.1.2}$$

LEMMA A.1.2 *Given matrices $\mathbf{A} \in \mathcal{R}^{n,n}$, $\mathbf{B} \in \mathcal{R}^{m,m}$, $\mathbf{X} \in \mathcal{R}^{n,m}$ and $\mathbf{F} \in \mathcal{R}^{n,m}$ the following equivalence holds:*

$$\mathbf{AX} + \mathbf{XB}^T = \mathbf{F} \iff (\mathbf{A} \otimes \mathbf{I}_m + \mathbf{I}_n \otimes \mathbf{B})\mathrm{vec}(\mathbf{X}) = \mathrm{vec}(\mathbf{F}) \tag{A.1.3}$$

Proof (Lemma A.1.1). We start with the RHS of Lemma A.1.1

$$\begin{aligned} (\mathbf{A} \otimes \mathbf{B})\mathrm{vec}(\mathbf{X}) &= \mathrm{vec}(\mathbf{F}) \\ \begin{pmatrix} \mathbf{A}b_{1,1} \dots \mathbf{A}b_{1,m} \\ \vdots \quad\quad \vdots \\ \mathbf{A}b_{m,1} \dots \mathbf{A}b_{m,m} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_m \end{pmatrix} &= \begin{pmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_m \end{pmatrix} \\ \mathbf{A}\sum_j b_{ij}\mathbf{x}_j &= \mathbf{f}_i \quad i = 1 \dots m \\ [\mathbf{AXb}_1, \dots, \mathbf{AXb}_m] &= \mathbf{F} \\ \mathbf{AXB}^T &= \mathbf{F} \end{aligned}$$

□

Proof (Lemma A.1.2). We start with the RHS of Lemma A.1.2

$$(\mathbf{A} \otimes \mathbf{I}_m + \mathbf{I}_n \otimes \mathbf{B})\text{vec}(\mathbf{X}) = \text{vec}(\mathbf{F})$$

Using Lemma A.1.1 we can rewrite the equation above as

$$\underbrace{\mathbf{A}\mathbf{X}\mathbf{I}_m^T}_{=\mathbf{A}\mathbf{X}} + \underbrace{\mathbf{I}_n\mathbf{X}\mathbf{B}^T}_{=\mathbf{X}\mathbf{B}^T} = \mathbf{F}$$

□

With the help of Lemma A.1.2 we can combine the x and y discretization:

$$\begin{aligned}
\mathbf{T}_x\mathbf{X}_{xy} + \mathbf{X}_{xy}\mathbf{T}_y^T &= \mathbf{F}_{\mathbf{xy}} \\
(\mathbf{T}_x \otimes \mathbf{I}_y + \mathbf{I}_x \otimes \mathbf{T}_y)\text{vec}(\mathbf{X}_{xy}) &= \text{vec}(\mathbf{F}_{xy}) \\
(\mathbf{D}_x \otimes \mathbf{D}_y)(\mathbf{T}_x \otimes \mathbf{I}_y + \mathbf{I}_x \otimes \mathbf{T}_y)\text{vec}(\mathbf{X}_{xy}) &= (\mathbf{D}_x \otimes \mathbf{D}_y)\text{vec}(\mathbf{F}_{xy}) \\
(\underbrace{\mathbf{D}_x\mathbf{T}_x}_{=\mathbf{D}\mathbf{T}_x} \otimes \underbrace{\mathbf{D}_y\mathbf{I}_y}_{=\mathbf{D}_y} + \underbrace{\mathbf{D}_x\mathbf{I}_x}_{=\mathbf{D}_x} \otimes \underbrace{\mathbf{D}_y\mathbf{T}_y}_{=\mathbf{D}\mathbf{T}_y})\text{vec}(\mathbf{X}_{xy}) &= (\mathbf{D}_x \otimes \mathbf{D}_y)\text{vec}(\mathbf{F}_{xy})
\end{aligned}$$

Remark A.1.1 The Kronecker product of two symmetric matrices is again symmetric.

$$(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T = \mathbf{A} \otimes \mathbf{B} \tag{A.1.4}$$

Remark A.1.2 The addition of two symmetric matrices yields a symmetric matrix.

Remark A.1.3 The equation on the RHS

$$(\mathbf{D}_x \otimes \mathbf{D}_y)\text{vec}(\mathbf{F})$$

can be computed in place (in vec($\mathbf{F}$)) since $D_x \otimes D_y$ results in another diagonal matrix which then is component-wise multiplied with vec($\mathbf{F}$)

Remark A.1.4 When computing the discretization matrix $(\mathbf{D}\mathbf{T}_x \otimes \mathbf{D}_y + \mathbf{D}_x \otimes \mathbf{D}\mathbf{T}_y)$ we only need to implement a "light" version of the Kronecker product for symmetric CSR matrices.

Now we can add the z direction

$$\begin{aligned}
\mathbf{T}_{xy}\mathbf{X}_{xyz} + \mathbf{X}_{xyz}\mathbf{T}_z^T &= \mathbf{F}_{xyz} \\
(\mathbf{I}_{xy} \otimes \mathbf{D}_z)(\mathbf{T}_{xy} \otimes \mathbf{I}_z + \mathbf{I}_{xy} \otimes \mathbf{T}_z)\text{vec}(\mathbf{X}_{xyz}) &= (\mathbf{I}_{xy} \otimes \mathbf{D}_z)(\mathbf{D}_x \otimes \mathbf{D}_y)\text{vec}(\mathbf{F}_{xyz}) \\
(\underbrace{\mathbf{I}_{xy}\mathbf{T}_{xy}}_{=\mathbf{T}_{xy}} \otimes \underbrace{\mathbf{D}_z\mathbf{I}_z}_{=\mathbf{D}_z} + \underbrace{\mathbf{I}_{xy}\mathbf{I}_{xy}}_{=\mathbf{I}_{xy}} \otimes \underbrace{\mathbf{D}_z\mathbf{T}_z}_{=\mathbf{D}\mathbf{T}_z})\text{vec}(\mathbf{X}_{xyz}) &= (\mathbf{I}_{xy} \otimes \mathbf{D}_z)(\mathbf{D}_x \otimes \mathbf{D}_y)\text{vec}(\mathbf{F}_{xyz})
\end{aligned}$$

where $\mathbf{T}_{xy}$ is the resulting matrix for the combination of the discretization in x and y direction.

REMARK A.1.5 The same argument (as mentioned in Remark A.1.3) also holds for the 3D case.

$$(\mathbf{I}_{xy} \otimes \mathbf{D}_z)(\mathbf{D}_x \otimes \mathbf{D}_y)\mathrm{vec}(\mathbf{F})$$