Designing Efficient Solvers for Large Scale Elliptic PDEs

by

Haixiang Liu

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2019

Date of final oral examination: Jan/10th/2019

The dissertation is approved by the following members of the Final Oral Committee:

Eftychios Sifakis, Associate Professor, Computer Sciences
Dan Negrut, Professor, Mechanical Engineering
Michael Gleicher, Professor, Computer Sciences
Matthew Sinclair, Assistant Professor, Computer Sciences

For all of my wonderful family, friends, and mentors, who have borne with me in the past six years.

ACKNOWLEDGMENTS

I want to thank my advisor Prof. Sifakis for the guidance and instructions, and for giving me the freedom to experiment in my own ways. Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Negrut, Prof. Gleicher, and Prof. Sinclair, for their time and engagement. Also, I would like to thank Mridul Aanjaneya, Nathan Mitchell, Bo Zhu, and Yuanming Hu for their contributions to the projects presented in this thesis.

Last but not least, I want to thank my family for supporting me studying oversea. Hope that I can see you soon.

This work was generously supported by the following grants and funding sources: NSF grants IIS-1253598, CCF-1533885, CCF-1423064, IIS-1407282, CMMI-1644558, CCF-1533753, CCF-1533885, IIS-1763638, CCF-1812944.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Numerical simulation of physical phenomena has long been of interest in the area of mechanical engineering, physics, and in recent years, computer graphics. Recent advances of computation hardware have opened up new opportunities for improving numerical simulations in terms of both scale and performance. However, the heterogeneity of modern hardware has imposed unique challenges that limit the utilization of computation hardware using the traditional programming paradigm. This dissertation investigates the design of efficient Poisson and linear elasticity solvers for modern hardware to demonstrate design practices and principles for utilizing modern hardware in the context of numerical solvers.

1 INTRODUCTION

1.1 Motivation

Physics based simulation originated in the field of engineering. For example, aerodynamic simulation has been wildly used in aircraft design. In addition, the simulation of elasticity has been employed for designs of large scale structures, such as bridge or skyscrapers. In recent years, physics based simulation, which we will simply refer to as *simulation* in this document, has also been applied in the field of computer graphics. The simulation of smoke and water has been ubiquitous in the field of special effects. Soft body simulations have been integrated into the standard animation pipelines.

Simulation with billions of active degrees of freedom marks a special milestone. It is not merely an artificial milestone in terms of magnitude, but it also has implications for the fields of visual effects and engineering. For the field of visual effects, billions of active voxels, with the dominating axis' resolution in the order of thousands, allow the capture of pixel level details in feature films. On the engineering front, the recent emergence of commercial 3D printers has rekindled interest in topology optimization, as accessible printers allow the manufacturing of the optimized structures. Billion active voxels topology optimization creates details that are at the same level at the commercial printer resolution.

Until recently, simulations of active voxels in the order of billions had only been achieved using clusters. In the feature film *The Good Dinosaur*, Reisch et al. (2016) simulated the river in sections, each assigned to a cluster, with the largest sections of the resolution $80000 \times 10000 \times 3000$. In earlier work by Henderson (2012), a simulation of a tornado at the resolution $300 \times 300 \times 1200$ was achieved using a shared memory system. For engineering applications, giga-voxel topology optimization has been

recently achieved by Aage et al. (2017), utilizing 8000 cores.

Resolution is an important factor for simulations. What is the strategy for maximizing the simulation resolution using a single-chassis computation platform? This document aims to tackle this question. Maximizing the simulation resolution extends beyond minimizing memory footprint. With the resolution scaling, domains with more complex boundary conditions and higher variance of material distribution emerge. The problems of fluid simulation with complex boundaries, and of elasticity simulation with highly varying material properties, are examined in this document. As the complexity of the domain increases with the resolution, the existing solvers face the challenge of solving the equation with good accuracy within a reasonable time frame. One of the major issues this dissertation aims to address is the poor solver convergence rate for high complexity problems. Before we venture into more details of how this is achieved, we will first cover the features of modern hardware.

In recent years, available computational power in workstations has greatly advanced. This has been achieved not by increasing core speed, but raising core count, widening SIMD (same instruction multiple data) width (SSE to AVX2, to AVX512), deepening the level of memory hierarchy (MCDRAM in *Knights Landing*), and bus interconnected accelerators (Xeon Phis and GPUs). These changes of hardware have opened up new opportunities in re-designing algorithms for numerical simulations that adapt to these paradigms.

This document presents a set of techniques and algorithms for solving large scale elliptic linear equations that emerge from the simulations of physical phenomena targeted for modern hardware. They are demonstrated by adapting and redesigning solvers for two specific problems: homogeneous Poisson and heterogeneous linear elasticity, both discretized on a Cartesian grid. The solvers presented in this thesis has enabled simulations using billions of voxels on a single workstation. Such scale not only

provides a stunning level of visual detail and higher accuracy in solution, but also exposes the limitations of the existing iterative solvers.

Targeting these discoveries from the first two projects presented in the thesis, the last part of this thesis proposes a novel geometric multigrid method with augmented rotational degrees of freedom for linear elasticity that address the slow convergence issue of standard multigrid preconditioned conjugate gradient solvers on domains with complex geometry.

1.2 Thesis

The progression of computation power of modern processors has shifted towards the direction of more parallelism and more heterogeneity. In pursuit of higher resolution simulations, new paradigms for designing solvers are needed to take advantage of this change. This document uses concrete designs of two solvers to demonstrate how to adapt to these new paradigms. The solvers are benchmarked and compared with the peak theoretical throughput of the platform to illustrate their efficiencies.

With the achievement of higher resolutions, the limitations of existing numerical methods are also explored. This document also proposes new algorithms that address these limitations and extend the boundaries of existing capabilities. As a mark of success, throughout this thesis, we evaluated the convergence and performance properties of the proposed solvers using domains with billions of degrees of freedom.

1.3 Scope of the Study

For large scale numerical simulations, the most time consuming step is commonly solving linear systems. This thesis work focus on techniques on solving large linear systems that emerge from numerical simulations. We evaluate and compare our proposed techniques based on their perfor-

mance, complexity, and convergence rate with well established existing method multigrid to demonstrate our advantage.

1.4 Main Contributions

In this part, an overview of the three major contributions in this thesis will be presented.

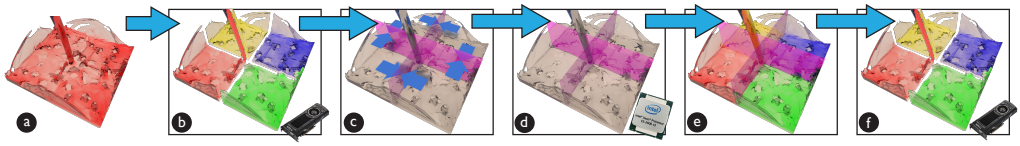


Figure 1.1: Illustration of a divide-and-conquer algorithm for solving imcompressible flow problem on a heterogenous server

An Algorithm for Utilizing the Computational Power of Multiple GPUs on A Large Memory System Large scale simulations that require memory exceeding the aggregate capacity of the GPUs are challenging for heterogeneous platform based numerical solvers. They typically need frequent global synchronization and access to non-local memory if the data does not fit on the GPUs. A novel divide-and-conquer algorithm for solving homogeneous Poisson problems is presented. It targets the minimization of frequency and amount of data that is communicated across GPUs. This algorithm not only has a lower solve time, but also significantly improves on the convergence rate on large irregular domains.

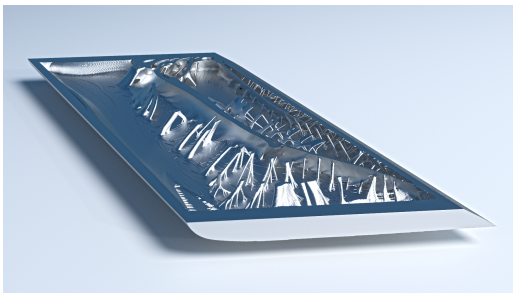


Figure 1.2: A cross section of the interior of a wing structure generated by topology optimization

An Efficient Vectorized Linear Elasticity Solver With a Mixed-Precision Scheme for Topology Optimization Driven by the scale demanded by topology optimization workload, we designed an efficient linear elasticity solver. With resolution scaling as the main objective, the solver is designed to be minimal in memory footprint and also fully utilized AVX-512 instructions that are featured in modern CPUs for performance. The key piece of the puzzle is a collection of techniques that allows the usage of aligned vector operation on the sparse data structure, SPGrid. For high resolution simulation, single-precision is insufficient in accuracy while double-precision takes twice the memory and computational time. We designed a mix-precision scheme that only requires a friction of the storage of double-precision while maintaining its accuracy.

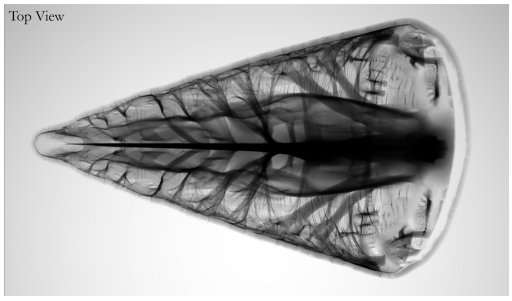


Figure 1.3: The complex density field of a bird beak created by topology optimization

A Stencil-Aware Geometric Multigrid Algorithm For Linear Elasticity One of the advantage of high resolution simulations is the amount of detail that they are capable of capturing. But those complex geometrical details can severely impair the convergence rate of the existing multi-linearly interpolated multigrid solver. To address this convergence problem for high res-

olution linear elasticity simulation, a stencil-aware geometric multigrid algorithm is proposed. This new method can reduce residual by approximately half every iteration even with the most irregular domains.

1.5 Related Technologies

Several existing technologies are the main building blocks for the projects presented in this thesis, namely: Sparse Paged Grid(SPGrid), Matrix Free Design of Linear Operators, Multigrid(MG), and Domain Decomposition Method. In this sections, I will give a brief introduction about those concepts.

Sparse Paged Grid

The simulation domains we targeted in this thesis are sparse. In one of the examples, the simulation domain is of the size $8192^2 \times 4096$ with 1.2 billion active cells, a mere 4% occupancy. Sparse Paged Grid, or SPGrid, has the capability of representing sparse domain while maintaining computational efficiency. This is the key reason for which it was chosen as the main data structure through out this thesis.

Sparse Paged Grid, or SPGrid, was introduced by Setaluri et al. (2014). It is a sparse data structure that stores data on a Cartesian grid using the unit of blocks. A common block size is $4 \times 4 \times 8$. Each block is associated with one or multiple physical memory page(s). If a block is active, i.e. one of the cell of the block is an active voxel in the simulation domain, the corresponding memory page(s) will be allocated in the physical memory. Otherwise, it will only occupy virtual memory without any physical footprint. SPGrid utilizes the virtual memory for its sparsity management, which eliminates additional memory address look up which are needed for traditional tree like sparse data structure. This makes its access cost close to a cache optimized dense uniform grid. The biggest drawback

of using a Cartesian grid discretization such as SPGrid is its inability in accurately capturing of the boundary of a domain. Though this document itself does not deal with this inaccuracy. There are many previous works that target this limitation in both the field of elasticity [Zhu et al. (2012)], and fluid simulation [Sethian and Smereka (2003)].

Matrix Free Design of Linear Operators

The targeted resolution is in the order of billions of active voxels. High end PCs nowadays usually have less than 512GB of memory. We can not afford to store the system matrix of the linear system created by the discretization. As an illustration of the memory footprint of the system matrix, for a Poisson problem, the number of non-zero entries in the matrix is 7 times the total degrees of freedom(4 if we only store the symmetric part). For linear elasticity problem that number is drastically increased to 81 times the total degrees of freedom (42 if we only store the symmetric part). Therefore, we use the technique of matrix free operations, to emulate the matrix multiplication without storing it by assembling the matrix on the fly.

Besides the memory saving benefit, matrix free operations can also improve performance. When a matrix is given and stored, matrix-vector multiply usually requires only one multiplication and one addition per memory access. This creates functions that are heavily bound by memory access. When using matrix free operations, the functions are no longer bounded by reading matrix entries from memory. In turn, we can greatly improve performance(see Chapter 3 and Chapter 4).

Of course, the performance gain from using matrix free operation is not free. To implement matrix free operations, it requires special design of our computation kernels for each imposing problem.

Multigrid

The goal of upscaling simulation can be hindered if the solve stage became more expensive as the resolution increases. Therefore, we want to select solvers that have approximately $O(N)$ complexity as we go to higher resolutions.

Multigrid was introduced by Brandt (1977) to solve linear systems discretized from elliptic partial differential equations. It may seem to be a narrow target. The good news is many equations emerged from physical phenomena are from this category. Unlike conjugate gradient, multigrid is not a fixed algorithm, but a principle for designing efficient solver for elliptic PDEs. More thorough overview of multigrid will be given in Chapter 2. Here I will just highlight some key features of multigrid:

- An ideal multigrid is proven to be able to reduce error a constant rate. With each iteration of cost of $O(N)$, multigrid has the **potential** to be a $O(N)$ complexity solver. However, in reality, multigrid can not always achieve this, or worse, in most cases it can not.

- Each component of a multigrid grid solver is relatively light-weight. They often do very small amount of computation for each memory access. It makes them very efficient for hardware has a lot of memory bandwidth such as GPUs, but it also makes them extremely inefficient to offload each stage to multiple GPUs if the GPUs would require constant access of remote memory and global synchronizations without much computation.

- Multgrid can be used as a preconditioner for conjugate gradient. It can further improve its convergence rate. This is referred to as **M**ulti**G**rid **P**reconditioned **C**onjugate **G**radient or **MGPCG** for short.

- Multigrid is a design principle. Each implementation of multigrid differs with the different continuous equation, discretization and

data structure chosen, and various components of the multigrid we select (smoothers, prolongations, coarsening operators, see Chapter 2). For optimal performance, in each project we have to re-design and re-implement the multigrid solvers based on all the criteria above.

Domain Decomposition Method

Based on multigrid theory we can design solvers with the potential of linear complexity. But as multigrid solvers require frequent synchronization and global memory access, it is difficult for them to utilize heterogeneous platform, for instance a multi-GPU equipped computer. In contrast, the **domain decomposition** method also has the potential of linear complexity. By dividing the simulation domains into smaller parts, either overlapping or non-overlapping, each parts can be solved in isolation. Therefore if we partition the domain into small pieces with each piece fitting on a single GPU, we can eliminate the need for frequent synchronizations.

But to retain domain decomposition's linear complexity and convergence rate, there are many complex pieces need to be considered. Those details will be presented in Chapter 3.

1.6 Outline

The remaining chapters of the documents are divided as the following. Chapter 2 provides an overview of state of the art numerical solvers for simulation, specifically the class of solvers named multigrid methods. Chapter 3 presents a novel domain decomposition method that utilizes multi-accelerator equipped platform for solving Poisson equations. Chapter 4 presents a SIMD optimized solver for linear elasticity that is designed for minimizing memory footprint and maximizing computational throughput of modern hardware. Chapter 5 presents an stencil aware multigrid

method that overcomes the slow convergence issue when solving linear elasticity problems with complex domain using traditional solvers. Chapter 6 concludes this thesis by revisiting the contributions brought by this dissertation and looks into the future work suggested by the limitations of the methods presented.

2 NUMERICAL SOLVERS IN PHYSICS BASED SIMULATION

In the pipeline of physics based simulation, the complexity of the majority of the stages is $O(N)$, where N is the number of unknowns, while the complexity of the solve stage can be as high as $O(N^2)$. For high resolution simulation, the solve stage can often take over 90% of the simulation time. There are two major categories of numerical solvers: direct solvers and iterative solvers.

Direct solvers provide solutions that approach the accuracy of machine precision, and they generally operate in sparse matrix format (for instance, Compressed Sparse Column). Therefore, they can be used for arbitrary discretization and problems. However, they have the minimal computational cost of $O(N^2)$ and memory footprint of $O(N^{\frac{4}{3}})$ or higher. This super-linear cost makes direct solvers unfeasible for high resolution simulations. But for small scale simulations, direct solver algorithms such as Cholesky factorization are attractive for their accuracy and robustness.

Iterative solvers, on the other hand, as the name suggests, provide a solution that converges to the exact solution with each iteration. Some iterative solvers, such as Conjugate Gradient Method [Nocedal and Wright (2006)] and Generalized Minimal Residual algorithm [Saad and Schultz (1986)], mathematically speaking, can achieve the exact solution at the Nth iteration. At the cost of $O(N)$ per iteration, it will give the exact solution with $O(N^2)$ cost. But in practice, due to numerical drift, these algorithms will require restarts for large problems, and only achieve the solution asymptotically.

For large problems, an iterative solver is preferred over direct solver for two major reasons: 1. the memory footprint of an iterative solver is usually $O(N)$, which allows larger problems to fit into memory. 2. Though iterative solve can potentially takes longer to achieve solution at numerical limit, the option to terminate early for an approximate solution is attractable

when computation time is limited. But at higher and higher resolution, iterative solvers can take considerably longer to converge and sometimes even stagnate (See results in Chapter 4). If terminated too early, the poorly approximated solution can lead to nonphysical results, due to that the solution does not satisfy the governing equation.

2.1 Multigrid Method Overview

Multigrid method was proposed by Brandt (1977). The multigrid fundamental idea lies as follows: we start with a set of grids G^0,G^1,...,G^M, which are all discretizations of the same domain Ω. Each higher level consists of coarser elements. In our uniform grid discretization, in 1D, if the nodes in G^0 lie on points $(0, h, 2h, 3h, 4h...)$. We can construct the next level G^1 as nodes on points $(0, 2h, 4h, 6h, 8h...)$.

Let our continuous PDE of the boundary value problem take the form:

$$LU(x) = F(x), \text{ in } \Omega, \Lambda U(x) = \Phi(x), \text{ on the boundary } \partial\Omega \tag{2.1}$$

Here, L and Λ are the differential operators that are on the interior of the domain and the boundary respectively. $U(x)$ is the solution we seek. $F(x)$ and $\Phi(x)$ are the loading condition that are given. We can now write the discretized PDE at each level k as:

$$L^k U^k(x) = F^k(x), \text{ for} x \in G^k, \Lambda^k U^k(x) = \Phi^k(x), \text{ for} x \in \partial G^k \tag{2.2}$$

Now, L^k and Λ^k are linear operators and can be perceived as matrices. We are interested in acquiring the solution at the finest level. The main idea is, if at kth level, the solution $U^k(x)$ is a good approximation to the continuous $U(x)$. We can use the kth level solution as a potential guess for the $k-1$ level. Then apply a correction routine called **smoother** that is $O(|G^{k-1}|)$ cost to bring the guess close to solution. This process, sometimes

referred to as V-cycle or W-cycle, is repeated until $U^0(x)$ is considered close enough to the solution.

Multigrid has the property that each iteration can reduce the error by at least a constant factor W. That is, $\frac{|\mathbf{e}_{i+1}|}{|\mathbf{e}_i|} < W$. Here we take $|\mathbf{e}|$ as the L_∞ norm of $\mathbf{e}$. What the value of W is depends on the PDE, how the discretization of each level is constructed, and how potent the smoother is. The convergence property of multigrid will be elaborated on the later section. But we can see that if we seek to reduce the error by 6 orders of magnitude, we will need $-\log_W 10^6$ iterations to converge, that is a constant. For instance, if $W = 0.5$, multigrid will take at most 20 iterations to converge. But, of course, in worse cases, for example, when $W = 0.99$, it will take about 1400 iterations.

Now that I have introduced some fundamental multigrid concepts, let us take a deeper look at multigrid constructions.

2.2 Construction of the Multigrid Hierarchy

To construct the discretized hierarchical operators in Equations 2.2, there are two common schemes, geometric coarsening and Galerkin coarsening. In work of Zhu et al. (2010b) and McAdams et al. (2010), geometric coarsening was used to construct hierarchical operators. For the domain decomposition solver, geometric coarsening based multigrid is used as its building block, see Chapter 3.

But in many cases, only the top level operator is provided to the solver, and the underlying PDE is inaccessible, re-discretization of the PDE with a coarser grid(or mesh) is not an option. In those cases, Galerkin coarsening is usually used for constructing the hierarchy, such as in Dendy (1982), Brezina et al. (2001), Dohrmann (2007). This is also our method for the multigrid construction in our SIMD-optimized solver presented in Chapter 4.

When the underlying PDE is unavailable, there is the possibility of acquiring the geometric coarsened operator through homogenization of Galerkin coarsened operator which is demonstrated in previous work by Moulton et al. (1998). But this aspect is not the focus of this work. Therefore, I will leave out the discussion of homogenization.

2.3 Geometric Coarsening

Geometric coarsening refers to the method of constructing the Multigrid hierarchy through re-discretization with different grid size. It has been proven effective for both homogeneous Poisson [McAdams et al. (2010)] and homogeneous elasticity [Zhu et al. (2010b)]. Here, we will take finite difference Poisson discretization as an example to demonstrates the geometric coarsening principles and its limitation. For more details, we would refer the reader to McAdams et al. (2010).

The continuous PDE of homogeneous Poisson can be written as:

$$\Delta p(\mathbf{x}) = f(\mathbf{x}) \text{ in } \Omega \in \mathcal{R}^3 \tag{2.3}$$

$$p(\mathbf{x}) = \alpha(\mathbf{x}) \text{ on } \Gamma_D,\ p_n(\mathbf{x}) = \beta(\mathbf{x}) \text{ on } \Gamma_N$$

For a finite difference discretization on a uniform grid of size h, we denote the sampling points as $\{i, j, k\} \in \mathcal{N}^3$. The sampling point corresponds to geometric location $\{ih, jh, kh\} \in \Omega$. The finite difference discretized operator for the interior can be written as:

$$\sum_{i',j',k' \in \mathcal{N}_{[i,j,k]}} \frac{p_{[i',j',k']} - p_{[i,j,k]}}{h^2} = f_{[i,j,k]} \tag{2.4}$$

$$\mathcal{N}_{[i,j,k]} = \{(i \pm 1, j, k), (i, j, k \pm 1), (i, j, k \pm 1)\}$$

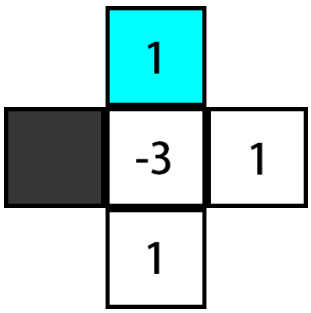


Figure 2.1: A finite difference Poisson stencil on the boundary. White cells are interior cells, i.e. they are degrees of freedom. Blue cells are Dirichlet conditions, or essential conditions, that their value are prescribed. Grey cells are exterior cells that are not part of the domain Ω. The face between gray and white cell describes a Nuemann condition, or a natural boundary.

It is a 2nd order accurate discretization. To show this, we can write in 1D:

$$\frac{\partial^2 p}{\partial x^2}(x_0) = \frac{(p(x_0 + h) - p(x_0)) - (p(x_0) - p(x_0 - h))}{h^2} + O(h^2)$$

As for boundary conditions, either Nuemann/nature boundary or Dirichlet/essential boundary, we can discretize the operator as shown in Figure 2.1. It is only 2nd order accurate, if the boundary condition is perfectly aligned with cell. But generally multigrid is used as preconditioner. For that purpose, this discretization is proven to be sufficient for preconditioning even with non-cell align boundaries by Aanjaneya et al. (2017).

But even though we may assume that the boundary conditions are cell aligned at the finest level, this may not hold true for the coarse levels, when the cell size doubles with each level. Figure 2.2 demonstrates the heuristic for coarsening boundary conditions. Note that after coarsening, the coarse level discretization may not be of the same order of accuracy to the continuous PDE as the finest level at the boundaries. These inconsistent boundary conditions break the principle that the each level of the multigrid hierarchy should be the discretization of the **same** continuous PDE. To

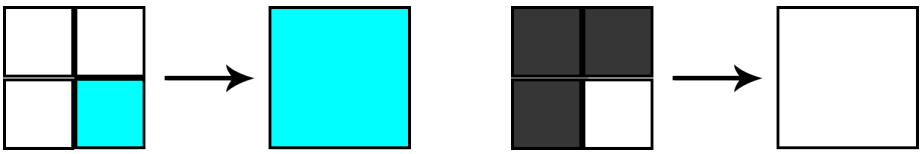


Figure 2.2: Geometric coarsening of the simulation domain. Left: if one of the cells is Dirichlet cell, we will coarsen it as Dirichlet cell at the coarse level, Right: otherwise if one of the cells is interior cell, it is coarsened as an interior cell at the coarse level.

compensate this discrepancy, additional boundary smoothing iterations need to be introduced to stabilize the multigrid.

2.4 Garlerkin Coarsening

Galerkin coarsening, also sometimes referred to as algebraic coarsening, is usually the preferred method for handling complex geometries or heterogeneous domains. It was summarized by Brandt (1986). Many algebraic algorithms assume only the top level linear operator is available in matrix form. In such case, the first step of the algorithm is to select the coarse grid DOF based on the matrix given. In this dissertation, we choose the coarsened DOF the same way as in geometric multigrid, that is, coarse DOF coincide with every other fine DOF in each dimension. If readers are interested in the coarse grid selection, we direct them to Brandt (1986), Vaněk et al. (1996), and Griebel et al. (2003).

After the coarse grid selection, a **prolongation** operator is constructed. It is a linear operator that interpolates fine DOF values from coarse DOF. The most common prolongation operator is multi-linear interpolation. In 1D, it can be illustrated as Figure 2.3.

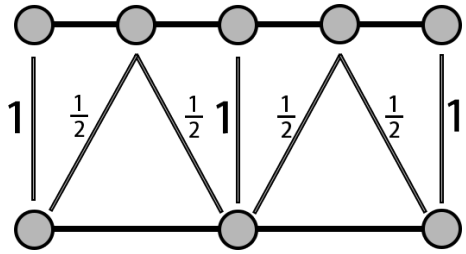


Figure 2.3: Multilinear prolongation operator in 1D. Top is the fine grid, on the bottom is the corresponding coarse grid. The value from corresponding fine nodes is copied from the coarse node, if they coincide in the domain. Otherwise the fine node value is interpolated from the two closest coarse nodes each with a weight of 0.5.

We can also write this in matrix form:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

The interpolation process is then:

$$\mathbf{u}^f = \mathbf{P}\mathbf{u}^c \tag{2.5}$$

Here $\mathbf{x}^f$ is the fine grid value and x^c is the coarse grid value. In the case of multi-level hierarchy, we can denote the prolongation operator from level $k+1$ to level k as $\mathbf{P}^k$. Therefore:

$$\mathbf{u}^k = \mathbf{P}^k\mathbf{u}^{k+1} \tag{2.6}$$

Here, P is a rank deficient matrix, which means when prolongating from a coarse level to a fine level, the vector at fine level has a higher dimensionality and is in general not the correct solution of the fine level discretization.

Therefore, smoothers are used to correct this error. This implies that the smoother should be selected based on error that is invisible to coarse grid, and vice versa, coarse grid should correct error modes that are hard to reduce by the smoother. In the next section we will discuss more this duality. Multilinear interpolation has proven to be effective prolongation operator for homogeneous and isotropic PDEs [McAdams et al. (2010); Aanjaneya et al. (2017); Zhu et al. (2010b)]. It is also the standard interpolation scheme for geometric multigrid.

Now we have defined the prolongation operator $\mathbf{P}^k$. The Galerkin coarsening process construct the $k+1$ operator $\mathbf{L}^{k+1}$ as follows:

$$\mathbf{L}^{k+1} = (\mathbf{P}^k)^T \mathbf{L}^k \mathbf{P}^k \tag{2.7}$$

The coarse level correction can be written as(for solving $\mathbf{Lu} = \mathbf{f}$):

$$\begin{aligned} \mathbf{r} &= \mathbf{b} - \mathbf{L}\mathbf{u}_0 \\ \mathbf{b}^c &= \mathbf{P}^T \mathbf{r} \\ \mathbf{u}^c &= (\mathbf{L}^c)^{-1} \mathbf{b}^c \\ \mathbf{u}^* &= \mathbf{u}_0 + \mathbf{P}\mathbf{u}^c \end{aligned}$$

Here, superscript c indicates unknowns and matrices from the coarse grid, $\mathbf{u}_0$ is the current guess of the fine level solution, and $\mathbf{u}^*$ is the new guess of the fine level solution after the coarse correction.

Lemma 2.1. *The coarse grid correction of a Galerkin coarsened hierarchy is a projection.*

Proof. This lemma states that when applying coarse correction twice, the second correction process will create the same result as first one.

Let us start with $\mathbf{u}^* = \mathbf{u}_0 + \mathbf{P}\mathbf{u}^c$.

$$\begin{aligned}
\mathbf{r}^* &= \mathbf{b} - \mathbf{L}\mathbf{u}^* \\
\mathbf{b}^{*c} &= \mathbf{P}^T\mathbf{r}^* \\
&= \mathbf{P}^T(\mathbf{b} - \mathbf{L}\mathbf{u}^*) \\
&= \mathbf{P}^T(\mathbf{b} - \mathbf{L}(\mathbf{u}_0 + \mathbf{P}\mathbf{u}^c)) \\
&= \mathbf{P}^T(\mathbf{b} - \mathbf{L}\mathbf{u}_0) - \mathbf{P}^T\mathbf{L}\mathbf{P}\mathbf{u}^c \\
&= \mathbf{P}^T\mathbf{r} - \mathbf{L}^c\mathbf{u}^c \\
&= \mathbf{b}^c - \mathbf{b}^c \\
&= 0
\end{aligned}$$

We can see that the second correction process will have 0 as right hand side. Therefore no correction will be produced. □

In general, we have $\mathrm{rank}(L^{k+1}) < \mathrm{rank}(L^k)$. Lemma 2.1 implies that the coarse grid correction is projecting the fine solution onto a subspace and it is solved within the subspace. All the fine error modes $\mathbf{e}$ that has the property $\mathbf{P}^T\mathbf{L}\mathbf{e} = \mathbf{0}$ will be invisible to the coarse level, and rely on the smoother for correction.

2.5 Multigrid V-cycle

Now we have constructed the hierarchy, let's take a look at a multigrid V-cycle.

Algorithm 2.1 Multigrid V-cycle

$\mathbf{u}^0 = \mathbf{0}$
$\mathbf{r}^0 = \mathbf{b}^0 - \mathbf{L}^0\mathbf{u}^0$
while $|\mathbf{r}^0| < \mathtt{threshold}$ || max_iteration has been reached **do**
  for i := 0 **to** k - 2 **do**
    smooth($\mathbf{u}^i, \mathbf{L}^i, \mathbf{b}^i$);
    $\mathbf{r}^i = \mathbf{b}^i - \mathbf{L}^i\mathbf{u}^i$
    $\mathbf{b}^{i+1} = (\mathbf{P}^i)^T\mathbf{r}^i$
  end for
  $\mathbf{u}^{k-1} = (\mathbf{L}^{k-1})^T\mathbf{b}^{k-1}$
  for i := k - 2 **to** 0 **do**
    $\mathbf{u}^i += \mathbf{P}^i\mathbf{u}^{i+1}$
    smooth($\mathbf{u}^i, \mathbf{L}^i, \mathbf{b}^i$);
  end for
  $\mathbf{r}^0 = \mathbf{b}^0 - \mathbf{L}^0\mathbf{u}^0$
end while

The process of rather straight forward: at each level, first, we reduce errors that are (hopefully) invisible to the coarse grid using a smoother. Then the residual of the remaining error is computed and restricted to the coarse level. This process is repeated until the bottom level is reached where the problem is small enough to be solved using a direct solver. This coarse level correction is than prolongated to each finer levels, then the smoother is applied again until this process reaches the top. The multigrid V-cycle can be repeated until satisfactory convergence is reached or the max number of iterations has reached.

This multigrid iteration has an asymptotic convergence, that is, after enough iterations, the ratio between the residual norm at iteration k and iteration $k+1$ will converge to a constant [Brandt (1977)]. After introducing the smoother in the next section, I will give more concrete examples of the metrics that are used for evaluating the asymptotic convergence rate.

2.6 Choices of Smoother

The term *smoother* generally refers to a class of iterative solvers that use only local information. The cost of each smoother application is comparable to a matrix multiplication. Common smoothers include (but not restricted to) Jacobi method, Gauss-Seidel method, Richardson iteration, and successive over-relaxation(SOR) method. As the simplest, I will take Richardson iteration as example. Richardson iteration can be written as (using notations from before):

$$\mathbf{r} = \mathbf{b} - \mathbf{L}\mathbf{u}_k$$
$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{r}$$

Here $\mathbf{u}_k$ is the solution at iteration k. If we define $\mathbf{u}^*$ the solution, that is

$$\mathbf{L}\mathbf{u}^* = \mathbf{b}$$

We can write the error of the current solution as

$$\mathbf{e}_k = \mathbf{u}^* - \mathbf{u}_k$$

And the residual is then

$$\mathbf{r}_k = \mathbf{L}\mathbf{e}_k$$

Assuming $\mathbf{L}$ is symmetric positive definite. We write the eigenvectors of $\mathbf{L}$ as $\mathbf{v}_i$, that are orthogonal to each other. Then we can write the error as combination of the eigenvectors:

$$\mathbf{e}_k = \sum_i \gamma_i^k \mathbf{v}_i$$

Here γ_i^k are the length when project $\mathbf{e}_k$ onto $\mathbf{v}_i$. We can rewrite the smoothing iteration as:

$$\begin{aligned}
\mathbf{u}_{k+1} &= \mathbf{u}_k + \mathbf{L}\mathbf{e}_k \\
\mathbf{u}_{k+1} - \mathbf{u}_k &= \mathbf{L}\mathbf{e}_k \\
(\mathbf{u}^* - \mathbf{u}_k) - (\mathbf{u}^* - \mathbf{u}_{k+1}) &= \mathbf{L}\mathbf{e}_k \\
\mathbf{e}_k - \mathbf{e}_{k+1} &= \mathbf{L}\sum_i \gamma_i^k \mathbf{v}_i \\
\mathbf{e}_k - \mathbf{e}_{k+1} &= \sum_i \gamma_i^k \mathbf{L}\mathbf{v}_i
\end{aligned}$$

We have $\mathbf{L}\mathbf{v}_i = \lambda_i \mathbf{v}_i$, where λ_i is the eigenvalue associated with eigenvector $\mathbf{v}_i$. Substitute it in:

$$\begin{aligned}
\mathbf{e}_k - \mathbf{e}_{k+1} &= \sum_i \gamma_i^k \lambda_i \mathbf{v}_i \\
\mathbf{e}_{k+1} &= \mathbf{e}_k - \sum_i \gamma_i^k \lambda_i \mathbf{v}_i \\
&= \sum_i \gamma_i^k \mathbf{v}_i - \sum_i \gamma_i^k \lambda_i \mathbf{v}_i \\
&= \sum_i \gamma_i^k (1 - \lambda_i)\mathbf{v}_i
\end{aligned}$$

We can write in the eigenspace components:

$$\gamma_i^{k+1} = \gamma_i^k (1 - \lambda_i) \tag{2.8}$$

That is, this smoother scales the error by each eigenspace component with a factor of $1 - \lambda_i$. That is, Richardson iteration can only converge if $\lambda_i < 1, \forall$ eigenvalues λ_i. The closer λ_i is to 1, faster that error mode is reduced. Other smoothers suppress eigenmodes differently, but in general, they follow the same principle: They are more effective in reducing error

modes associated with larger eigenvalues. Therefore, the error modes with smaller eigenvalues should be captured by the coarse grid. Intuitively, the smallest eigenvector that is not projected onto the coarse grid will be the bottleneck of the convergence, and dictates the asymptotic convergence rate. But the relation is not trivial. Given the coarse grid operator L^c is a subspace of the fine grid operator L^f. If we can decompose an error vector $\mathbf{e}^f$ into two parts $\mathbf{e}^{fc}$, such that $\exists \mathbf{e}^c$, satisfies $\mathbf{P}\mathbf{e}^c = \mathbf{e}^{fc}$, and the remainder $\mathbf{e}^{ff} = \mathbf{e}^f - \mathbf{e}^{fc}$. i.e.:

$$\mathbf{e}^f = \mathbf{e}^{fc} + \mathbf{e}^{ff}$$

A good smoother in this case, would be able to effectively reduce error mode $\mathbf{e}^{ff}$, even it may potentially increase error $\mathbf{e}^{fc}$.

2.7 Convergence Metrics

To better quantify multigrid's convergence property, mathematically, it can be measured by two metrics, summarized in Brezina et al. (2001):

$$M_1(\mathbf{Q}, \mathbf{e}) := \frac{< (\mathbf{I} - \mathbf{Q})\mathbf{e}, (\mathbf{I} - \mathbf{Q})\mathbf{e} >}{< \mathbf{L}\mathbf{e}, \mathbf{e} >} \tag{2.9}$$

$$M_2(\mathbf{Q}, \mathbf{e}) := \frac{< \mathbf{L}(\mathbf{I} - \mathbf{Q})\mathbf{e}, (\mathbf{I} - \mathbf{Q})\mathbf{e} >}{< \mathbf{L}\mathbf{e}, \mathbf{e} >} \tag{2.10}$$

M_2 is used by McCormick (1984); McCormick and Ruge (1982); McCormick (1985). M_1 is introduced by Brandt (1986). The metric measures how well the prolongation operator is for a given error $\mathbf{e}$. $\mathbf{Q}$ is a projection operator that projects a fine vector $\mathbf{e}^f$ onto a subspace that can be resolved by the coarse grid, $\mathbf{e}^{fc}$.

$$\mathbf{e}^{fc} = \mathbf{Q}\mathbf{e}^f$$

Here is the formal definition of $\mathbf{Q}$:

$$\mathbf{Q} = \mathbf{PR} \tag{2.11}$$

$\mathbf{P}$ is the prolongation operator defined in the previous sections. $\mathbf{R}$ is a restriction operator that computes a coarse vector $\mathbf{e}^c$ from a fine vecotor $\mathbf{e}^f$. In the context of Galerkin coarsening, if our fine grid solution is $\mathbf{u}^*$ with initial guess of $\mathbf{u}^0 = \mathbf{0}$. The coarse grid vector $\mathbf{u}^c$ can be computed as follows:

$$\begin{aligned}
\mathbf{r}^f &= \mathbf{L}^f\mathbf{u}^* \\
\mathbf{b}^c &= \mathbf{P}^T\mathbf{r}^f \\
\mathbf{u}^c &= (\mathbf{L}^c)^{-1}\mathbf{b}^c \\
&= (\mathbf{P}^T\mathbf{L}^f\mathbf{P})^{-1}\mathbf{b}^c \\
&= (\mathbf{P}^T\mathbf{L}^f\mathbf{P})^{-1}\mathbf{P}^T\mathbf{r}^f \\
&= (\mathbf{P}^T\mathbf{L}^f\mathbf{P})^{-1}\mathbf{P}^T\mathbf{L}^f\mathbf{u}^* \\
\mathbf{u}^c &= \mathbf{R}\mathbf{u}^*
\end{aligned}$$

So we can derive:

$$\mathbf{R} = (\mathbf{P}^T\mathbf{L}^f\mathbf{P})^{-1}\mathbf{P}^T\mathbf{L}^f \tag{2.12}$$

By using this definition of $\mathbf{R}$, we have $\mathbf{RP} = \mathbf{I}_c$. $\mathbf{I}_c$ is the identity matrix of the same dimension of number of DOF in the coarse grid. That is, for error modes that can be resolved in the coarse grid, i.e. error modes can

be written as $\mathbf{e}^f = \mathbf{P}\mathbf{e}^c$. We have:

$$\begin{aligned}
\mathbf{Q}\mathbf{e}^f &= \mathbf{P}\mathbf{R}\mathbf{e}^f \\
&= \mathbf{P}\mathbf{R}\mathbf{P}\mathbf{e}^c \\
&= \mathbf{P}(\mathbf{P}^T\mathbf{L}^f\mathbf{P})^{-1}\mathbf{P}^T\mathbf{L}^f\mathbf{P}\mathbf{e}^c \\
&= \mathbf{P}(\mathbf{L}^c)^{-1}\mathbf{L}^c\mathbf{e}^c \\
&= \mathbf{P}\mathbf{I}\mathbf{e}^c \\
&= \mathbf{P}\mathbf{e}^c \\
&= \mathbf{e}^f
\end{aligned}$$

We can see that

$$(\mathbf{I} - \mathbf{Q})\mathbf{e}^f = \mathbf{0}, \text{ if } \exists\ \mathbf{e}^c, \text{ s.t. } \mathbf{e}^f = \mathbf{P}\mathbf{e}^c \tag{2.13}$$

But unfortunately, using this definition of $\mathbf{R}$ for analysis often is far too computationally expensive as it requires the inversion of the coarse level operator $(\mathbf{L}^c)^{-1}$. Therefore, in work by Brezina et al. (2001), an alternative definition of $\mathbf{R}$ is used:

$$\mathbf{R}_{ij} = \begin{cases} 1, \text{ if } \mathbf{X}^f_j = \mathbf{X}^c_i \\ 0, \text{ otherwise} \end{cases} \tag{2.14}$$

There, $\mathbf{R}$ is a simple injection. That is with value 1, where the coarse node and fine node coincide in space, i.e. $\mathbf{X}^f_j = \mathbf{X}^c_i$, where $\mathbf{X}^f_j$ is the spacial location of fine node j and $\mathbf{X}^c_i$ is the spacial location of coarse node i.

Lemma 2.2. *The injection based* $\mathbf{R}$ *creates a valid projection* $\mathbf{Q}$ *that satisfies Equation 2.13.*

Proof. We can rearrange the fine DOFs such that fine DOFs that does not spatially coincide with any coarse node come first, then those that coincide

with coarse node come after. We can write the fine vector $\mathbf{e}$ as:

$$\mathbf{e} = \begin{bmatrix} \mathbf{e}_{ff} \\ \mathbf{e}_{fc} \end{bmatrix}$$

Accordingly, $\mathbf{R}$ and $\mathbf{P}$ can be written as:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{cf} \\ \mathbf{I}_c \end{bmatrix}$$
$$\mathbf{R} = \begin{bmatrix} \mathbf{0} & \mathbf{I}_c \end{bmatrix}$$

$\mathbf{Q}$ therefore become:

$$\begin{aligned} \mathbf{Q} &= \mathbf{PR} \\ &= \begin{bmatrix} \mathbf{0} & \mathbf{P}_{cf} \\ \mathbf{0} & \mathbf{I}_c \end{bmatrix} \end{aligned}$$

Plugging into Equation 2.13:

$$\begin{aligned} (\mathbf{I} - \mathbf{Q})\mathbf{e} &= \begin{bmatrix} \mathbf{I} & -\mathbf{P}_{cf} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{e}_{ff} \\ \mathbf{e}_{fc} \end{bmatrix} \\ &= \mathbf{e}_{ff} - \mathbf{P}_{cf}\mathbf{e}_{fc} \end{aligned} \tag{2.15}$$

If $\exists \mathbf{e}_c$ s.t. $\mathbf{e} = \mathbf{P}\mathbf{e}_c$:

$$\begin{aligned}
\mathbf{P}\mathbf{e}_c &= \begin{bmatrix} \mathbf{P}_{cf} \\ \mathbf{I}_c \end{bmatrix} \mathbf{e}_c \\
&= \begin{bmatrix} \mathbf{P}_{cf}\mathbf{e}_c \\ \mathbf{e}_c \end{bmatrix} \\
&= \mathbf{e} \\
&= \begin{bmatrix} \mathbf{e}_{ff} \\ \mathbf{e}_{fc} \end{bmatrix}
\end{aligned}$$

Therefore:

$$\begin{aligned}
\mathbf{e}_{ff} &= \mathbf{P}_{cf}\mathbf{e}_c \\
\mathbf{e}_{fc} &= \mathbf{e}_c
\end{aligned}$$

Plug into Equation 2.15:

$$\begin{aligned}
(\mathbf{I} - \mathbf{Q})\mathbf{e} &= \mathbf{e}_{ff} - \mathbf{P}_{cf}\mathbf{e}_{fc} \\
&= \mathbf{P}_{cf}\mathbf{e}_c - \mathbf{P}_{cf}\mathbf{e}_{fc} \\
&= \mathbf{0}
\end{aligned}$$

This concludes the proof. □

Lemma 2.2 states that though the $\mathbf{Q}$ defined above does not reflect the real process of acquiring the coarse grid correction, it is still a valid operator that projects a fine vector $\mathbf{e}$ onto the range of the prolongation operator $\mathbf{P}$. Then, in the metrics M_1 and M_2, the inner products $< (\mathbf{I}-\mathbf{Q})\mathbf{e}, (\mathbf{I}-\mathbf{Q})\mathbf{e} >$ and $< \mathbf{L}(\mathbf{I}-\mathbf{Q})\mathbf{e}, (\mathbf{I}-\mathbf{Q})\mathbf{e} >$ measure how well the prolongation operator captures a given vector $\mathbf{e}$, either in terms of L^2 norm or the energy norm $\mathbf{u}^T\mathbf{L}\mathbf{u}$. Note that if $\mathbf{e}$ can be captured exactly by the coarse grid, both M_1 and M_2 are zero. On the denominator, $< \mathbf{L}\mathbf{e}, \mathbf{e} >$ measures effectiveness

of the smoother. In general, the larger the energy norm of an error vector is, the more effective the smoother is. A simple example here: if $\mathbf{e}$ is one of the eigenvector, the energy norm is square of the eigenvalue associated with the eigenvector. Therefore smaller M_1 and M_2 are, better the quality of prolongation is. So we can write the prolongation as a minimization problem:

$$\underset{\mathbf{P}}{\operatorname{argmin}} \max_{\mathbf{e}} M_i(\mathbf{P}, \mathbf{e}), \text{ subject to } ||\mathbf{e}||_2 = 1 \tag{2.16}$$

Here i can be either 1 or 2 depends on which metric is used. But in reality other constraints will need to be imposed on $\mathbf{P}$ to limit its sparsity and the sparsity of a coarse level operator. In Chapter 5, the construction of the prolongation $\mathbf{P}$ operator will be explored under the context of FEM discretized linear elasticity. But let us first examine how can we deploy multigrid algorithms utilizing heterogeneous platforms.

3 A SCHUR-COMPLEMENT DOMAIN DECOMPOSITION SOLVER FOR MULTI-ACCELERATOR EQUIPPED PLATFORM

3.1 Features of Multi-accelerator Equipped Platforms

Modern compute platforms for scientific computing are evaluated based on their compute power in FLOPS(FLoating Operation Per Second) and memory bandwidth in GB/s (GibaByte Per Second). A single CPU equipped machine, Intel® Xeon® Gold 6150 Processor for example, have 1.3 TFLOP compute power and 120 GB/s memory bandwidth with maximum of 768GB of memory. A multi-GPU equipped machine, with 4 GTX 1080TI for instance, can have an aggregately 44 TFLOP compute power and 1936 GB/s memory bandwidth, but only 44 GB of total memory at the same cost. This 33x compute power and 8x memory bandwidth difference does not come without any trade offs. Each accelerator can only operate at peak performance for the data allocated at its local memory. Cross GPU data access and inter CPU/GPU data access are significantly slower not only in terms of bandwidth but also in terms of latency. PCI-E 3.0 16x bus has a bandwidth of 16GB/s, NVLink 1.0 has a bandwidth 80 GB/s, NVLink 2.0 has 150 GB/s bandwidth. A heterogeneous platform often refers to those type of machines in which not all resources can be accessed at the same cost across the platform. For instance, a GPU (GTX 1080TI as an example) may be able to access data allocated on its local memory at 484GB/s, but when accessing data allocated on CPU, it would require communication across PCI-E at maximum speed 16GB/s with couple of microseconds latency.

When designing algorithms for a heterogeneous platform, the locality of data needs to be kept in mind for best utilization of the computing

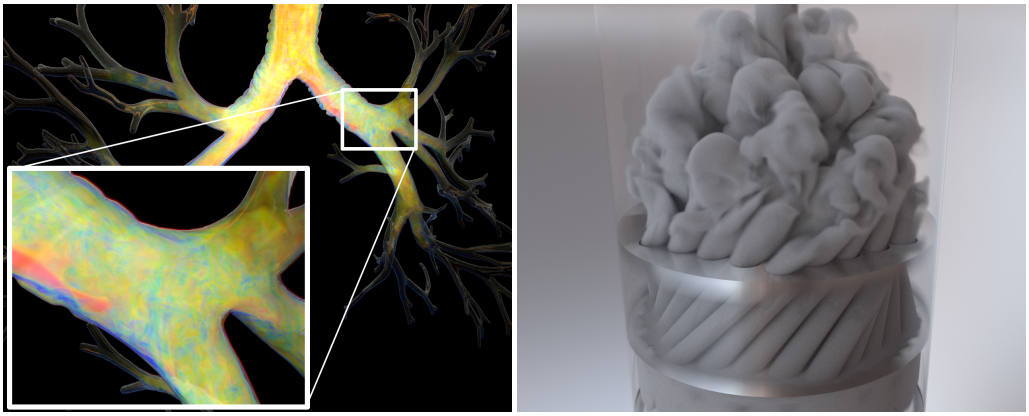


Figure 3.1: Left: Smoke injected into a model of the bronchi. Color illustrates vorticity magnitude. Simulation contains 1.8 billion active cells, sparsely occupying a $8192^2 \times 4096$ background grid. Right: Smoke injected from the bottom of a cylinder, and forced through a metal gasket (rendered semi-transparent) with a twisted bundle of cylindrical holes. Total of 1.2 billion active cells, in a $1024^2 \times 2048$ background grid.

resources. Otherwise, it will be limited by the interconnecting bus. In such case, the single CPU platform with 140 GB/s memory bandwidth may outperform this multi-GPU equipped machine that is limited by its 16GB/s PCI-E bus. It should be noted that this non-uniformity of bandwidth is not an aberration of hardware design that is likely going away; in fact it is consistent with the fact that memory hierarchies, and physical proximity of memory to computational units, have long given rise to differentiation in latency and bandwidth between different parts of the computational platform.

Unfortunately, some of the best performing solvers (in terms of convergence efficiency) such as multigrid are the cases of the worst scenarios. The building blocks of a multigrid solver, namely the smoothing routine and transfer operators, require global synchronization after their execution while carrying out only a very modest amount of useful computation in between synchronization points. In particular, a Jacobi or Gauss-Seidel

style smoother requires no more than two passes over the data in memory, which completes in a very small fraction of the cost incurred for transferring the data to the GPU card, or out of it upon completion. Of course, one could take the opportunity to carry out several smoothing iterations per offload operation; however, without synchronization at partition boundaries this extra effort will hardly translate to worthwhile gains in convergence. As a result, the benefit of the GPU offload is negated, and such large problems are better off being solved homogeneously on the CPU. Although there might be room for implementation refinements and adaptation of multigrid paradigms to curb this overhead, we are not aware of prior work that has demonstrated viability of a GPU-offload paradigm for multigrid solvers, when the problem size exceeds the memory capacity of the GPU card(s), compare to a well-optimized CPU implementation. Our proposed approach directly addresses this challenge: instead of executing just a few iterations of a smoother routine, we run an entire solver routine on the GPU for each independent subdomain we offload to it. In our case, that extra effort does translate to accelerated convergence, and the GPU computation is long enough to absorb the transfer cost.

In this chapter, we will examine a domain decomposition solver for fluid simulation, that is specifically designed for heterogeneous platform by minimizing the need for cross accelerator communication.

3.2 Related Work

Fluid simulation has been an active area of research within computer graphics since the early work of Stam (1999); Foster and Fedkiw (2001). Since the memory overhead associated with uniform grids quickly escalates in three spatial dimensions, several adaptive techniques have been proposed, including adaptive Cartesian grids Losasso et al. (2004); Zhu et al. (2013); Ferstl et al. (2014a); Setaluri et al. (2014), adaptive tetrahe-

dral meshes Klingner et al. (2006); Chentanez et al. (2007); Ando et al. (2013), RLE-based schemes Houston et al. (2006); Irving et al. (2006); Chentanez and Müller (2011), adaptive mesh refinement (AMR) and chimera grid schemes Dobashi et al. (2008); Tan et al. (2008); Cohen et al. (2010); English et al. (2013a). Lagrangian methods present an interesting alternative because they avoid many of the numerical dissipation issues characteristic of Eulerian methods. Several methods have been proposed, including smoothed particle hydrodynamics (SPH) Solenthaler and Gross (2011); Ihmsen et al. (2014); Bender and Koschier (2015), particle-based schemes Adams et al. (2007); de Goes et al. (2015), position-based fluids Macklin and Müller (2013), triangle meshes Wojtan et al. (2010); Thürey et al. (2010); Da et al. (2015) and simplicial complexes Zhu et al. (2014). However, due to their unstructured nature, these methods are unable to leverage the regularity and parallelism potential of uniform grids. To circumvent this issue to some extent, hybrid methods have also been proposed Foster and Metaxas (1996); Zhu and Bridson (2005); Losasso et al. (2008); Zhu et al. (2010a); Raveendran et al. (2011); Jiang et al. (2015); Chen et al. (2015). Authors have also investigated the use of Fast Fourier transforms Stam (2002), which was later extended to handle slip boundary conditions Long and Reinhard (2009), model reduction Liu et al. (2015), and regression forests Ladický et al. (2015).

The pressure projection step is widely accepted to be the computationally dominating step in fluid simulations, and researchers have investigated the design of fast solvers using coarse grids Lentine et al. (2010), multigrid methods McAdams et al. (2010); Chentanez and Müller (2011); Setaluri et al. (2014); Dick et al. (2016), iterated orthogonal projection Molemaker et al. (2008), dimension reduction Ando et al. (2015b), fast summation methods Zhang and Bridson (2014), and stream functions Ando et al. (2015a). The concept of warm starts was recently explored in Hecht et al. (2012)by exploiting sparsity patterns in the Cholesky factorization in ways

analogous to our method, albeit in the context of elasticity simulations. An increasing number of researchers have also adopted the use of GPUs for better computational performance since efficient solvers such as multigrid tend to be memory-bound Ament et al. (2010); Dick et al. (2011b); Zhang and Bridson (2014); Chen et al. (2015); Wu et al. (2016a).

Unlike previous approaches where the goal was to increase performance on homogeneous platforms, we use domain decomposition techniques to develop an efficient Krylov preconditioner whose design is tailored towards maximizing performance on heterogeneous computing platforms. One earlier investigation that does address heterogeneity is Jung et al. (2013), which proposed a wavelet-based method that used GPUs for increasing the performance of a multigrid solver hosted on the CPU. While researchers have proposed methods classified as domain decomposition Golas et al. (2012); Edwards and Bridson (2015), these are quite different from ours because we work specifically in the context of Schur complement methods Smith et al. (1996). Methods based on Schur complements have been used for virtual surgery simulations Bro-nielsen and Cotin (1996), skinning Gao et al. (2014), and subspace deformable body simulations Teng et al. (2015); Wu et al. (2015), or fluid control Raveendran et al. (2012).

3.3 Domain Decomposition as Divide-and-Conquer

In this chapter, a Schur Complement domain decomposition method is presented, the objective is to demonstrate an effective, and hopefully inspiring adaptation to fluids simulation of a class of numerical techniques that has received much more exposure in scientific computing than graphics research. We note, however, that Schur Complement methods provide a general framework, and not just a singular algorithm; in fact, similar to

multigrid methods, careful variations from the general algebraic theme make all the difference between a given scheme being highly effective or underwhelming for a specific application. In this vein, we consciously restricted the scope of our investigation to just uniform discretizations of fluids, specifically targeted the Poisson equation (although our formulations should readily extend to elasticity, or other elliptic problems see, for example, Chapter 4 and Chapter 5), and did not emphasize the implications of heterogeneous computing to other parts of the fluid simulation pipeline.

The classical divide-and-conquer paradigm encountered in combinatorial algorithms often presumes building blocks that accurately solve subsets of the overall problem. In our numerical context, one of the key opportunities we will exploit is the option to design what is not an exact solver, but an excellent approximation of one, and subsequently use it as a preconditioner. In this context, we will adopt a slightly different standard which we will design our "inaccurate" divide-and-conquer scheme to satisfy. Our building block will be an inexact solver for the Poisson equation on independent partitions of our domain, which is however of good enough quality to be used as an excellent Conjugate Gradients preconditioner (i.e. it would lead to convergence in a small number of iterations, that does not significantly increase as the subdomain size grows larger). Subsequently, our objective would be to combine such "nearly accurate" building blocks into a global approximate solver that meets the same benchmark, i.e. it can be used as a highly effective preconditioner that allows CG to converge in a comparably small number of iterations as its individual constituents. We note that,short of this standard, using divide-and-conquer tricks can be a slippery slope and result is significantly degraded performance.

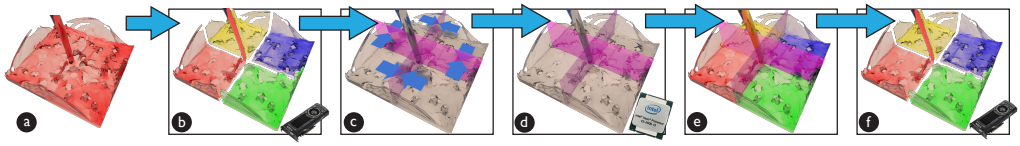


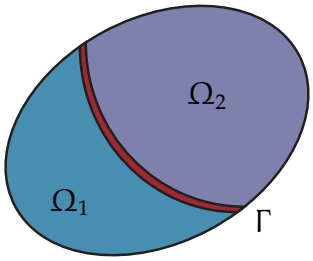
Figure 3.2: Illustration of the core concept of our method: (b) We split the computational grid into subdomains, and independently solve them on the GPU(s), using zero Dirichlet conditions on subdomain boundaries. (c) Fluxes of the subdomain solutions are computed and sent to the CPU. (d) A specially formulated system is solved on the interface, using the CPU. This produces the exact value of the interface variables. (e) Those values are sent to the subdomains, and set as Dirichlet conditions. (e) A final subdomain solve on the GPU yields the global solution.

3.4 The Classic Schur Complement Method

We introduce the basic principles of the Schur complement method [Quarteroni and Valli (1999)] by explaining how an aggregate solver for the pressure Poisson equation can be assembled using as subroutines two independent solvers for two non-overlapping partitions of the entire computational domain. After covering the basic theory we will detail how this construction extends to multiple partitions, and derive a preconditioner based on this concept in later sections.

The two-subdomain case

Consider a domain Ω that has been partitioned into two subdomains Ω_1 and Ω_2 through an interface region Γ. Let us assume we have a finite-difference discretization of the pressure Poisson equation on Ω, and that the interfacial region Γ is thick enough to shield any stencil in Ω_1 from including a point in Ω_2 (and vice-versa). In practice, when using the standard 7-point stencil in a Cartesian discretization, the interface layer Γ can simply



be one-node thick as long as it cleanly decouples Ω into two distinct subdomains (although Γ could also be made wider, if desired). For simplicity of notation we will write the Poisson equation as $Ax = b$, with the understanding that the vector x contains the unknown pressure values and b contains the respective divergence values of the velocity field. We then reorder degrees of freedom as:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_\Gamma \end{bmatrix}, \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_\Gamma \end{bmatrix} \tag{3.1}$$

where x_i, b_i correspond to values in Ω_i, for $i \in \{1, 2\}$ (and similarly x_Γ, b_Γ correspond to degrees of freedom in Γ). Under this reordering, the matrix A assumes the following block form:

$$A = \begin{bmatrix} A_{11} & & A_{1\Gamma} \\ & A_{22} & A_{2\Gamma} \\ A_{\Gamma 1} & A_{\Gamma 2} & A_{\Gamma\Gamma} \end{bmatrix} \tag{3.2}$$

Note that due to symmetry of A, we have $A_{\Gamma 1}^T = A_{1\Gamma}$ and $A_{\Gamma 2}^T = A_{2\Gamma}$ for the off-diagonal blocks. Using this block form of A it is possible to write the following factorization of the *inverse* matrix A^{-1}:

$$A^{-1} = \underbrace{\begin{bmatrix} I & & -A_{11}^{-1}A_{1\Gamma} \\ & I & -A_{22}^{-1}A_{2\Gamma} \\ & & I \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} A_{11}^{-1} & & \\ & A_{22}^{-1} & \\ & & \Sigma^{-1} \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} I & & \\ & I & \\ -A_{\Gamma 1}A_{11}^{-1} & -A_{\Gamma 2}A_{22}^{-1} & I \end{bmatrix}}_{U^T}$$

where

$$\Sigma = A_{\Gamma\Gamma} - A_{\Gamma 1}A_{11}^{-1}A_{1\Gamma} - A_{\Gamma 2}A_{22}^{-1}A_{2\Gamma}$$

is the *Schur complement* of the block $A_{\Gamma\Gamma}$ in equation (3.2). The validity of this factorization can be verified via a direct substitution into the identity

$A \cdot A^{-1} = I$. Finally, since A (and its inverse) is a symmetric positive definite(SPD) matrix, this factorization implies that the Schur complement is also symmetric and positive definite (the matrix D is equal to the symmetric conjugation of the symmetric positive definite matrix A^{-1} with the matrix U^{-1}; hence its diagonal sub-block Σ^{-1} is symmetric definite, too).

$$A^{-1} = \begin{bmatrix} I & & -A_{11}^{-1}A_{1\Gamma} \\ & \ddots & \vdots \\ & I & -A_{kk}^{-1}A_{k\Gamma} \\ & & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & & & \\ & \ddots & & \\ & & A_{kk}^{-1} & \\ & & & \Sigma^{-1} \end{bmatrix} \begin{bmatrix} I & & & \\ & \ddots & & \\ & & I & \\ -A_{\Gamma 1}A_{11}^{-1} & \dots & -A_{\Gamma k}A_{kk}^{-1} & I \end{bmatrix} \tag{3.3}$$

$$= \begin{bmatrix} A_{11}^{-1} & & & \\ & \ddots & & \\ & & A_{kk}^{-1} & \\ & & & I \end{bmatrix} \begin{bmatrix} I & & & -A_{1\Gamma} \\ & \ddots & & \vdots \\ & & I & -A_{k\Gamma} \\ & & & I \end{bmatrix} \begin{bmatrix} A_{11} & & & \\ & \ddots & & \\ & & A_{kk} & \\ & & & \Sigma^{-1} \end{bmatrix} \begin{bmatrix} I & & & \\ & \ddots & & \\ & & I & \\ -A_{\Gamma 1} & \dots & -A_{\Gamma k} & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & & & \\ & \ddots & & \\ & & A_{kk}^{-1} & \\ & & & I \end{bmatrix} \tag{3.4}$$

$$\approx \begin{bmatrix} A_{11}^{\dagger} & & & \\ & \ddots & & \\ & & A_{kk}^{\dagger} & \\ & & & I \end{bmatrix} \begin{bmatrix} I & & & -A_{1\Gamma} \\ & \ddots & & \vdots \\ & & I & -A_{k\Gamma} \\ & & & I \end{bmatrix} \begin{bmatrix} I & & & \\ & \ddots & & \\ & & I & \\ & & & \Sigma^{\dagger} \end{bmatrix} \left\{ I + \begin{bmatrix} & & & \\ & & & \\ & & & \\ -A_{\Gamma 1} & \dots & -A_{\Gamma k} & I \end{bmatrix} \begin{bmatrix} A_{11}^{\dagger} & & & \\ & \ddots & & \\ & & A_{kk}^{\dagger} & \\ & & & I \end{bmatrix} \right\} \tag{3.5}$$

The multiple subdomain solver

This formulation extends naturally to an arbitrary number of k subdomain partitions $\Omega_1, \dots, \Omega_k$ separated by an interface set Γ (figure 3.2 depicts such a partitioning into four subdomains, with the interface Γ highlighted as the magenta-colored separator surface). The corresponding factorization of A^{-1} in this case is given in equation (3.3). In order to translate this algebraic expression into a solver algorithm, we first re-factor this into the five-matrix product of equation (3.4), which has every subdomain inverse A_{ii}^{-1} appear only twice (as opposed to three inversions per subdomain, in equation 3.3). The last algebraic manipulation, as given in equation (3.5) further avoids the appearance of the subdomain Laplacian A_{ii}, requiring only the inverses of such matrices. In this expression we have also substituted

the symbol $M^{\dagger} \approx M^{-1}$ for *approximate* inverses of A_{ii} and Σ. If the *exact* inverse of these matrices was used, equation (3.5) becomes identically equal to the five-factor expression of equation (3.4). We will later engage in such approximations; for now, we may assume that all these inverses are exact.

The Schur complement method effectively solves the equation $Ax = b$ by multiplying the right hand side b with the factorized equivalent of A^{-1} from equation (3.5). The key observation is that we can apply this multiplication indirectly, without explicitly constructing the matrix in this factorization. We do this as follows:

1. Solve k subproblems: $A_{11}\hat{x}_1 = b_1$, ..., $A_{kk}\hat{x}_k = b_k$.
2. Solve $\Sigma x_\Gamma = b_\Gamma - A_{\Gamma 1}\hat{x}_1 - A_{\Gamma 2}\hat{x}_2 - \ldots - A_{\Gamma k}\hat{x}_k$.
3. Solve the k new subproblems
   $A_{11}\delta x_1 = -A_{1\Gamma}x_\Gamma$, ..., $A_{kk}\delta x_k = -A_{k\Gamma}x_\Gamma$.
4. Update $x_1 \leftarrow \hat{x}_1 + \delta x_1$, ..., $x_k \leftarrow \hat{x}_k + \delta x_k$.

Observe that steps (1) and (3) require the solution of fully decoupled systems for each subdomain Ω_i, and this can easily be performed in parallel without any need for communication or synchronization. Step (2) requires the solution of a symmetric and positive definite system (with the Schur complement Σ as the coefficient matrix). Traditionally, solvers based on this method attempt to solve this interface system using a preconditioned Krylov subspace method such as Conjugate Gradients. We will deviate from this practice, and use equation (3.5) instead, to design a preconditioner for the *global* (coupled) system.

It is important to examine the algebraic structure of Σ, and assess the performance implications of attempting to solve the system in step (2) directly. For a volumetric domain Ω with N total degrees of freedom, the dimensionality of Γ would be $O(\sqrt{N})$ in 2D, and $O(N^{2/3})$ in 3D. Note,

however, that in contrast to the sparse Laplace matrix A, the Schur complement is a dense matrix, thus having $O(N^{4/3})$ entries and requiring at least as much computation to solve. Asymptotically, this would make step (2) above by far the bottleneck of the solver, if Σ was to be explicitly constructed. Furthermore, the construction of the matrix alone would likely require even more computation, as it would need to account for computing the subdomain inverses A_{ii}^{-1}. Using Conjugate Gradients as the solver in step (2) opens up an interesting possibility: the CG algorithm does not need an explicitly constructed matrix Σ, as long as we have a way to compute matrix-vector products Σx_Γ. In turn, this would require computing products of the form $A_{\Gamma i}A_{ii}^{-1}A_{i\Gamma}x_\Gamma$ as efficiently as possible. Although the factors $A_{\Gamma i}$, $A_{i\Gamma}$ are sparse enough to allow efficient multiplication, multiplying with A_{ii}^{-1} (i.e. solving a subdomain Poisson problem) requires at least linear cost relative to the size of the subdomain (assuming a linear-complexity solver, like an extremely well built multigrid scheme, iterated to full convergence). There would be opportunity for parallelization across subdomains, but we would be still confronted with a linear complexity cost for *each* CG iteration, and we would have to rely on constructing an extremely efficient preconditioner to ensure that only a small finite number of iterations would suffice, independent of resolution. Nevertheless, this is the path followed by many derivative techniques of the Schur complement method (often referred to as *iterative substructuring*; an excellent synopsis of such options is given in the classic book by Quarteroni and Valli (1999)).

3.5 A Schur-Complement Preconditioner

In light of the challenges detailed in section 3.4 we propose certain strategic simplifications that would make the Schur complement method yield an approximate solver of the Poisson equation, rather than a strictly accurate

one. Our intent would be to use this approximation as a preconditioner for the Conjugate Gradients method, applied to the full-scale Poisson problem. Our last transformation of the factorized form for A^{-1}, captured in equation (3.5) was precisely intended to facilitate this process. We can easily show that any *nonsingular* approximation $A_{ii}^{\dagger} \approx A_{ii}^{-1}$ of the subdomain inverses, combined with a *symmetric positive definite (SPD)* approximation $\Sigma^{\dagger} \approx \Sigma^{-1}$ will produce, after substitution in equation (3.5), a symmetric and positive definite matrix approximation to A^{-1}. Thus multiplication with this expression can be used as a preconditioner for the Conjugate Gradients method.

From an implementation standpoint, we map the application of this preconditioner to a heterogeneous platform by assigning the interior degrees of freedom of each subdomain Ω_i to a single GPU or Many-Core accelerator card, while the interface degrees of freedom (Γ) will be maintained on the CPU. We design the approximate subdomain inverses $A_{ii}^{\dagger}$ so that they can be multiplied with respective vectors exclusively on the GPU, local to the accelerator that owns the subdomain Ω_i. Multiplication with the matrix blocks $A_{\Gamma i}$ will coincide with data transfer from the card that owns Ω_i to the CPU, while multiplication with the transpose $A_{i\Gamma}$ will relay data from the CPU to the respective accelerator in the opposite direction. Multiplication with the approximate inverse of the Schur complement, i.e. $\Sigma^{\dagger}$, will be handled fully on the CPU. The application of this preconditioner is formalized in pseudocode in Algorithm 1.

Multigrid subdomain solver

For approximating $A_{ii}^{\dagger}$ in the formulation described above, we use a simple, voxel-accurate multigrid solver in the spirit similar to prior works McAdams et al. (2010); Molemaker et al. (2008), with some embellishments to support sparsely populated domains as discussed in section 3.7. Multigrid solver is selected for its linear cost of each iteration, and its ability

Algorithm 3.1 Preconditioner application $\boldsymbol{z} = A^{\dagger}\mathbf{r}$, from eqn. (3.5)

for $i = 1 \dots k$ **do**
  {In parallel, on GPU}
  Get $r_i \leftarrow$ CPU
  Solve $q_i \leftarrow A_{ii}^{\dagger} r_i$
  Compute $s_{\Gamma}^{(i)} \leftarrow -A_{\Gamma i} q_i$
  Send $s_{\Gamma}^{(i)} \rightarrow$ CPU
  Send $q_i \rightarrow$ CPU
end for
Compute $f_{\Gamma} = r_{\Gamma} + s_{\Gamma}^{(1)} + \ldots + s_{\Gamma}^{(k)}$ {on CPU}
Solve $z_{\Gamma} \leftarrow \Sigma^{\dagger} f_{\Gamma}$
for $i = 1 \dots k$ **do**
  {In parallel, on GPU}
  Get $q_i \leftarrow$ CPU
  Get $z_{\Gamma} \leftarrow$ CPU
  Compute $f_i \leftarrow -A_{i\Gamma} z_{\Gamma}$
  Solve $z_i \leftarrow A_{ii}^{\dagger} f_i$
  Add $z_i += q_i$
  Send $z_i \rightarrow$ CPU
end for

to be implemented in a matrix-free fashion to save memory. This is also the reason that sparse linear system solvers such as *cuSparse* by Nvidia (2014) was not used. The multigrid hierarchy is constructed by classifying every grid cell as "interior", "exterior" (to the active domain) or "Dirichlet", and coarsening this classification to voxels of lower resolution grids. Trilinear transfer operators and a damped Jacobi smoother are employed, with an additional smoothing effort devoted to a narrow band around the boundary (3-7 iterations) for each interior smoothing pass.

3.6 The interface Schur-complement system

The last remaining piece for generating our preconditioner is the design of the approximation $\Sigma^{\dagger}$ and the application of its effective numerical solution which is hosted exclusively on the CPU. As previously stated, our objective is to arrive at an algorithm that is sublinear in complexity relative to the size of the overall boundary and yields a good approximation to the exact matrix $\Sigma = A_{\Gamma\Gamma} - \sum_{i=1}^{k} A_{\Gamma i} A_{ii}^{-1} A_{i\Gamma}$. We will solve the approximate system $\Sigma^{\dagger} x_{\Gamma} = b_{\Gamma}$ using a multigrid method, which will avoid forming the dense matrix $\Sigma^{\dagger}$ explicitly. Furthermore, we will leverage adaptive coarsening of the subdomains to reduce the dimensionality of the direct algebra involved in our manipulations.

Multigrid solver for the interface

The Schur complement matrix Σ is not only symmetric and positive definite, but it can further be shown that it is an *elliptic* operator, as it is a discretization of the continuous and elliptic *Steklov-Poincaré operator* for the Poisson equation [Smith et al. (1996)]. This suggests that a multigrid solver (on the interface variables; separate from the multigrid cycles used to approximate the subdomain inverses) could be applicable. The simplest technique for building the multigrid hierarchy is to use Galerkin coarsening to construct the operator at each resolution level. However, we do not pursue this option as it requires explicitly computing the matrix Σ at the finest level, which is a computationally expensive proposition. We propose a different conceptual construction of the operator hierarchy, and design a smoother that can use them indirectly without assembling their explicit matrix form.

We construct the coarser level operators in the following fashion, $\Sigma^{2h} = A_{\Gamma\Gamma}^{2h} - \sum_{i=1}^{k} A_{\Gamma i}^{2h} (A_{ii}^{2h})^{-1} A_{i\Gamma}^{2h}$, where the entire matrix A^{h} has first been coarsened down to A^{2h} using trilinear interpolation, and then the individual

Figure 3.3: Smoke injected from the bottom of a cylinder, and forced through a twisted bundle of cylindrical holes. Color corresponds to vorticity magnitude. Total 1.2B active cells, in a $1024^2 \times 2048$ grid.

building blocks are harvested and reassembled for computing Σ^{2h}. While this may appear a plausible choice for the multigrid hierarchy, and indeed our experiments show that this gives good convergence, the intuition behind it comes from the following observation. Suppose we constructed a multigrid hierarchy for the full problem $Ax = b$, where the right hand side b has non-zero entries *only* on the interface Γ, i.e., we are solving the

following equation via multigrid:

$$\begin{bmatrix} A_{11} & & A_{1\Gamma} \\ & A_{22} & A_{2\Gamma} \\ A_{\Gamma 1} & A_{\Gamma 2} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_\Gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ b_\Gamma \end{bmatrix} \tag{3.6}$$

then the solution can be shown to satisfy $x_\Gamma = \Sigma^{-1} b_\Gamma$. Let us further assume that our smoother routine was designed such that it completely eliminated any residual of equations interior to the subdomains, leaving nonzero residuals only on interface degrees of freedom. We can then interpret our proposed multigrid procedure, which operates solely on the interface, as algebraically equivalent to a full-domain multigrid scheme used with a right hand side that is zero anywhere outside the boundary, as in equation (3.6), combined with a smoother that annihilates residuals on subdomain interiors.

The terms $\sum_{i=1}^{k} A_{\Gamma i} A_{ii}^{-1} A_{i\Gamma}$ in the formula of Σ correspond directly to this concept of subdomain-interior equations being solved exactly, and used to eliminate those degrees of freedom from the dimensionality of Σ. A smoother that can operate on Σ directly, implicitly ensures that all equations in the interior of each subdomain are satisfied at all times. Finally, the restriction and prolongation operators for the interface-based multigrid solver can be inferred from the transfer operators of the global problem. Note that, for the purposes of this section we depart from the cell-centered perspective of grid values, as employed for example in the hierarchy construction by McAdams et al. (2010), and switch to viewing unknowns as stored in the nodes of the *dual* of the typical MAC grid used for the Navier-Stokes discretization (i.e. pressure values stored on *nodes* of this new grid). We then coarsen the cells in the typical 8-to-1 fashion, using trilinear interpolation. This ensures that interface degrees of freedom remain fully aligned across levels of the multigrid hierarchy, and that trilinear prolongation of a finer level's interface variables will only need

coarse interface variables as input. Although restriction *into* coarse interface values technically touches interior values as well, the residuals of all such interior equations will be zero (since the Schur complement operator assumes those equations fully satisfied). Thus the transfer operators we ultimately use in our cycle are *bilinear interpolation* along the aligned 2D interface surfaces at each level of the hierarchy.

Smoothing the Schur-complement system

As previously shown, Σ is a symmetric and positive definite matrix. Thus, in principle, damped Jacobi or Gauss-Seidel would have been convergent smoothers. However, since we do not have access to the explicit matrix form of Σ, operations that would be required (for example, the diagonal elements of Σ) are not readily available. Thus, we take a different approach of designing a smoother that can be iterated without an explicit construction of the matrix. Using the definition of Σ, the system $\Sigma x_\Gamma = b_\Gamma$ can be rewritten as:

$$A_{\Gamma\Gamma} x_\Gamma = b_\Gamma + \sum_{i=1}^{k} A_{\Gamma i} A_{ii}^{-1} A_{i\Gamma} x_\Gamma \tag{3.7}$$

We can use equation (3.7) to design a fixed-point iteration as follows:

$$A_{\Gamma\Gamma} x_\Gamma^{(n+1)} = b_\Gamma + \sum_{i=1}^{k} A_{\Gamma i} A_{ii}^{-1} A_{i\Gamma} x_\Gamma^{(n)} \tag{3.8}$$

One may recognize the similarity of equation (3.8) with the analogous matrix form $Dx_\Gamma^{(n+1)} = b_\Gamma + (L+U)x_\Gamma^{(n)}$ of the Jacobi iteration based on the decomposition $\Sigma = D - L - U$, should that have been explicitly available. Instead of isolating just the diagonal part of Σ, our decomposition employs the entire $A_{\Gamma\Gamma}$ term. In the next section we provide a proof that this iterative scheme will always converge.

The iterative scheme in equation (3.8) requires two basic blocks. First, given an already computed right hand side, solving for x_Γ requires solving a sparse symmetric and positive definite system. Since the matrix $A_{\Gamma\Gamma}$ is very sparse and structured, we use a sparse Cholesky factorization using the Intel MKL PARDISO library, which we have found to be very well-performing especially due to the fact that the interface is highly structured and admits a very effective nested bisection for reordering its degrees of freedom to maximize sparsity. Second, the right hand side requires the inverse operator A_{ii}^{-1} for each subdomain. Again, our approach would be to use a Cholesky factorization of A_{ii} (with appropriate reordering) to solve the inversion problem using forward/backward substitution. Pseudocode for the smoother routine is given below:

Algorithm 3.2 Application of smoother routine. Input: $b_\Gamma, x_\Gamma^{(n)}$

1: **for** $i = 1 \dots k$ **do**
2: Compute $y_i \leftarrow A_{i\Gamma} x_\Gamma^{(t)}$ {sparse; fast}
3: Solve $z_i \leftarrow A_{ii}^{-1} y_i$ {PARDISO}
4: Compute $w_\Gamma^{(i)} \leftarrow A_{\Gamma i} z_i$ {sparse; fast}
5: **end for**
6: Compute $f_\Gamma = b_\Gamma + w_\Gamma^{(1)} + \dots + w_\Gamma^{(k)}$
7: Solve $x_\Gamma^{(n+1)} \leftarrow A_{\Gamma\Gamma}^{-1} f_\Gamma$ {PARDISO}

The matrix $A_{\Gamma\Gamma}$ in step 7 has $O(N^{2/3})$ nonzero entries, and we observed that with appropriate reordering (using PARDISO) the nonzero entries in the Cholesky factors remain asymptotically well below $O(N)$. The subdomain matrices A_{ii}, however (step 3) contain on the aggregate $O(N)$ nonzero entries, which will yield a strictly superlinear number of nonzero entries in their Cholesky factors, even with excellent reordering. We thus proceed to make one last approximation, in the interest of reducing the dimensionality of these factors, as explained in the following section.

Proof of convergence of the interface smoother

Consider the fixed-point iteration

$$\begin{aligned} A_{\Gamma\Gamma}\mathbf{x}_{\Gamma}^{(n+1)} &= \mathbf{b}_{\Gamma} + \sum_{i=1}^{k} A_{\Gamma i}A_{ii}^{-1}A_{i\Gamma}\mathbf{x}_{\Gamma}^{(n)} \\ \Rightarrow S_1\mathbf{x}_{\Gamma}^{(n+1)} &= \mathbf{b}_{\Gamma} + S_2\mathbf{x}_{\Gamma}^{(n)} \end{aligned} \tag{3.9}$$

where $S_1 = A_{\Gamma\Gamma}$ and $S_2 = \sum_{i=1}^{k} A_{\Gamma i}A_{ii}^{-1}A_{i\Gamma}$. Let $\mathbf{x}_\star$ be the exact solution of this iterative scheme, i.e., $\mathbf{x}_\star$ satisfies the equation

$$S_1\mathbf{x}_\star = \mathbf{b}_{\Gamma} + S_2\mathbf{x}_\star \tag{3.10}$$

Subtracting equation (3.10) from equation (3.9) gives

$$S_1\mathbf{e}^{(n+1)} = S_2\mathbf{e}^{(n)} \Rightarrow \mathbf{e}^{(n+1)} = S_1^{-1}S_2\mathbf{e}^{(n)}$$

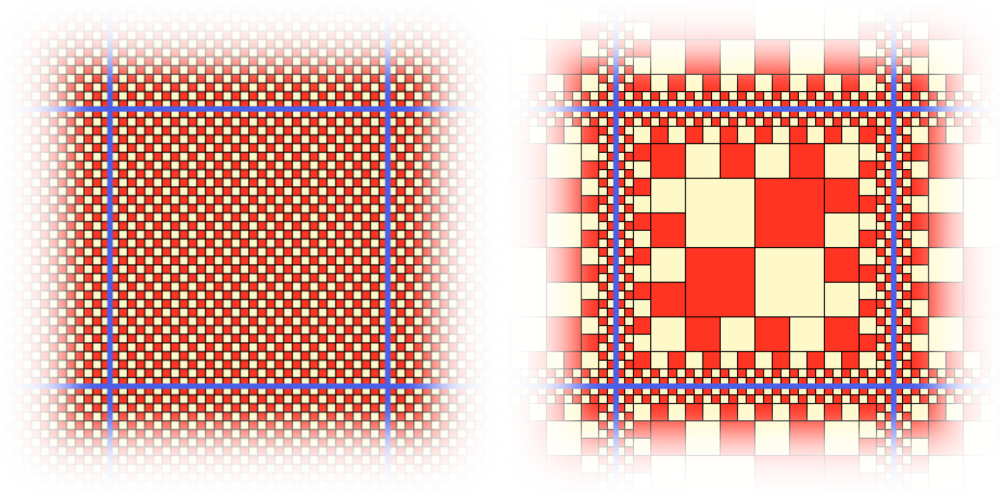
where $\mathbf{e}^{(n)} = \mathbf{x}_{\Gamma}^{(n)} - \mathbf{x}_\star$ is the error in the n-th iteration. In order to show convergence of the iterative scheme in equation (3.9), we need to show that the spectral radius of $S_1^{-1}S_2$ is less than 1. Now,

$$\Sigma = S_1 - S_2 \Rightarrow S_1 = \Sigma + S_2$$

Since S_2 is symmetric and positive definite (SPD), $S_2^{1/2}$ is well-defined. Noting that the two matrices $S_1^{-1}S_2$ and $S_2^{1/2}S_1^{-1}S_2^{1/2}$ are related by a similarity transform (via $S_2^{1/2}$), it follows that

$$\begin{aligned} \rho(S_1^{-1}S_2) &= \rho(S_2^{1/2}S_1^{-1}S_2^{1/2}) = \rho\left[(S_2^{-1/2}S_1S_2^{-1/2})^{-1}\right] \\ &= \rho\left[(S_2^{-1/2}(\Sigma + S_2)S_2^{-1/2})^{-1}\right] = \rho\left[(I + S_2^{-1/2}\Sigma S_2^{-1/2})^{-1}\right] \\ &= \frac{1}{\lambda_{\min}\left(I+S_2^{-1/2}\Sigma S_2^{-1/2}\right)} = \frac{1}{1+\lambda_{\min}\left(S_2^{-1/2}\Sigma S_2^{-1/2}\right)} \end{aligned}$$

Since Σ is SPD and $S_2^{-1/2}$ is symmetric, the matrix $S_2^{-1/2}\Sigma S_2^{-1/2}$ is SPD, so $\lambda_{\min}\left(S_2^{-1/2}\Sigma S_2^{-1/2}\right) > 0$. Thus, $\rho(S_1^{-1}S_2) < 1$.



(a) (b)

Figure 3.4: Comparison of (a) a uniform discretization of a subdomain interior and (b) our adaptive approximation in Section 3.6.

Adaptive approximation of subdomains

Another opportunity for a dimensionality-saving approximation can be exposed by analyzing the action of the operator $A_{\Gamma i}A_{ii}^{-1}A_{i\Gamma}$ on a vector x_Γ (as used in equation 3.8). This matrix-vector multiplication can be equivalently interpreted as the following process:

1. The value x_Γ is used as a Dirichlet boundary condition in a *Laplace* problem $A_{ii}\hat{x}_{ii} = A_{i\Gamma}x_\Gamma$, that computes a harmonic interpolant $\hat{x}_{ii}$ of x_Γ in the interior of Ω_i.

2. A global scalar field $\hat{x}$ is assembled by combining the values x_Γ on the interface, with the harmonic interpolants $\hat{x}_{ii}$ from each subdomain.

3. The Laplacian $y = A\hat{x}$ of this interpolated result is computed. Naturally y will be zero in the interior of each subdomain, as $\hat{x}$ was built as a harmonic interpolant in those locations. Nonzero values will occur along the interface, however. It can be shown that the restriction y_Γ

of y on the interface degrees of freedom is exactly what the Schur complement operator $y_\Gamma = \Sigma x_\Gamma$ computes. The contribution of each subdomain Ω_i to this result is exactly equal to $-A_{\Gamma i}\hat{x}_{ii}$.

Based on this interpretation, we observe that the harmonic interpolant $\hat{x}_{ii}$ in this process could be very well approximated by an *adaptive* tessellation of the subdomain interior, as shown in figure 3.4. Starting from the uniform grid spanning each subdomain (figure 3.4a), we aggressively coarsen as we transition to regions farther towards the subdomain interior (figure 3.4b). All our experiments have indicated that the quality of this approximation is excellent; remember that even if small errors might be observed in the actual interpolants, deep inside the subdomains, only the Laplacian of the resulting interpolant *on the interface* is ultimately relevant.

The performance implications of this approximation are substantial. When adaptively approximated using our aggressive coarsening in figure 3.4, the actual degrees of freedom of the (octree-type) adaptive subdomain discretization enumerate in the same order of magnitude as the interface variables in $\Gamma \cap \Omega_i$. In practical terms, this adaptive subdomain approximation translates to matrices $A_{i\Gamma}$, A_{ii}, $A_{\Gamma i}$ in Algorithm 2 being replaced by lower dimensionality, adaptive variants $A^*_{i\Gamma}$, A^*_{ii}, and $A^*_{\Gamma i}$. Matrix $A^*_{i\Gamma}$ will is simply constructed from $A_{i\Gamma}$ by removing rows that correspond to interior nodes that have been coarsened away (or have become T-junctions); all such rows would have been full of zeros in $A_{i\Gamma}$, since our coarsening scheme preserves the layer of nodes immediately adjacent to the interface at full resolution (and those are the only interior nodes touched by the stencils of interface equations). Likewise for the transposes $A_{\Gamma i}$, $A^*_{\Gamma i}$ of those matrices. Combined, all adapted interior matrices A^*_{ii} have $O(N^{2/3})$ nonzero entries, and we observed that with proper reordering their Cholesky factors remain clearly sublinear in their aggregate size, allowing us to run step 3 of Algorithm 2 (and the entire smoother) with asymptotic cost safely below the $O(N)$ mark.

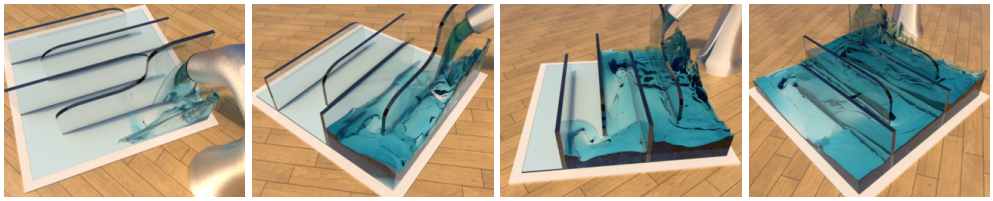


Figure 3.5: Free-surface simulation of water poured in a container with multiple interior walls causing the flow to meander around them. The frame shown in the right-most illustration corresponds to 114 million active cells, in a $1024 \times 1024 \times 1024$ background grid.

3.7 Implementation Details

Assignment of Subdomains If the domain can fit into the aggregate memory of the accelerators, the subdomains are assigned in a round robin scheme. The cost of each subdomain solve is proportional to the number of active cells within that subdomain. In the algorithm, the subdomains are shaped as perfect cubes of the same size for best adaptation saving. Given this constraint, our resulting number of subdomains are in general far more than the number of accelerators. Even though other schemes may provide better load balancing. In practice, round robin is simple and produces assignment that each accelerator can finish approximately the same time especially when the number of subdomains are significantly larger than the number of accelerators. When the simulation domain does not fit the aggregate memory of the accelerators, the subdomains have to be swapped in and out accelerators' memory. In these cases, the subdomains are assigned to the accelerator with the shortest queue. The assignment will halt until one of accelerator finishes with the subdomain at the top of the queue and freed up its memory.

Construction of adaptive operators We construct the adaptively coarsened discretization of section 3.6 based on a Galerkin process. Let us consider the example of the finest level of the multigrid hierarchy. We define an

interpolation operator P_h^* that "prolongates" the adaptive degrees of freedom x^* into their trilinearly interpolated uniform counterparts $x = P_h^* x^*$ (this interpolation is conscious of any T-junctions). Thus, the adaptive discretization of the Laplacian is simply computed as $A_h^* = (P_h^*)^T A_h P_h^*$. In fact, we never explicitly build the uniform matrix A_h, but rewrite this equation as

$$A_h^* = \sum_{a_{ij} \neq 0} a_{ij} \mathbf{p}_i \mathbf{p}_j^T$$

where $a_{ij} = [A_h]_{ij}$, and $\mathbf{p}_k^T$ denotes the k-th row of P_h^*. This formulation allows us to construct the adaptive discretization directly (by iterating over the uniform grid, and processing every spoke a_{ij} of any stencil we encounter), without ever building the uniform matrix. Since our smoother (Algorithm 2) never needs to use the *uniform* subdomain discretization, we construct the adaptive discretizations of coarser levels of the hierarchy A_h^*, A_{2h}^*, A_{4h}^*, etc. by selective (Galerkin) coarsening of the immediately finer *adaptive* discretization, rather than coarsening the corresponding uniform discretization at that level into an octree.

Avoiding nullspace issues Global nullspace components (pockets of fluid with purely Neumann boundary conditions) are handled at the top-level PCG algorithm via projection, as usual. In our smoother subroutine, we generally have the guarantee that the left-hand-side matrix $A_{\Gamma\Gamma}$ will be positive definite (it is always symmetric), if the global matrix A is definite too. However, it is possible for nullspace components to appear in the coarsened version of this matrix $A_{\Gamma\Gamma}^{2h}$, $A_{\Gamma\Gamma}^{4h}$, as a result of the Galerkin procedure, in the vicinity of Neumann domain boundaries. To avoid this, we slightly shift the eigenvalues of every coarsened discretization, say A_{2h}^* by adding a minute multiple of the identity. The shifted matrix $A_{2h}^* + \epsilon I$ is practically spectrally equivalent to the original, and fully appropriate as a substitute in a multigrid hierarchy. Since we use direct solvers to invert $A_{\Gamma\Gamma}$ in the smoother, conditioning is not an issue. Effectively, this

eigenvalue shift will penalize solution components that lie in the nullspace to be effectively equal to zero.

Boosting accuracy We use a high-order defect correction technique Trottenberg et al. (2001) to allow our approximate inverse of a first-order discretization to be used as a CG preconditioner for a higher order scheme. We structure our top-level PCG solver to implement matrix-vector multiply operations in accordance with a second order accurate discretization of the Laplace operator Enright et al. (2003). Upon invocation of the preconditioner, however, we perform the following steps: (i) We execute a few iterations of a Jacobi smoother, using the 2nd order operator, (ii) we then compute the residual, and multiply this with our first-order preconditioner, and finally (iii) we again compute the residual r, write the error equation $Ae = -r$ using the second order operator, which we solve using the same number of Jacobi iterations. We finally add the correction back to the result returned by the preconditioner. This operation, as described, preserves the symmetry and definiteness of the preconditioner, and allows the first order method to be used as an effective preconditioner for the second-order problem (at the comparably minimal expense of some additional smoothing effort near the high-order interface).

Sparse grid storage We used an ad-hoc sparse grid data structure to hold our grid data, for all our examples which utilized highly irregular, sparsely populated grids. Our data structure partitions a virtual enclosing grid into rectangular blocks (of typical size 8^3 voxels, in our examples), and stores them in a linearized array, maintaining explicit pointers to the 26 neighboring blocks to facilitate traversal during kernel invocation. This representation is reflected in both our CPU and GPU/Xeon Phi implementations of all performance-sensitive kernels.

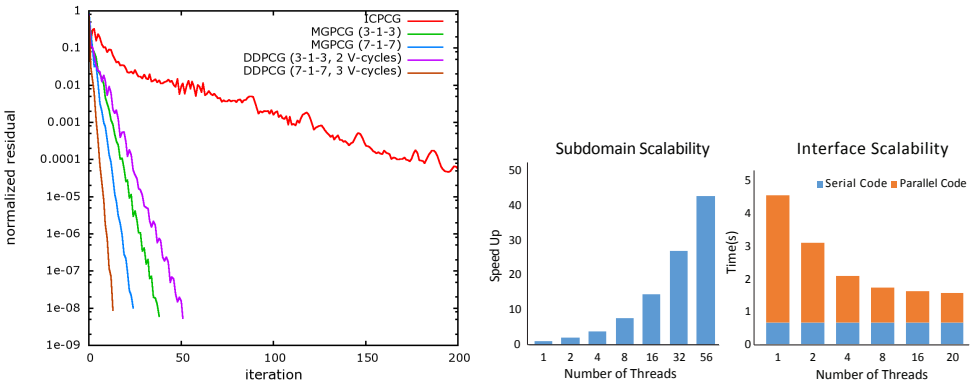


Figure 3.6: Convergence profiles for ICPCG, MGPCG (1 V-cycle, 3 boundary and 1 interior smoothing iteration), MGPCG (1 V-cycle, 7 boundary and 1 interior smoothing iteration), DDPCG (2 V-cycles per subdomain, 3 boundary and 1 interior smoothing iteration), and DDPCG (3 V-cycles per subdomain, 7 boundary and 1 interior smoothing iteration).

3.8 Application in Incompressible Free Surface Flow

We solve the incompressible Euler equations

$$\vec{u}_t + (\vec{u} \cdot \nabla)\vec{u} + \frac{\nabla p}{\rho} = \vec{f}, \qquad \nabla \cdot \vec{u} = 0$$

using the splitting scheme as described in Stam (1999). Here, $\vec{u} = (u, v, w)$ is the velocity field vector, ρ is the fluid density, p is the scalar pressure field, and $\vec{f}$ denotes external forces (such as gravity). We discretize these equations on a MAC grid, where we first explicitly update the advection terms

$$\frac{\vec{u}^\star - \vec{u}^n}{\Delta t} + (\vec{u} \cdot \nabla)\vec{u} = \vec{f}$$

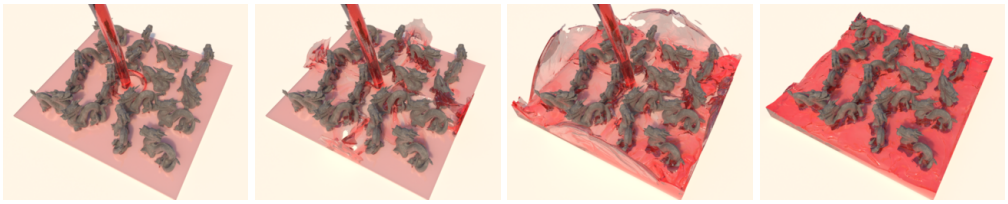


Figure 3.7: Water poured in a pool with multiple immersed objects. Second figure from the right shows 70M active cells, in a $1024^2 \times 512$ grid.

using a semi-Lagrangian scheme Selle et al. (2008) and then solve for the pressure via a Poisson equation

$$\nabla \cdot \frac{\nabla p}{\rho} = \frac{\nabla \cdot \vec{u}^{\star}}{\Delta t}$$

in order to update the intermediate velocity as follows

$$\frac{\vec{u}^{n+1} - \vec{u}^{\star}}{\Delta t} + \frac{\nabla p}{\rho} = 0$$

For tracking the free surface, we generally follow Enright et al. (2002b) using the level set advection of Enright et al. (2005), the reinitialization scheme of Losasso et al. (2005), velocity extrapolation method of Adalsteinsson and Sethian (1999), and a second order accurate pressure discretization of Enright et al. (2003).

3.9 Examples and performance benchmarks

We demonstrate the effectiveness of our preconditioner through several examples. Figure 3.1 illustrates two smoke simulations with more than one billion of active degrees of freedom, each. Figure 3.8 shows a network of interconnected vessels where smoke enters from the lower left corner and exists from the upper right corner. Our solver is able to capture the correct

incompressible behavior in relatively few iterations with four subdomains. Figure 3.7 shows an example where water is poured in a pool with multiple immersed objects, creating complex Neumann interfaces. Figure 3.5 shows an example where water flows in a channel with multiple interior walls, which cause the flow to meander around them. Figure 3.9 provides a breakdown of individual kernels of our Schur Complement solver for all these examples, along with timings for alternative solvers, detailed in the following section. We note that no vorticity confinement was used in our smoke examples. Finally, in the interest of efficiency we used as high of a CFL number as our examples could tolerate – sometimes leading to minor loss of detail.

3.10 Discussion

Evaluation of convergence and scaling In our benchmarks, we compared the convergence behavior of our Schur-Complement Domain Decomposition preconditioned CG *("DDPCG")* with a standard Incomplete Cholesky preconditioner *("ICPCG")* Foster and Fedkiw (2001), and a standard Multigrid-Preconditioned CG algorithm *("MGPCG")* McAdams et al. (2010). For the multigrid option, specifically, we note that although we did not experiment with improved CPU-based versions of MGPCG that take extra steps to better capture the topology of the domain on coarser levels of the multigrid hierarchy Ferstl et al. (2014a), we invested a significant effort to optimize the stock MGPCG to the absolute best of our capacity, both on the CPU as well as on the accelerator cards – this was a natural step to take, as the multigrid kernels used in MGPCG are re-used in the subdomain solver of our own DDPCG method. We produced two, heavily optimized MGPCG implementations: One designed to run exclusively on the CPU, and one designed to run *homogeneously* on just a single GPU, for problems that are small enough to fit entirely in GPU memory. There is only one

algorithmic difference between the two implementations: The pure-CPU MGPCG was set up to solve the coarsest level of the multigrid hierarchy using ICPCG – this was done to improve the convergence behavior at the bottom of the multigrid cycle, which was crucial in obtaining acceptable performance at our examples with more than a billion degrees of freedom (without requiring an extremely deep, and occasionally inaccurate V-cycle). The GPU-native implementation of MGPCG used a large number of smoother applications at the bottom of the V-cycle (which was effective for its smaller problem size), to avoid using Incomplete Cholesky on the GPU. We benchmarked the pure-CPU MGPCG solver on the faster (dual socket) of our two test platforms.

In all our examples, the ICPCG solver exhibited dramatically slower convergence performance than both MGPCG, and our proposed DDPCG method, often needing more than an order of magnitude of iterations higher than DDPCG to reach comparable performance. We were unable to use ICPCG for our largest of examples with billions of cells, as the footprint of the explicitly constructed matrices would cause it to run out of memory. For our smallest examples, even each iteration of our heterogeneous DDPCG actually required less time than one CPU-based ICPCG iteration. As a consequence, we did not find ICPCG to be a competitive alternative.

On the other hand, the convergence behavior of MGPCG remained competitive in several of our smaller-size examples. We should point out that the behavior of DDPCG is tunable; investing more V-cycles in the independent subdomain solves, or additional multigrid iterations in the interface solve can boost its convergence efficiency. We found MGPCG to be most competitive with our DDPCG technique in the context of our smaller examples, especially the free-surface water simulations. This is attributed to the prominence of Dirichlet boundary conditions in those scenarios, which dramatically improves the efficacy of smoothing boundary regions, which is essential for multigrid to behave favorably as

a preconditioner McAdams et al. (2010). On average, across the various frames of the water simulations (Figures 3.5,3.7) MGPCG would converge in no more than 1.5x-3x the number of iterations required by our tuned DDPCG, while in the smoke simulation of Figure 3.8, MGPCG required approximately 2x-2.5x more iterations than DDPCG. In terms of run time, however, the findings paint a quite different picture. The smaller two of our examples (Figures 3.8, 3.7) were compact enough to fit on just a single GPU card, where a single iteration of MGPCG was between 5x-8x times faster than a DDPCG iteration. Thus, in spite of the moderately slower convergence of MGPCG, its faster per-iteration cost on the homogeneous, single-GPU implementation makes it preferable to DDPCG by a factor of 3x-5x. Incidentally, the geometry of the smoke example in Figure 3.8 led to another interesting observation: Although the narrow cylindrical connectors between the glass spheres allowed for a small interface between subdomains used in our solver (and a reduction in CPU computation cost), the same geometry traits increased the approximation error induced by our adaptive coarsening of the subdomain interiors, increasing the required iterations for PCG convergence.

The situation is dramatically different for our larger simulation examples, which cannot be solved with MGPCG on a single accelerator card. For those examples, the only practical alternative was to run MGPCG homogeneously on the CPU. In this context, we observed that each iteration of our DDPCG method was within 20% of the cost of a CPU-only MGPCG iteration. However, for the large-scale examples, dominated by Neumann boundary conditions, we observed MGPCG requiring up to 5x more iterations for convergence, leading to a 3.5x-4.5x performance benefit of DDPCG versus the CPU-only MGPCG. We conclude that for small problem sizes, in the order of 100M degrees of freedom or less, a *homogeneous GPU* implementation of MGPCG is the preferred solver, provided that the problem can fully fit in GPU memory. For problem sizes that do not fit

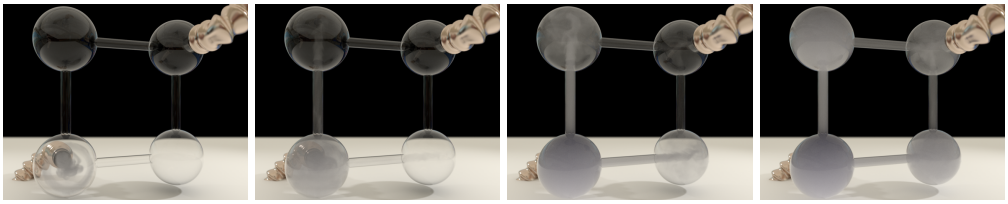


Figure 3.8: Smoke flow in a network of interconnected vessels simulated using a $1024^2 \times 512$ background grid and 42 million active cells. The computational domain was divided into four subdomains. The proper flux is observed both in the inlet and the outlet of the flow.

completely in GPU memory, DDPCG appears to be consistently superior to CPU-based MGPCG, with the performance gap becoming larger as the resolution increases.

Limitations and future work The most fundamental limitation of our proposed method is that, in order for its scaling benefits to take effect, it needs to be applied to a problem of adequately large size. In designing our (GPU-hosted) multigrid cycle for the solution of the subdomain problems, we made a conscious choice to keep the design of this solver as simple as possible, approximating the domain at voxel-accuracy at every level, and not enacting any remedies for topological inconsistencies, such as regions merging or small Neumann gaps disappearing after coarsening. This was done in the interest of simplicity, to facilitate low-level optimizations of the solver components. It is quite likely that topology-conscious coarsening schemes Ferstl et al. (2014a) could further improve the convergence properties of this component, and the balance between enacting such improvements and retaining opportunities for aggressive optimization certainly merits investigation. Finally, one should not discount the software engineering challenges that are still associated with developing numerical software that is as inherently heterogeneous as our solver. The established programming paradigms that are available for *homogeneous* thread-based

parallel development (e.g. OpenMP) are arguably much more accessible to the non-expert developer. Given the precedent of CUDA, and the growing presence of heterogeneity in modern systems, we hope that programming abstractions for these platforms will continue to evolve.

The scope of our work was specifically restricted to the design and optimization of the pressure Poisson solver on a heterogeneous computer. We also specifically targeted fluid simulation on *uniform* grids in the development of our solver. Although extending the concepts of our solver to an adaptive discretization is certainly possible from an algebraic perspective, we feel that a careful investigation is warranted to make sure that the complexity of work that needs to happen on the interface region remains comparatively lower. This aspect, as well as practices for efficient dynamic partitioning of temporally changing adaptive grids would be an exciting topic for continued investigation.

A very interesting venue for future work might focus on extending our technique to deeper hierarchies of heterogeneous platforms, using for example clusters of network-interconnected GPU-accelerated nodes. The key challenge of using this algorithm directly on a cluster is that the current interface solver is not designed as a salable operation. With more computation units inside a cluster, more subdomains will need to be created. This inevitably creates a larger interface problem as well. As the interface problem is solved on a single CPU, it will became the bottleneck of the whole process. There are two ways to mitigate this issue. The first is to redesign the interface solver to be scalable. The schur-compliment operator requires solving the adapted subdomain operators. They can be solved in parallel and in isolation. The second way to make the algorithm more scalable, is to using a hierarchy of subdomains. The same way that we used a multigrid cycle to approximate a subdomain solver, one could envision using our entire preconditioner as the approximate solver for a subdomain assigned to each cluster node, which is internally subdivided

	Water (Fig. 3)		Water (Fig. 9)		Smoke (Fig. 4)		Smoke (Fig. 1; right)	Smoke (Fig. 1; left)
	Active DOFs: 114M		Active DOFs: 70M		Active DOFs: 42M		1.2G DOFs	1.8G DOFs
	GPU	Phi	GPU	Phi	GPU	Phi	GPU	Phi
Number of subdomains	16		16		4		16	40
Subdomain resolution	128 x 128 x 768		128 x 128 x 768		512 x 512 x 512		512 x 512 x 512	1536 x 1536 x 1920
DDPCG total solve time	51.894	44.532	61.446	42.882	38.472	25.704	1790.11	4247.98
DDPCG iteration cost	5.766	4.948	4.389	3.063	1.603	1.071	100.49	193.09
Preconditioner Application	4.291	4.314	3.657	2.769	0.999	0.843	88.9341	183.9
Subdomain Solve (3 V-cycles)	1.362	1.249	1.046	0.865	0.457	0.351	26.4284	152.79
Each MG V-cycle	0.378	0.311	0.311	0.218	0.115	0.079	2.68	21.83
Data transfer from/to CPU	0.158	0.309	0.084	0.169	0.103	0.105	7.6252	16.002
Interface Solve (1 V-cycle)	1.392	1.539	1.214	1.096	0.029	0.032	33.042	14.3918
Smoothing (Top-Level)	0.413	0.349	0.314	0.215	0.006	0.003	7.864	3.741
Restriction/Prolongation	0.008	0.012	0.008	0.007	0.00013	0.0002	0.145	0.094
ICPCG iteration cost (CPU only)		4.16		2.82		1.32	N/A	N/A
MGPCG iteration cost (GPU only)	0.424		0.316		0.1736		N/A	N/A
MGPCG solve time (GPU only)	10.1838		17.679		7.983		N/A	N/A
MGPCG iteration cost (CPU only)		5.55		3.135		4.489	67.28	175.4
MGPCG solve time (CPU only)		127.65		153.5933		218.64	3902.8062	16742

Figure 3.9: Timing information for four examples. **All run times cited are in seconds**. Our "GPU" platform is an Intel Xeon E5-1650v3 CPU equipped with two NVidia GTX Titan X GPUs and 128GB RAM, while our "Phi" platform is an Intel Xeon E5-2650v3 CPU equipped with six Intel Xeon Phi 31S1P cards.

to use GPU accelerations as we currently do. Although bandwidths of network interconnects would be even slower than that of PCIe, due to economy of scale the relevant asymptotics (relative size of interfaces vs. entire grid) could remain favorable. Finally, emerging GPU architectures and technologies (stacked memory, integration of CPU and GPU) might facilitate programming in a homogeneous model (using capabilities such as unified memory spaces), but non-homogeneity in memory bandwidth is almost certain to persist in some form (cores having significantly higher bandwidth to their "local" region of memory). We feel that the adaptation of solver concepts to such architectural traits is an exciting research thread.

4 NARROW-BAND TOPOLOGY OPTIMIZATION ON A SPARSELY POPULATED GRID

In the context of designing efficient solvers, topology optimization as an application creates very interesting challenges to the numerical solvers. It desires high resolution, for many interesting details only emerge in high resolution simulation. It generates sparse domain, as in many scenarios only a fraction($< 5\%$) of domain is filled in the final solution. It also creates high contrast ($1 : 10^{-9}$) material distributions.

In this chapter, I present a design of an efficient linear elasticity solver that is tailored for high resolution and sparse domain. Regarding the high contrast material distribution, I will demonstrate how it impairs the multi-linear multigrid solver's convergence rate, which inspired a design of stencil-aware interpolation scheme that drastically improves convergence rate in those situations.

4.1 Related Work

Uniform and adaptive grids Among the various kinds of data structures that have been developed by researchers in computational science and computer graphics, uniform grids play a central role in the vast majority of topology optimization applications for their multiple advantages in computation. These advantages include cache-coherent memory access, regular subdivisions for parallelization, simple data layout, and, in particular, the existence of efficient numerical PDE solvers. A multigrid FEM solver discretized on a uniform grid has been established as one of the standard solutions for elastic solids Zhu et al. (2010b), character skinning McAdams et al. (2011), and topology optimization (e.g. Sigmund and Torquato (1999)). One of the main challenges of uniform grids lies in their lack of adaptivity. Due to their uniformly distributed and

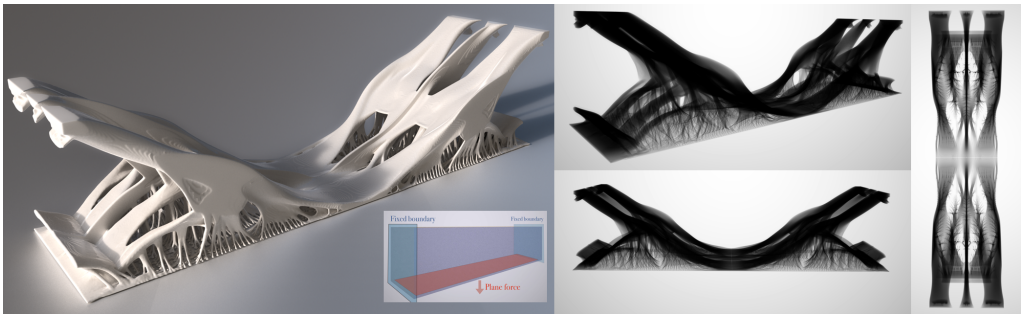


Figure 4.2: An optimized bridge structure is obtained on a $271 \times 540 \times 1081$ grid (157 M voxels) with mirror boundary conditions applied on the X and Z axes. The left figure shows a solid render, and the right three figures show the visualization of the density field from different camera views. The middle sub-image illustrates the boundary conditions, including fixed boundaries on the sides and a load applied on the bottom plane.

axis-aligned grid lines, it is difficult to dynamically allocate fine grid cells around the regions of interest while still preserving all the computational advantages. To overcome this limitation, researchers have been devoting efforts to create multiple levels of resolutions by dynamically refining grid cells, which lead to the invention of a variety of hierarchical data structures, e.g., the octree grid Losasso et al. (2004) and the AMR grid Donea et al. (1982). These adaptive data structures have been integrated with high-performance solvers for computing large-scale systems. For example, Ferstl et al. (2014b) combined an octree grid with a multigrid-preconditioned conjugate gradients (MGPCG) solver for Poisson problems in large-scale liquid simulation. In addition, new dynamic structures have been proposed to obtain adaptivity without breaking the topology of a uniform grid, e.g., by overlapping grids with different resolutions English et al. (2013b) or anisotropically stretching grid cellsZhu et al. (2013).

Lagrangian approaches An unstructured Lagrangian mesh inherently exhibits adaptivity. It is natural to allocate degrees of freedom on a La-

grangian mesh adaptively, according to some local meshing criteria, e.g., the distance to structures of interest. Further, Lagrangian approaches are capable of tracking the structure interface by explicitly maintaining the boundary mesh. These two main advantages allow Lagrangian meshes to be widely used in topology optimization (see Eschenauer et al. (1994), Sokolowski and Zochowski (1999), Christiansen et al. (2014), Christiansen et al. (2015) for examples). The main limitation of Lagrangian approaches lies in their inefficiency in solving large-scale linear systems. The unstructured nature of a Lagrangian mesh makes it difficult to build an efficient preconditioner for iterative solvers.

Sparse representations In contrast to uniform grids, sparse data structures reduce the computational cost of a volumetric discretization by maintaining active elements selectively. The key idea for sparse data structures is to establish a mapping from a grid cell index in the real, sparse space to an index in the virtual, compact storage, enabling the allocation of computational resources only to grid cells occupied by or near the real structures. This mapping can be implemented by a standard hash table (e.g., local level set Brun et al. (2012)), an octree (e.g., adaptive distance field Frisken et al. (2000), OpenVDB Museth (2013)), or via a Virtual Memory Page Table and the Translation Lookaside Buffer (TLB) (e.g., SPGrid Setaluri et al. (2014)) for fast access to grid cells. In addition to using a single type of discretization, researchers have also invented a variety of hybrid data structures to model thin phenomena embedded in a high-dimensional space. One example is the particle level set Enright et al. (2002a), which maintains a narrow band of particles around an implicit interface discretized on a background grid. The coupling of the two representations enables an accurate computation for the signed distance function. Similar concepts can be seen in the narrow-band FLIP method Ferstl et al. (2016) and the hybrid grid-mesh approach Zheng et al. (2015), where a thin layer

of Lagrangian elements are hybridized with a Eulerian discretization to efficiently capture the features around the interface.

Hardware acceleration At the heart of a high-resolution topology optimization algorithm is a highly efficient numerical solver for FEM discretized linear elasticity. Hardware acceleration is essential to boost the performance of these solvers. GPU-based approaches have been widely used in speeding up topology optimization algorithms. For example, Wu et al. (2016b) have proposed a high-performance multigrid FEM solver with deep integration of GPU hardware to achieve good convergence, low memory consumption, and reduced bandwidth requirements. More related work on GPU-based topology optimization can be seen in Wadbro and Berggren (2009) Schmidt and Schulz (2011), Challis et al. (2014), and Yadav and Suresh (2014). The power of supercomputing has also been explored. In Aage et al. (2015), a parallel topology optimization framework was proposed using the Portable and Extendable Toolkit for Scientific Computing (PETSc). A variety of mechanical designs with intricate thin features was obtained by running

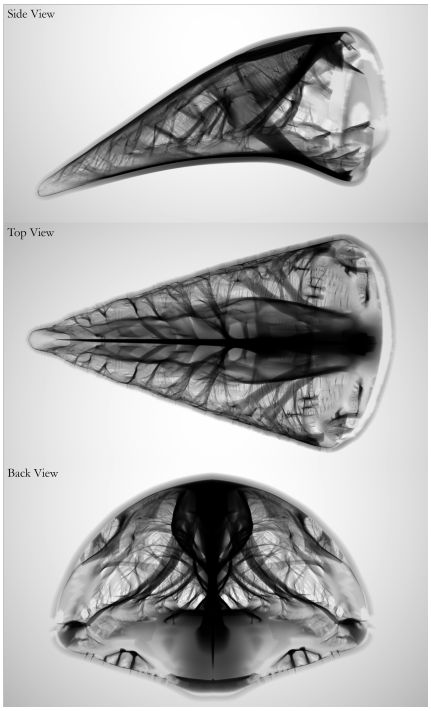


Figure 4.1: The interior structures supporting the shell of a bird beak generated using our narrow-band topology optimization algorithm with $1,040,875,347$ (1.04 billion) active FEM voxels. The resolution of the background grid is $3000 \times 2400 \times 1600$. The three figures show the volumetric rendering of the same structure from different views.

high-resolution simulations (with up to 83 million elements) on a supercomputer. This computational framework was further used in Aage et al. (2017) to explore the optimal design of airplane wings at the level of one billion voxels. Besides GPU and supercomputer, multi-accelerator heterogeneous computing has also been explored. For example, a scalable parallel solver on a workstation fitted with GPUs or many-core accelerators has been proposed in Liu et al. (2016) to solve systems in the order of a billion degrees of freedom. The proposed Schur-complement numerical technique allows to benefit from the high memory and compute bandwidth of GPUs for large-scale problems.

4.2 Topology Optimization Overview

Topology optimization aims to solve the problem of distributing materials in a given domain that minimizing the structural compliance or other objectives to given load(s). It has demonstrated its efficacy in creating mechanical designs with complex structures and extreme properties in various engineering problems (see Rozvany (2009), Sigmund and Maute (2013), Deaton and Grandhi (2014) for surveys). Starting from a volumetric domain that is uniformly filled with material, a standard topology optimization algorithm iteratively removes and redistributes material to develop a structure that minimizes a design objective (e.g., structural compliance), given the prescribed target volume and boundary conditions.

However, due to the limitations of the previous computational frameworks, it is challenging to perform a standard topology optimization algorithm to emerge and evolve these thin and sparse features. For example, to model the evolution of a thin sheet at the length scale of ten micrometers within a centimeter cube, it requires an FEM solver discretized on a 1000^3 Cartesian grid. This grid resolution amounts to the order of magnitude of one billion active elements.

Recent advances in supercomputing provide some solutions that overcome this challenge. For example, Aage et al. (2017) ran a parallel topology optimization program on a cluster with 8000 cores for days and obtained the structure of an airplane wing with the computational domain filled with 1.1 billion voxels. A variety of novel, intricate, and multi-scale structures naturally emerge from the super-resolution computations. This result is impressive, yet the limited accessibility and high cost of the supercomputing resources impede the use of such numerical approaches in a broader range of research and engineering applications. New computational tools, which are easy to access, efficient to run, and able to solve super-resolution systems, are needed in the scientific community.

4.3 Main Contributions

We propose a new computation framework that combines a sparse representation in conjunction with a highly optimized elastic multigrid solver for topology optimization, which delivers a significant leap in solving super-resolution problems from the prior state-of-the-art results. Our framework has enabled the simulation of a sparsely populated computational domain on the level of billion active grid voxels on a single workstation. By examining the simulation results at such scale, we identify and analyze new challenges that have not been addressed, or even observed, in the previous research on elastic deformation simulation and topology optimization, such as poor asymptotic convergence of standard Galerkin-coarsened multigrid algorithm and the algorithmic limits of floating-point precision.

To address these scale-dependent challenges, we combined design practices and algorithmic interventions on data structures, numerical methods, and parallel solver implementations. First, our framework is centered around a sparsely populated grid data structure, which allows

the dynamic allocation of the degrees of freedom within a narrow band around the structure. This data structure combines the benefits of sparse storage, implicit topology representation, and the performance potential of high-throughput stream processing. We have observed and demonstrated numerically that the elements with high structural sensitivities, which are essential for developing the structure towards its optimal topology, tend to bundle within a narrow band around the structure during evolution. In contrast, the vast bulk of void regions far from the structure contribute little to the total structural compliance, but they are the primary source of the computational cost in a dense discretization. Therefore, the computation can be dramatically reduced by using a dynamically allocated sparse discretization, i.e the narrow-band representation.

In addition to the narrow-band representation, we developed a novel mixed-precision multigrid solver that is capable of solving FEM discretized linear elasticity in the order of billion elements on a single workstation. By proposing an SPGrid-optimized matrix-free formulation for data storage and a novel mixed-precision computation, the solver can meet the memory storage and bandwidth demands of a multiprocessor workstation while maintaining high accuracy. Also, our vectorized multigrid solver takes advantage of the AVX512 instruction set on the Intel Skylake-X/SP architecture in order to further enhance performance.

We summarize the main contributions of our work as follows:

- We demonstrate an ensemble of data structures, numerical schemes, and implementation best practices to perform topology optimization with the highest resolution (over one billion voxels) on a single shared-memory multiprocessor.

- We propose a sparse, adaptive topology optimization framework where simulation of elastic deformation is restricted to a narrow band surrounding the high-density region.

- We demonstrate the capacity of our framework to obtain a variety of complex thin and codimensional features, such as thin films and beams, that previous approaches might suppress.

4.4 Method Overview

Our sparse topology optimization framework consists of three key components: a sparse grid structure, a high-resolution multigrid FEM solver, and a narrow-band and unbounded structure optimizer. First, we briefly introduce the sparse paged structure (SPGrid) Setaluri et al. (2014) as the base data structure to track the optimizing structures (Section 4.5). Next, we discuss our multigrid FEM solver for computing large-scale elastic systems on SPGrid in Section 4.6. Finally, in Section 4.7, we provide method validations with respect to both the multigrid solver.

4.5 Sparsely Populated Grid Structure

The Sparsely Populated Grid (SPGrid) data structure Setaluri et al. (2014), in comparison to other sparse data structures, leverages the virtual memory system to allocate a very large virtual memory address span, corresponding to a sparsely populated background grid, while only materializing in physical memory the parts of this grid that are active. The allocation unit in SPGrid is a *block*, a rectangular region of the Cartesian grid that is made contiguous in memory address space by virtue of a space-filling traversal scheme; The size of SPGrid blocks is chosen to be a multiple of a 4KB, i.e. the size of a physical memory page. SPGrid stores a number of data channels, which have similar sparsity pattern and corresponding indexing, in the same allocation block. Using the SPGrid data structure enables us to compute effectively on a sparsely populated domain with effective bandwidth comparable to a cache-optimized, dense uniform grid.

In our multigrid FEM solver, we utilized the flexibility of SPGrid's block size, and chose a block size of $4 \times 4 \times 8$, tailored for vectorization as described in Section 4.6.

4.6 Multigrid Solver

Our topology optimization framework utilizes a numerical solver for a lattice-based Finite Element Method (FEM) discretization of linear elasticity, with spatially varying material parameters. In order to accommodate the large resolutions targeted by our method, we employ a solver based on the Multigrid-Preconditioned Conjugate Gradients (MGPCG) algorithm. Algebraically, our methodology is similar to prior formulations of multigrid-preconditioned solvers [Wu et al. (2016c); Dick et al. (2011a)], in employing a hexahedral discretization of the linear elasticity operator, leveraging Galerkin coarsening to generate coarse level operators, and using a symmetric V-cycle as a preconditioner for Conjugate Gradients. We deviate from standard practices as documented in prior work by way of design choices, data structures, and parallelization practices including:

- A matrix-free implementation of the finest-level elasticity operator tailored for the SPGrid sparse storage structure.
- A bandwidth-saving construction of the Galerkin coarsened operator at each level, which avoids streaming through explicitly-built matrices as input any only relies on material properties at the finest level.
- Storage of the explicitly formed coarse grid operators in a novel, SPGrid-specific banded sparse matrix format.
- A modified eight-color Gauss-Seidel smoother designed to optimize memory bandwidth utilization and SIMD efficiency.

- A mixed-precision implementation of MGPCG, which combines the accuracy of double-precision arithmetic with the storage saving of single-precision representations.

Matrix-free design of finest level operator. Due to the size of the simulation domains we target, economy in memory footprint is essential to our approach. An explicit matrix storage at any level of the discretization would necessitate 243 scalar coefficients per lattice node, consisting of 27 spokes of 3×3 matrices. This number excludes any compaction afforded by storing only the symmetric half of the operator, but also does not account for any additional storage of matrix indices (e.g. in compressed sparse row format), or explicit topology storage of the computational domain (e.g. explicit hexahedral mesh).

The SPGrid data structure provides the abstraction of a sparsely populated grid, without the need to explicitly encode the connectivity of hexahedral simulation elements (e.g. as in an explicit mesh structure), as it is implied by the background uniform grid topology. Although our computational domain is embedded in a large background regular grid (up to the resolution of $3000 \times 2400 \times 1600$), its active cells in our simulation are only a sparse subset (up to 1.04 billion active cells, in our tests). The SPGrid structure allows us to directly store these active cells, their material parameters as well as their nodal forces and displacements in a sparsely populated grid.

At the finest level, we implemented a numerical kernel that computes the elastic forces resulting from the elasticity operator, for all nodes of a given $4 \times 4 \times 8$ block of the SPGrid structure; this kernel is designed to be free of write-dependencies, hence all blocks can be processed in parallel on different threads.

Consider a single grid cell with nodal displacements $\{\mathbf{u}_i\}_{i=1}^{8}$. We denote by $\{\mathbf{f}_i\}_{i=1}^{8}$ the nodal forces produced by the elastic response of the same cell,

which can be expressed as $\mathbf{f}_i = \sum_{j=1}^{8} \mathbf{K}_{ij} \cdot \mathbf{u}_j$. Each $\mathbf{K}_{ij}$ in this expression is a 3×3 matrix; we store these coefficients in a $8 \times 8 \times 3 \times 3$ tensor $\mathcal{K}$, whose elements are given by $\mathcal{K}_{ijvw} = [\mathbf{K}_{ij}]_{vw}$. This tensor is given as a linear combination of two canonical tensors $\mathcal{K}^\mu$ and $\mathcal{K}^\lambda$ based on the Lamé coefficients, corresponding to the stiffness matrices computed for $(\mu, \lambda) = (1, 0)$ and $(\mu, \lambda) = (0, 1)$ respectively. They can be computed either by analytic integration of the linear elastic trilinear element, or an eight-point Gauss quadrature rule. Ultimately, the elemental stiffness tensor is expressed as $\mathcal{K} = \mu\mathcal{K}^\mu + \lambda\mathcal{K}^\lambda$. We use the notation $\mathcal{C}(i)$ for the set of eight cells incident to node i, while $\mathcal{V}(c)$ denotes the eight vertices at the corners of cell c. We can then express the total force on node i as:

$$\mathbf{f}_i = \sum_{c \in \mathcal{C}(i)} \sum_{j \in \mathcal{V}(c)} (\mu^{(c)} \cdot \mathbf{K}^\mu + \lambda^{(c)} \cdot \mathbf{K}^\lambda)_{i^{(c)} j^{(c)}} \cdot \mathbf{u}_j$$

where $\mu^{(c)}, \lambda^{(c)}$ are the parameters of cell c, and $1 \leqslant i^{(c)}, j^{(c)} \leqslant 8$ are the local indices of nodes i, j as vertices of cell c. This operation can be trivially vectorized; let $\underline{\mathbf{f}} := (f_{(p,q,r)}, f_{(p,q,r+1)}, \ldots, f_{(p,q,r+7)})$ be a SIMD vector of eight forces on nodes that are sequential along the z-axis, while $\underline{\mu}^{(c)}, \underline{\lambda}^{(c)}$ are the parameters of a cell neighbor for each of the eight sequential grid indices, and finally, $\underline{\mathbf{u}}_j$ the set of eight sequential nodal neighbors at a specific offset direction. The previous operation is then expressed as:

$$\underline{\mathbf{f}} = \sum_{c \in \mathcal{C}(i)} \sum_{j \in \mathcal{V}(c)} (\underline{\mu}^{(c)} \cdot \mathbf{K}^\mu + \underline{\lambda}^{(c)} \cdot \mathbf{K}^\lambda)_{i^{(c)} j^{(c)}} \cdot \underline{\mathbf{u}}_j. \tag{4.1}$$

We note that a SIMD line need not be structured strictly along a sequence of nodes aligned along the z-axis (or any other single axis); a rectangular arrangement, e.g. eight nodes straddling a 2×4 rectangle along the y- and z-axes would work in exactly the same fashion. From a programming standpoint, equation (4.1) indicates that SIMD vectors of each Lamé parameter are scaled with a constant coefficient (a single multiply instruction,

with embedded broadcast, in the AVX2 and AVX512 instruction sets), and multiply the respective neighboring displacements $\underline{\mathbf{u}}_j$ (a fused multiply-add operation) to compute a term contributing to the nodal force $\underline{\mathbf{f}}$. With the increased number (32) of available registers in AVX512, we have verified that the entire stencil application can be executed with 2×24^2 multiply and 2×24^2 fused multiply-add instructions (FMA) without any register spilling, and enough distance between operation dependencies to allow the full throughput of two FMA instructions per cycle in Skylake-X/SP[1] (approximately 1200 cycles for the stencil application on one 16-wide SIMD line).

Modifications to the SPGrid structure In order to accommodate the SIMD-heavy stencil computations in our elasticity operator, we enacted two crucial modifications/enhancements of the baseline implementation (http://www.cs.wisc.edu/~sifakis/SPGrid.html) of the SPGrid structure of Setaluri et al. (2014). The first of those, is a vectorized load routine, with a compile-time stencil offset, as in:

```
   template <int di,int dj,int dk,class SPG_array>
__m512 VectorGet<di,dj,dk>(SPG_Array a,int64_t offset)
```

where a 16-entry SIMD width in single-precision has been used, as an AVX512 example. The offset given as argument is presumed aligned at SIMD-width granularity, while a stencil offset `(di,dj,dk)` is given as a compile-time argument. For this specific application `di,dj,dk` only takes value of `(-1,0,1)`. Using these semantics, grid data at a given stencil "spoke" (from an aligned baseline vector address) can be loaded for an entire SIMD line, as illustrated in Figure 4.3. Even though the stencil shift might cause data to originate from different SPGrid blocks, the fact that

[1] Intel Xeon Gold 6140 processor (18 cores at 2.30 GHz) with 192 GB memory.

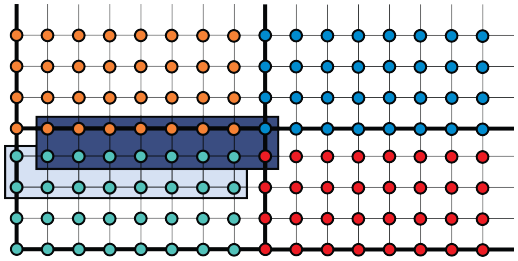


Figure 4.3: The `offset` passed to `VectorGet` points to an sequence of SPGrid entries with SIMD-width alignment (light blue box). Using a stencil offset, e.g., shifts the region to be loaded by `VectorGet`, as shown in the dark blue box. The data of this shifted box may originate in several distinct SPGrid blocks (indicated by different node colors).

such shift is known at compile time allows significant optimizations that avoid expensive gather operations and minimize address translations.

The second SPGrid modification was the relaxation of the design restriction in Setaluri et al. (2014) that each SPGrid block be sized to exactly 4KB (the size of a virtual memory page). Our solver used a total of 128 bytes for all variables stored at each grid index, leaving the block size to just $2 \times 4 \times 4$ when a 4KB block size is used. We found it more effective to be able to use a larger block size, namely $4 \times 4 \times 8$ in order to (a) minimize the number of SIMD stencil accesses that straddle multiple blocks, and simplify implementation of `VectorGet`, and (b) allow an adequate number of nodes to be present per-block, to ensure that even after eight-coloring (as required by our smoother, described later in this section), the nodes on each color are a multiple of the SIMD width, even on AVX512 systems. We include source code for both proposed SPGrid modifications, as a supplement to our paper. As an indication of performance, we have achieved an effective bandwidth of 17.45 GB/s for our multiply kernel.

Efficient Galerkin Coarsening. In the construction of the multigrid hierarchy, we used the Galerkin coarsening method, computing the coarse

grid operator as $\mathbf{K}^c = \mathbf{P}^T \cdot \mathbf{K}^f \cdot \mathbf{P}$, where $\mathbf{P}$ is the prolongation matrix and $\mathbf{K}^f$ the fine grid operator. At all but the two finest levels, we can afford to store the coarsened operator explicitly, as the reduced dimensionality of the coarse grid allows us to do so at one-eighth of the memory footprint that such matrix would have occupied at the finer level. Our construction of $\mathbf{K}^c$ is tailored around the following implementation objectives:

- Neither $\mathbf{K}^f$ or $\mathbf{P}$ are presumed available in matrix form.
- The rows of $\mathbf{K}^c$ should be computed independently, to avoid write hazards. †
- We seek the flexibility to compute $\mathbf{K}^c$ at *any* coarse level directly from the material parameters at the finest level, without depending on operators at intermediate levels.

Let us consider the specific example of constructing the operator $\mathbf{K}^{4h}$ at two levels coarser from the finest grid:

$$\mathbf{K}^{4h} = \mathbf{P}^T_{4h\to 2h}\mathbf{P}^T_{2h\to h}\mathbf{K}^h\mathbf{P}_{2h\to h}\mathbf{P}_{4h\to 2h} \quad (4.2)$$

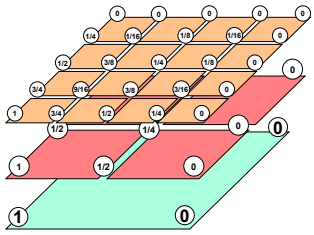


Figure 4.4: Two successive prolongation operations on a Kronecker delta function during Galerkin coarsening of a coarse cell, illustrated in 2D.

The coefficients of the i-th row of this matrix (or equivalently, the i-th column, due to symmetry) are given by the action $\mathbf{K}^{4h}\mathbf{e}_i$ of this operator on the basis vector $\mathbf{e}_i$. Equation (4.2) suggests that this action can be computed by successively prolongating $\mathbf{e}_i$ to the finest level, applying the fine-grid operator $\mathbf{K}^h$, and restricting the result back to the coarse grid. We perform this operation separately on each of the eight cells (at level $4h$) incident on node i, as illustrated in Figure 4.4. The input to this process is a discrete Kronecker delta, shown as the input coefficients to the coarsest level. We can use an eight-wide SIMD register to store all

eight nodal values of this coarse cell. We have implemented a routine ProlongateCell() that interpolates this eight-value SIMD register into eight more eight-wide registers corresponding to the nodal values of the child cells at the immediate finer level. This routine is called recursively to prolongate all the way to the finest level. At that point, a routine CellMultiply() is used to compute the force response of each individual fine cell to these prolongated nodal displacements (using the material properties at the finest level). A routine RestrictCell() implements the adjoint of the prolongation operation by collecting force contributions from fine child cells to their coarser parent. Since these routines are called recursively, all SIMD vectors are stack-allocated and can be effectively cached. As an indicator of performance, the construction of the entire operator hierarchy in our 1.04 billion-voxel example (Figure 4.1) requires 113.9 seconds using AVX512 instructions, which is a very small fraction of the MGPCG cost at this resolution.

Sparse Matrix Storage The storage of the Galerkin-coarsened matrix needs to be handled as to exploit sparsity, facilitate the application of the smoother routine, and allow a direct solver (in our case, Intel MKL PARDISO) to be used for solving the problem at the coarsest level of the hierarchy. Given that the topology of the background is a regular Cartesian lattice, we use a band-storage approach, where the 243 nonzero coefficients associated with the stencil each node (a 3×3 matrix for every spoke of a 27-connected $3 \times 3 \times 3$ stencil) are stored into a secondary SPGrid structure. We supplement these 243 scalars with a bit field, indicating whether each stencil spoke is structurally present in our discretization. This representation allows straightforward implementation of the smoother routine, and can be easily converted to compressed sparse row (CSR) format for usage in direct solvers like PARDISO. In this conversion, the only serial operation is the calculation of linearized indices, and the necessary allocated length

of each compressed row; the data transfer into the CSR coefficient buffer is performed in parallel over SPGrid blocks.

Optimization of the relaxation routine In order to balance convergence efficiency with parallelization potential, we employ an eight-color Gauss-Seidel (GS) routine, as other authors have similarly adopted in prior work Wu et al. (2016c), with the slight modification that we collectively update all three collocated degrees of freedom at each grid node, by inverting the 3×3 diagonal block of the stiffness matrix corresponding to that node. A drawback of combining the eight-color GS smoother with a SIMD implementation is the suboptimal utilization of memory bandwidth, as each SIMD vector will require data that is consistently discontinuous in memory (Figure 4.5, left). Such scattered memory assess is particularly wasteful for modern hardware which always performs load operation from memory at cache-line granularity. In order to circumvent the need for such scattered data access, we preemptively transpose the data in each SPGrid block as to reorder the indices of each color to be consecutive in memory (Figure 4.5, right/bottom). We observe that applying the same stencil offset to the nodes of one color always leads to nodes of a different yet consistent color (e.g., in Figure 4.5, applying a (+1,+1) offset to a yellow node always leads to a red node). As a consequence, after the described transposition, applying the colored GS smoother on each individual color can be performed by a straightforward application of the `VectorGet` routine to the transposed data. Each SPGrid block is transposed back to the original ordering at the end of the application of the relaxation routine. The reordering operation is a streaming memory operation and is required only twice (one transpose into the colored layout and one transpose back) for each sequence of smoother application. The transposition cost is negligible in comparison to the smoother. We have observed an effective bandwidth of up to 68 GB/s (out of maximum 128 GB/s) on a Skylake-SP platform for

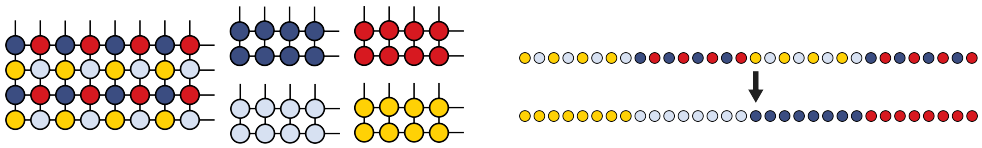


Figure 4.5: Left: a SPGrid block of 4x8 in which the nodes are ordered lexicographically, Right: the transposed colored storage enables efficient vectorization for Gauss-Seidel iteration. Bottom: Transposition operation of a SPGrid block illustrated in a linear memory layout.

the eight-color GS routine. This is computed by assuming perfect caching behaviour and all data is read from/write to the main memory exactly once.

A mixed-precision MGPCG solver The linear systems arising from the equations of elasticity in our large-scale topology optimization tasks impose a unique set of challenges to the numerical algorithms used. Due to both the sheer size of the computational domains we seek to accommodate, and the large contrast of material stiffness values used in different regions of the simulated domain, we often encounter situations where an MGPCG solver using single-precision (32-bit) floating-point arithmetic cannot sustain satisfactory convergence, or even instances where the solver will plainly diverge. There are also scenarios where single precision will have catastrophic consequences on our solver, when Galerkin-coarsened matrices will be reported as effectively "singular" by direct solvers (i.e. MKL PARDISO) if constructed to single precision. We note that the frequency of incidence of such issues was dramatically increased, in our experience, when dealing with domains in excess of 10^8 voxels, while lower-resolution problems would be significantly more resilient.

An MGPCG solver implemented natively in double precision was fully effective for all the examples in our paper. However, using double precision would double our memory footprint, which was a significant

concession given our pursuit of exceptionally high-resolution domains. We thus designed a variant of such solver that used a carefully crafted mix of single- and double-precision arithmetic, which we have found to produce results of effectively identical accuracy as a native double-precision solver. Consider the main loop of a preconditioned conjugate gradients algorithm, as captured in the following pseudocode: Our modifications which yield

$$
\begin{aligned}
&\textbf{for}\ \ k = 1 : N \\
&\qquad \mathbf{q}_k \leftarrow \mathbf{A}\mathbf{p}_k \qquad (4.3)\\
&\qquad \alpha_k \leftarrow \mathbf{r}_k{}^{\mathsf{T}}\mathbf{z}_k / \mathbf{p}_k{}^{\mathsf{T}}\mathbf{q}_k \\
&\qquad \mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k \qquad (4.4)\\
&\qquad \mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \alpha_k \mathbf{q}_k \\
&\qquad \mathbf{z}_k \leftarrow \mathbf{M}^{-1}\mathbf{r}_k \qquad (4.5)\\
&\qquad \beta_k \leftarrow \mathbf{z}_{k+1}{}^{\mathsf{T}}\mathbf{r}_{k+1} / \mathbf{z}_k{}^{\mathsf{T}}\mathbf{r}_k \\
&\qquad \mathbf{p}_{k+1} \leftarrow \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k
\end{aligned}
$$

a mixed-precision implementation are summarized as follows:

- Vectors $\mathbf{r}$, $\mathbf{q}$, $\mathbf{z}$ and $\mathbf{p}$ (colored blue, above) are persistently stored in single-precision floating-point variables.
- The solution $\mathbf{x}$ is stored in double precision. The accumulation operation in (4.4) is also performed in double precision.
- The operator application in line (4.3) is performed in double precision (the single-precision input $\mathbf{p}$ is up-cast to double precision prior to the multiplication). The result of the operator application is then truncated to single precision and stored into $\mathbf{q}$.
- The multigrid V-cycle used as the preconditioner $\mathbf{M}^{-1}$ in line (4.5) is modified as follows: The smoother at the finest level uses double precision for the application of the operator, just as in line (4.3), although inputs and outputs are stored in single-precision. Every level

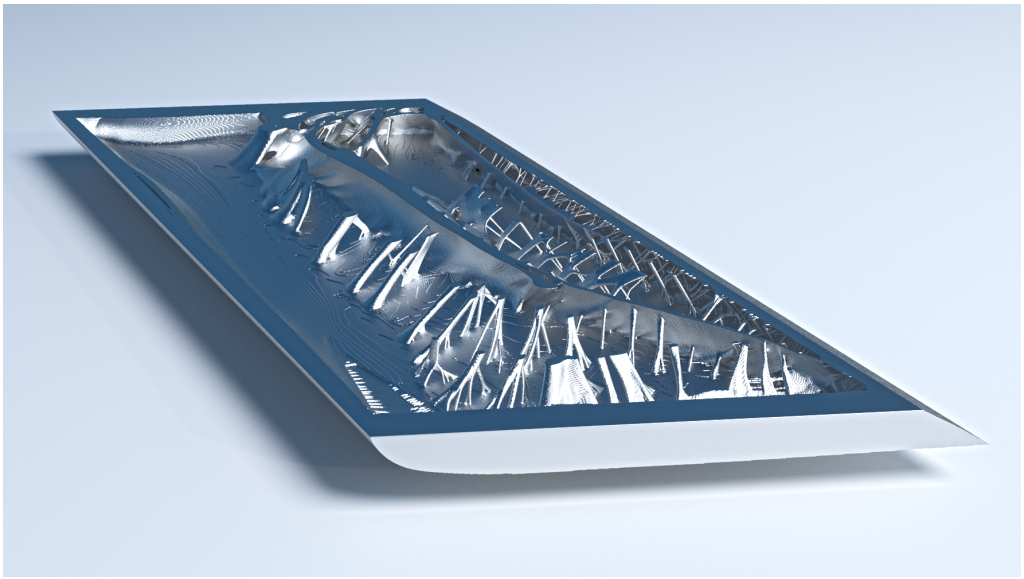


Figure 4.6: An optimized interior supporting structure of part of a wing is generated using our algorithm on a $1696 \times 342 \times 1971$ grid (402 M active voxels).

of the V-Cycle other than the finest uses double-precision arithmetic entirely.

Using this mixed-precision approach, the memory footprint of our solver is further reduced, providing a significant boost in the maximum resolution we can accommodate for a given amount of physical memory (128 bytes/node suffice to store all variables necessary for the MGPCG solver as well as the minimum compliance optimizer. Using full double precision, it would require 256 bytes/node for MGPCG solver and minimum compliance optimizer). Our results and validation section provides experimental evidence indicating that this mixed-precision approach yields almost identical accuracy of final results in all our tests.

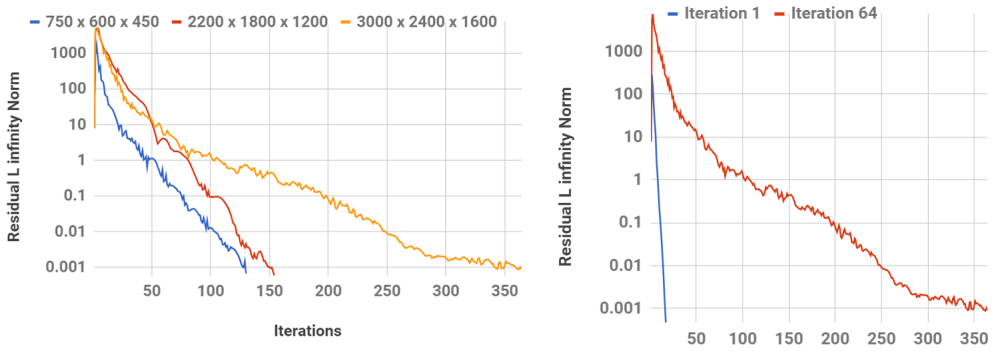


Figure 4.7: (Left) Convergence comparison of the bird beak example of different resolutions at the last topology optimization iteration. (Right) Convergence comparison of the one-billion-voxel bird beak example at different topology optimization iterations. All residuals reported in this work are L_∞ norm across all active cells, normalized relative to the L_∞ norm of the load.

4.7 Multigrid Solver Validation

Algorithmically, our implementation of the solver is a standard multi-linearly interpolated Garlerkin-coarsened Multigrid preconditioned conjugate gradient method. At early topology optimization iterations or small domains, such method proves to have fast asymptotic convergence. But in our examples, especially the bird beak Figure 4.1 and the wing Figure 4.6, the high variation of material spatial distribution has challenged the convergence of the multigrid solver as shown by Figure 4.7 (left). Thiner features resulted from Higher resolution, can significantly impact the convergence of multigrid as indicated in Figure 4.7 (right).

Besides convergence, the high resolution has also pushed the numerical limited of floating point. To validate our mixed precision scheme, we have conducted the following tests, both on analytical structures and the structures naturally emerged from topology optimization: 1. The last iteration of the bird beak example; 2. The last iteration of the wing example; 3-8. Homogeneous and isotropic cantilever beams of different length with

one side fixed and the other side loaded with a uniform downward force.

Table 4.1 shows the final residual of five different precision schemes: 1. Full double precision; 2. Only solution vector and computation are in double precision, the mix precision scheme we used in our topology optimization; 3. Only solution vector is in double precision; 4. Only computation is in double precision; 5. All in float precision. The results shows that under all examples our proposed mix precision scheme(column 2) can reduce the residual to an order of magnitude close to the double precision. Due to the fact that at high resolutions, cells that are far from Dirichlet boundaries have large displacements but yet only small strains. This loss of precision leaves it insufficient to use single precision for the solution vector. As shown in (column 4 and 5), using single precision for solution can result in inaccurate computation of strain and final residual. Similarly, using single-precision multiply will not be able to compute search direction to sufficient accuracy, conjugate gradient, in this case, will halt due to a detected singularity (column 3 and 5).

Table 4.1: The final residuals of different precision schemes after the same number of iterations. Test cases include bird beak, plane wing, and cantilever beam (CB) with different resolutions. The final residuals are recomputed based on solution vectors. All tests are scaled to initial residual infinity norm of 1. From the left to right, the 5 schemes are: 1. Full double precision; 2. Only solution vector and computation are in double precision; 3. Only solution vector is in double precision; 4. Only computation is in double precision; 5. All in float precision. SINGULAR indicates conjugate gradients have halted due to detected singularity.

Example	Double Precision	Mix Precision (double multiply)	Mix Precision (single multiply)	Single Precision (double multiply)	Single Precision (single multiply)
Bird beak	9.192e-5	9.063e-5	1.969e-4	6.846e+0	7.611e+0
Wing	8.968e-5	1.815e-4	SINGULAR	1.850e+1	SINGULAR
CB (32x32x32)	7.942e-6	7.942e-6	5.827e-5	2.259e-5	5.827e-5
CB (32x32x64)	8.217e-6	8.218e-6	5.773e-5	4.905e-5	6.019e-5
CB (32x32x128)	8.207e-6	8.208e-6	1.041e-3	1.018e-4	1.041e-3
CB (32x32x256)	9.322e-6	9.321e-6	6.295e-3	1.934e-4	6.295e-3
CB (32x32x512)	4.506e-6	4.508e-6	SINGULAR	3.990e-4	SINGULAR
CB (32x32x1024)	5.237e-6	5.242e-6	SINGULAR	7.043e-4	SINGULAR

5 STENCIL AWARE GALERKIN COARSENED MULTIGRID FOR LINEAR ELASTICITY

In Chapter 4, we examined an efficient implementation of a standard multilinear, Galerkin coarsened multigrid preconditioned conjugate gradient algorithm for linear elasticity. In Figure 4.7, we have observed that with increasing resolution (left), and domain complexity(right) the MGPCG convergence is severely impaired. In this chapter, as a continuation to Chapter 2, we will examine the design of a stencil-aware Galerkin coarsened multigrid and its efficacy when solving complex domains in comparison to the multigrid method whose hierarchy is constructed using multilinear interpolation and Galerkin coarsening.

5.1 Related Work

Multigrid solvers have been one of the most potent types of solvers for large scale elliptic PDEs. In Chapter 2, we saw two classes of coarsening strategies: geometric coarsening and Galerkin coarsening. Geometric coarsening is fast and has been used in many applications, such as Zhu et al. (2010b), McAdams et al. (2011) and McAdams et al. (2010). It can, however, be unstable when the simulation domain has complex boundary conditions or highly varying material properties.

Galerkin coarsening, on the other hand, is significantly more expensive but has much stronger stability properties. It also gives the solver the freedom of choosing both the coarse grid degrees of freedom and the prolongation operator. Many works have been dedicated to discovering the optimal strategy to find coarse grid degrees of freedom. Smoothed aggregation(SA), introduced by Vaněk et al. (1996), has been the most popular method to accomplish this. *BoomerAMG* by Yang et al. (2002) is a commercial algebraic multigrid solver that uses the smoothed aggregation

method. The main issue with the smoothed aggregation method is that instead of creating a regular mesh, like a 2-dimensional quad mesh, at each level of the multigrid hierarchy, it may create meshes with mixed elements of connectivity. In 2D, instead of each node being shared by 4 quad elements, some nodes at coarse levels may be shared by an arbitrary number of elements, which include both quads and triangles. Such irregularity causes many issues for optimization on modern hardware. Therefore, for better regularity at coarse levels, throughout this work, we always choose to geometrically coarsen the degrees of freedom, while retaining the freedom to choose the prolongation operator. For selecting the prolongation operator, Dendy (1982) introduced the blackbox multigrid for Poisson problems which geometrically coarsens the degrees of freedom, but constructs a constraint-local problem for computing the prolongation operator. It has been proven to be extremely efficient for solving Poisson problems with domains of highly varying density fields. In the field of elasticity, the most commonly used interpolation scheme is still, however, multilinear (see Chapter 4, Wu et al. (2016c), Wu et al. (2016b)). The main reason multilinear interpolation is the best interpolation for regularly coarsened hierarchies in 3D, is due to rank deficiency of the local problem. I will provide the proof for this statement in Section 5.4. In work by Brezina et al. (2001), vigorous proofs were given on what is the optimal criteria in selecting the prolongation operator. The AMGe method introduced in this work can give the optimal prolongation operator with its local optimal criteria. But AMGe requires the storage of the element matrices, which is prohibitively expensive. For a $128 \times 128 \times 128$ domain, the storage requirement of the element matrices is approximately $87\mathrm{GB}$. In work by Henson and Vassilevski (2001), an approximate method using harmonic extensions was proposed to enable the use of AMGe without storing the element matrices. In work by Dohrmann (2007), augmented degrees of freedom were introduced to overcome the rank deficiency of

the local problem in 3D, but the work assumes the knowledge of the local null space, which is not always easily accessible.

In this Chapter, a new Stencil Aware MultiGrid (SAMG) for linear elasticity will be introduced. It is based on the works by Brezina et al. (2001) and Dohrmann (2007), with the following improvements:

- We designed a method for computing the prolongation operator using matrix free operations that can compute the exact optimal prolongation operator based on the local optimal criteria given by Brezina et al. (2001). This provides significant memory savings, especially AMGe requires not only store the system matrix, but also, element matrices.
- The first level coarsening strategy combines Dohrmann (2007) and Brezina et al. (2001). However, at coarser levels, we propose an easy and inexpensive scheme to compute the prolongation operator that does not requires any local matrix analysis and guarantees the correct interpolation of rigid body motion.
- For the selection of smoothers, we provide insights into why we choose a box smoother over others. We also provide experimental evidence that the combination of the box smoother and the stencil aware coarsening strategy provides excellent convergence over other options.

5.2 Selection of Coarse Grid Nodes and Prolongation Operator Sparsity

In this chapter, we will examine linear elasticity problems discretized with 3D eight-node cube element on a Cartesian grid. Therefore, the position

of node (i, j, k) in undeformed space $\mathbf{X}$, can be written as:

$$\mathbf{X}(i, j, k) = (i \cdot h, j \cdot h, k \cdot h), (i, j, k) \in G^0$$

Here, h is the element length and G^0 is the finest level of discretization. Since we wish to maintain regularity across all levels, we always choose to geometrically coarsen the grid. We can write the position of node (i, j, k) in undeformed space $\mathbf{X}$ at level 1 as:

$$\mathbf{X}(i, j, k) = (i \cdot 2h, j \cdot 2h, k \cdot 2h), (i, j, k) \in G^1$$

The nodal positions for the coarser levels can be selected similarly. After we have selected the coarse grid degrees of freedom, a proper prolongation operator is required for building the multigrid hierarchy. Given that the finest level operator $\mathbf{L}^0$ has a regular 27 point stencil sparsity, we desire all coarse level operators $\mathbf{L}^c$ to share the same sparsity. Here we propose a sparsity pattern for the prolongation operator that guarantees such a property. This matches the sparsity pattern of standard multilinearly-interpolated prolongation operators in Chapter 4. Although it is not the only operator that satisfies, this is the most commonly used, and fits well with the methods for computing the prolongation operator in the next sections.

1. If a fine node f coincides with a coarse node c in undeformed space, the prolongation stencil only includes that one coarse node, i.e. $P_{fc^*} = 0$, if $c^* \neq c$. *Figure 5.1 left.*

2. If a fine node f lies on the center of a coarse-element edge in undeformed space, the prolongation stencil includes the coarse nodes on the two ends of the edge. Let coarse node c_1 and c_2 define this edge, $P_{fc^*} = 0$, if $c^* \neq c_1$ and $c^* \neq c_2$, *Figure 5.1 middle.*

3. If a fine node f lies on the center of a coarse element face in unde-

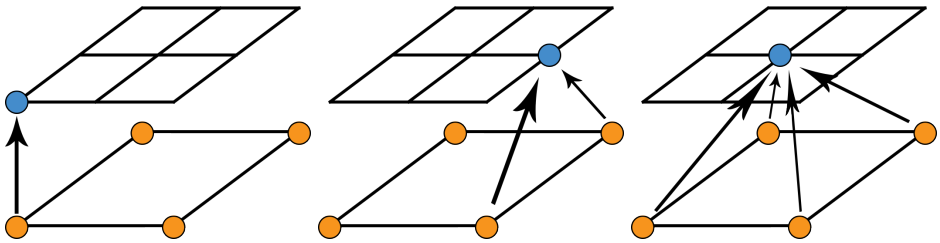


Figure 5.1: Illustration of the prolongation stencil sparsity in 2D. Top is the fine grid, bottom is the coarse grid. On the left, is a fine node coinciding with a coarse node. The prolongation stencil is a single injection. In the middle, the fine node lies on the edge of a coarse cell. The prolongation stencil consists of two spokes connecting to the two ends of the edge. On the right, the fine node lies on the center of the coarse node. The prolongation stencil includes spokes to all the nodes of the coarse cell.

formed space, the prolongation stencil includes the coarse nodes at the four corners of the face. If coarse node c_1, c_2, c_3, and c_4 define this face and, $P_{fc^*} = 0$, then $c^* \neq c_p, \forall c_p \in \{c_1, c_2, c_3, c_4\}$.

4. If a fine node f lies in the center of a coarse element in undeformed space, the prolongation stencil includes all eight nodes of the coarse element. If coarse node c_1, c_2, c_3, c_4, c_5, c_6, c_7, and c_8 define this coarse element, $P_{fc^*} = 0$, if $c^* \neq c_p, \forall c_p \in \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$. Figure 5.1 right.

5.3 Building the Prolongation Operators Using Local Problems

Defining the Local Matrix

In section 2.7, we stated that the quality of the prolongation operator can be measured by two metrics:

$$M_1(\mathbf{Q}, \mathbf{e}) := \frac{< (\mathbf{I} - \mathbf{Q})\mathbf{e}, (\mathbf{I} - \mathbf{Q})\mathbf{e} >}{< \mathbf{L}\mathbf{e}, \mathbf{e} >} \tag{5.1}$$

$$M_2(\mathbf{Q}, \mathbf{e}) := \frac{< \mathbf{L}(\mathbf{I} - \mathbf{Q})\mathbf{e}, (\mathbf{I} - \mathbf{Q})\mathbf{e} >}{< \mathbf{L}\mathbf{e}, \mathbf{e} >} \tag{5.2}$$

Here we take $\mathbf{Q}$ as:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{0} & \mathbf{P}_{cf} \\ \mathbf{0} & \mathbf{I}_c \end{bmatrix}$$

Both measurements require the global operator $\mathbf{L}$. In work presented by Brezina et al. (2001), an alternative local operator $\mathbf{L}_i$ was proposed. $\mathbf{L}_i$ is the sum of the element stiffness matrix adjacent to a given fine node i.

$$\mathbf{L}_i = \sum_{\alpha \in \mathcal{T}_i} L_\alpha \tag{5.3}$$

L_α is the elemental stiffness matrix of element α, $\mathcal{T}$ is the collection of cells that are adjacent to the fine node f (called the fine node's ring neighborhood), illustrated in Figure 5.2. The localized measurement $M_{i,1}$ and $M_{i,2}$ can then be defined as:

$$M_{i,1}(\mathbf{Q}, \mathbf{e}) := \frac{< \epsilon_i \epsilon_i^T (\mathbf{I} - \mathbf{Q})\mathbf{e}, \epsilon_i \epsilon_i^T (\mathbf{I} - \mathbf{Q})\mathbf{e} >}{< \mathbf{L}_i \mathbf{e}, \mathbf{e} >} \tag{5.4}$$

$$M_{i,2}(\mathbf{Q}, \mathbf{e}) := \frac{< \mathbf{L}_i \epsilon_i \epsilon_i^T (\mathbf{I} - \mathbf{Q})\mathbf{e}, \epsilon_i \epsilon_i^T (\mathbf{I} - \mathbf{Q})\mathbf{e} >}{< \mathbf{L}_i \mathbf{e}, \mathbf{e} >} \tag{5.5}$$

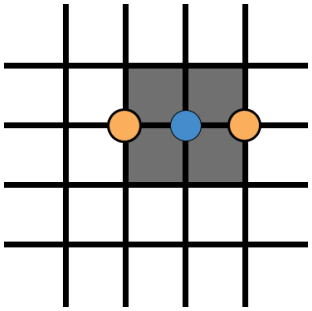


Figure 5.2: A neighborhood of a given fine node that aligns with a coarse cell edge, shaded in blue, in a Cartesian discretization. The neighboring cells of the given node are shaded in gray. In this case, we are interpolating the fine node values from the given coarse nodes, shaded in yellow.

Here ϵ_i is a vector, whose entries satisfy:

$$\epsilon_i(j) = \delta_{ij}$$

where δ is the Kronecker delta. If we denote the set of projection operators $\mathbf{Q}$ that satisfy the sparsity conditions stated in Section 5.2 as $\mathcal{Z}_i$, the ith row if $\mathbf{Q}$ is then:

$$\mathbf{q}_i^T = \epsilon_i^T \mathbf{Q}$$

and finding the prolongation operator for a given fine node i can be expressed by the following min-max problem:

$$\begin{aligned} K_{i,p} &= \min_{\mathbf{q}_i \in \mathcal{Z}_i} \max_{\mathbf{e} \perp \text{Null}(\mathbf{L}_i)} M_{i,p}(\mathbf{q}_i, \mathbf{e}) \\ \text{subject to } &(\epsilon_i - \mathbf{q}_i)^T \mathbf{e} = 0, \forall \mathbf{e} \in \text{Null}(\mathbf{L}_i) \end{aligned} \tag{5.6}$$

The condition in Equation 5.6 states that the prolongation operator must correctly interpolate local null space. For Poisson problems, the null space dimension of the local matrix is only 1, but for elasticity problems, the null space dimension of the local matrix is drastically raised to 6 in 3D: three for displacements and three for linearized rotations.

Computation of the Prolongation Operator

For the local problem defined above with fine node i, we denote the set of nodes included in this local problem as $\mathcal{N}_i$. We write c as the coarse nodes that fine node i is interpolated from. f as the fine nodes from $\mathcal{N}_i$. n_c is the size of the set c, and n_f is the size of the set f. We can, therefore, express the local matrix L_i as:

$$\mathbf{L}_i = \begin{bmatrix} \mathbf{L}_{ff}^{(1)} & \mathbf{L}_{fc}^{(1)} \\ \mathbf{L}_{cf}^{(1)} & \mathbf{L}_{cc}^{(1)} \end{bmatrix}$$

for metric $M_{i,1}$, and:

$$\mathbf{L}_i^2 = \begin{bmatrix} \mathbf{L}_{ff}^{(2)} & \mathbf{L}_{fc}^{(2)} \\ \mathbf{L}_{cf}^{(2)} & \mathbf{L}_{cc}^{(2)} \end{bmatrix}$$

for metric $M_{i,2}$. If we place node i as the first of all the nodes f. We can replace ϵ_i with ϵ_1 in the two local metrics.

Lemma 5.1. *There exits* $\mathbf{q}_i \in \mathcal{Z}_i$, *such that* $\epsilon_1 - \mathbf{q}_i \in$ *Range*$(\mathbf{L}_i^p)$ *if and only if*

$$\hat{\epsilon}_1 \in \textit{Range}(\mathbf{L}_{ff}^{(p)})$$

$\hat{\epsilon}_1$ *is the first canonical basis vector of length* n_f. $p = 1$ *or* 2.

For a more special case, if $\mathbf{L}_{ff}^{(p)}$ is invertible, then there always exists an interpolation that correctly prolongates error vectors that are in the null space of the local problem. In our application, $\mathbf{L}_{ff}^{(p)}$ is, in general, invertible, therefore, at least the constructed prolongation operator can prolongate null space correctly.

Theorem 5.1. If $\hat{\epsilon}_1 \notin \text{Range}(\mathbf{L}_{ff}^{(p)})$, then $K_{i,p} = \infty$. If $\hat{\epsilon}_1 = \mathbf{L}_{ff}^{(p)}\hat{\delta}_1$, and the unique solution of equation 5.6 is given by

$$\mathbf{q}_i^* = \begin{pmatrix} 0 \\ -\mathbf{L}_{cf}^{(p)}\hat{\delta}_1 \end{pmatrix} \in \mathcal{Z} \tag{5.7}$$

and $K_{i,p} =< \hat{\epsilon}_1, \hat{\delta}_1 >$, for $p = 1$ or 2.

The first part of Theorem 5.1 states that if no solution exists, multigrid cannot converge for the error mode containing node i.

The second part of Theorem 5.1 states that if the solution exists, we can compute the prolongation operator using the following formula:

$$
\begin{aligned}
\hat{\epsilon}_1 &= \mathbf{L}_{ff}^{(p)} \hat{\delta}_1 \\
\hat{\delta}_1 &= (\mathbf{L}_{ff}^{(p)})^{(-1)} \hat{\epsilon}_1 \\
\mathbf{q}_i^* &= (\mathbf{e}_i^T \mathbf{Q})^T \\
&= \mathbf{Q}^T \epsilon_i \\
&= \begin{pmatrix} 0 \\ \mathbf{P}_{fc}^T \epsilon_i \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ -\mathbf{L}_{cf}^{(p)} \hat{\delta}_1 \end{pmatrix} \\
\mathbf{P}_{fc}^T \epsilon_i &= -\mathbf{L}_{cf}^{(p)} \hat{\delta}_1 \\
&= -\mathbf{L}_{cf}^{(p)} (\mathbf{L}_{ff}^{(p)})^{-1} \hat{\epsilon}_1
\end{aligned}
\tag{5.8}
$$

$\mathbf{P}_{fc}^T \epsilon_i$ is a row of the prolongation operator. Here we require the inverse of $\mathbf{L}_{ff}^{(p)}$. If the inverse exists, from From Lemma 5.1, there exists prolongation operators that can correctly interpolate null space. Furthermore, Equation 5.8 is a prolongation operator that can correctly interpolate null space.

A practical way for computing $\mathbf{P}_{fc}^T \epsilon_i$ when using metric $M_{i,1}$, is to set the coarse nodes in the local problem as Dirichlet. Then we set each degree of freedom j with value of 1, i.e. a Kronecker delta, one by one. For each Kronecker delta imposed boundary problem, the value(s) of the DOF of fine node i in the solution vector is then the entry $\epsilon_i^T \mathbf{P}_{fc} \epsilon_j$. Each row of the prolongation operator is the harmonic extension of a canonical basis vector.

If the reader is interested in the proof of Lemma 5.1 and Theorem 5.1, we refer to Brezina et al. (2001). Here, we will only use the results and view the implications. Furthermore, it is worth noting that in work Henson and Vassilevski (2001), an alternative formulation of Equation 5.6 in the perspective of energy minimization was proposed, upon which Dohrmann (2007) was based. As it is not essential for the derivation of our method, we will omit that formulation here.

5.4 Multigrid Method with Augmented Variables

The condition of Equation 5.6, states that the prolongation operator must correctly interpolate error vectors $\mathbf{e}$ that are in the null space of the local matrix $\mathbf{A}_i$. Here we will give without prove that standard multi-linear interpolation satisfies this condition for both linear elasticity and Poisson problems, as it is used for homogeneous problems and proven to be effective for those problem. In Section 5.2, a sparsity condition for the prolongation operator was proposed to guarantee the sparsity of the coarse level operators. Considering Figure 5.2, the fine node is interpolated from the edge adjacent two coarse nodes. For 3D elasticity, in the cases that the fine node lies on the center of a coarse cell edge, the two coarse nodes provide 6 equations for solving the local prolongation operator. Note that the null space dimension of $\mathbf{A}_i$ is also 6. Which means there is only one prolongation operator satisfies the condition of Equation 5.6, and we know that the standard multi-linear interpolation satisfies this condition. Therefore:

Lemma 5.2. *Multi-linear interpolation is the **only** interpolation that can capture error vectors in the null space of local matrix $\mathbf{L}_i$ for fine nodes that are interpolated from only two coarse nodes in 3D linear elasticity problems.*

Given that the largest $M_{i,p}$ is the upper bound for the multigrid convergence rate, which means it is very difficult to improve the convergence over multi-linear interpolated multigrid while maintaining our sparsity constraints for the prolongation operator. Dohrmann (2007) has made similar observations. To overcome this limitation, he has proposed that, by introducing additional degrees of freedom in the coarse grids, better solutions to Equation 5.6 can be achieved.

Introducing Linearized Rotational Degrees of Freedom

Similar to Dohrmann (2007), we introduced linearized rotational degrees of freedom to the coarse grid to improve the quality of the prolongation operator. Given that our top level operator is matrix free as stated in Chapter 4, we can easily compute the element matrix on the top level and, therefore, we can avoid the local null space analysis suggested by Dohrmann (2007). First, we will introduce the additional linearized rotational DOF and their physical interpretation. Then an algorithm will be proposed for computing the prolongation operator given by Theorem 5.1 with the expanded DOF. Figure 5.3 illustrates the deformation created by the linearized rotation DOF θ_L of node L in 2D. The rotational degrees of freedom associated with node L, namely θ_L, is viewed to influence the horizontal, and only the horizontal, displacement of nodes LT and LB. If we write the the coarse DOF of node L, displacements and rotation, we can denote the total fine DOF $\mathbf{u}_L^c$ as:

$$\mathbf{u}_L^c = \begin{pmatrix} u_L^c(x) \\ u_L^c(y) \\ u_L^c(\theta) \end{pmatrix}$$

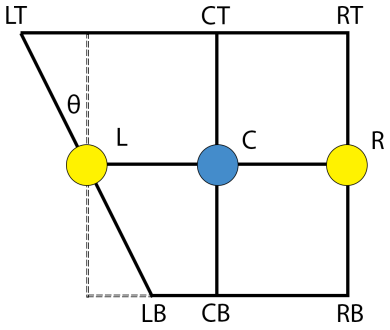


Figure 5.3: A 2D illustration of a rotational degree of freedom θ and its physical interpretation. In this case, the fine node, marked blue, is aligned with a coarse cell edge. The fine node is interpolated from the two adjacent coarse nodes, marked yellow. Letters are the associated names for the nodes. The solid lines indicates the deformed cells after applying the given rotation θ. The doted lines illustrated the undeformed cells.

The superscript ^{c} indicates the DOF live on the coarse grid. Accordingly, the fine DOF of nodes *LT*, *L*, and *LB*, then can be computed:

$$\mathbf{u}^f_{LT} = \begin{pmatrix} u^c_L(x) - u^c_L(\theta)h \\ * \end{pmatrix}$$

$$\mathbf{u}^f_{L} = \begin{pmatrix} u^c_L(x) \\ u^c_L(y) \end{pmatrix}$$

$$\mathbf{u}^f_{LB} = \begin{pmatrix} u^c_L(x) + u^c_L(\theta)h \\ * \end{pmatrix}$$

h is the cell width. The symbol $*$ indicates that this DOF is not fixed for the local problem. Similarly we can write the right side nodal values as:

$$\mathbf{u}^f_{RT} = \begin{pmatrix} u^c_R(x) - u^c_R(\theta)h \\ * \end{pmatrix}$$
$$\mathbf{u}^f_{R} = \begin{pmatrix} u^c_R(x) \\ u^c_R(y) \end{pmatrix}$$
$$\mathbf{u}^f_{RB} = \begin{pmatrix} u^c_R(x) + u^c_R(\theta)h \\ * \end{pmatrix}$$

So if we rearrange the local matrix in Equation 5.3, so that the nodal value $\mathbf{u}^f_C$ comes first, then the other undetermined(or free) nodal values, and last, the eight determined nodal values. We can write the DOF of local problem as vector $\mathbf{u}^f_D$:

$$\mathbf{u}^f_D = \left(\mathbf{u}^f_{LT}(x), \mathbf{u}^f_{LT}(x), \mathbf{u}^f_{LT}(y), \mathbf{u}^f_{LB}(y), \mathbf{u}^f_{RT}(x), \mathbf{u}^f_{RT}(x), \mathbf{u}^f_{RT}(y), \mathbf{u}^f_{RB}(y)\right)^T$$

Similarly we can rearrange the local matrix and denote it as:

$$\mathbf{L}_i = \begin{bmatrix} \mathbf{L}_{ff} & \mathbf{L}_{fd} \\ \mathbf{L}_{df} & \mathbf{L}_{dd} \end{bmatrix} \tag{5.9}$$

Here, $\mathbf{L}_{dd}$ is a 8×8 matrix, $\mathbf{L}_{ff}$ is of dimension 10×10. Based on theorem 5.1, the free fine nodal DOF vector $\mathbf{u}^f$ of size 10 can than be computed as:

$$\mathbf{u}^f = -(\mathbf{L}_{ff})^{-1}\mathbf{L}_{fc}\mathbf{u}^f_D \tag{5.10}$$

Given that each component in vector $\mathbf{u}^f_D$ can be computed from $\mathbf{u}^c_L$ and $\mathbf{u}^c_R$ using a linear transformation. We can construct a transformation matrix T, s.t.:

$$\mathbf{u}^f_D = \mathbf{T}\mathbf{u}^c \tag{5.11}$$

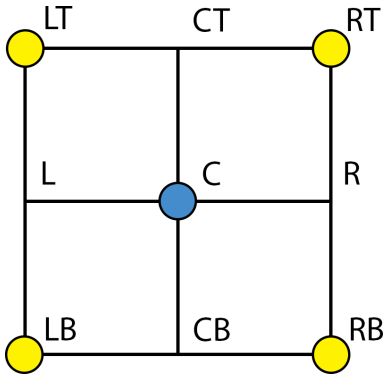


Figure 5.4: Interpolation of a fine node, blue, that is at the center of a coarse cell. The coarse nodes are shaded in yellow. Each node is given a name.

The local prolongation operator is therefore:

$$\mathbf{P}_i = -(\mathbf{L}_{ff})^{-1}\mathbf{L}_{fd}\mathbf{T} \tag{5.12}$$

The first two rows of $\mathbf{P}_i$ are the prolongation stencils of the fine node i. By using $\mathbf{L}_i$ instead of $\mathbf{L}_i^2$, our local convergence metric here is $M_{1,q}$.

Interpolation of Cell Centered Nodes in 2D

Now that we have interpolated the fine nodes that are aligned with the coarse cell edges, the next step is to interpolate the fine nodes that are located at the center of the coarse cells, Figure 5.4 Notice that here node *CT* can be interpolated from node *LT* and *RT*. Similarly to node *L*, *R*, and *CB*. As a matter of facts, that all the 8 neighbors of node *C* can be interpolated from the given coarse nodes. Therefore, we can prolongate the node *C* so that it has zero residual at the fine level. If i is the index of node *C*, and $\mathcal{N}_i$ is the set of nodes that are adjacent to node i, i.e.:

$$\mathcal{N}_i = \{LB, CB, RB, L, R, LT, CT, RT\}$$

We can solve for the nodal value of node i, denoted as $\mathbf{u}_i^f$ using the following equation:

$$\mathbf{L}_{ii}\mathbf{u}_i^f + \sum_{j \in \mathcal{N}_i} \mathbf{L}_{ij}\mathbf{P}_j\mathbf{u}^c = 0 \tag{5.13}$$

Here $\mathbf{P}_j$ is the jth rows of the prolongation operator that computes the nodal values of j. In 2D, it is two rows, and in 3D, it is three rows. $\mathbf{P}_j\mathbf{u}^c$ computes the nodal value vector $\mathbf{u}_j^f$. $\mathbf{L}_{ij}$ is the stencil coefficient coupling node j and i. It is a 2×2 matrix in 2D and 3×3 matrix in 3D. We can write the solution of $\mathbf{u}_i^f$ as:

$$\mathbf{u}_i^f = -\sum_{j \in \mathcal{N}_i} (\mathbf{L}_{ii})^{-1}\mathbf{L}_{ij}\mathbf{P}_j\mathbf{u}^c \tag{5.14}$$

Then, the prolongation operator for node i, $\mathbf{P}_i$ is:

$$\mathbf{P}_i = -\sum_{j \in \mathcal{N}_i} (\mathbf{L}_{ii})^{-1}\mathbf{L}_{ij}\mathbf{P}_j \tag{5.15}$$

Interpolation of Fine Nodes that Coincide with a Coarse node in 2D

This is the simplest case, same as a standard multi-linear interpolation. The fine node displacement are injected from the corresponding coarse node. The rotational DOF are ignored as they do not exist in the finest level.

5.5 Rotational Degrees of Freedom in 3D

Fine nodes in 3D can be categorized into 4 types as stated in section 5.2:

1. coinciding with a coarse node
2. laying on a coarse cell edge center

3. laying on a coarse cell face center

4. laying on a coarse cell cell center

For case 1 and case 4, the expressions for computing the prolongation operator are almost identical for 2D and 3D. This section will focus on case 2 and 3, and although they are similar to the 2D case, they still require special treatment.

Interpolating Fine Nodes that Lay on a Coarse Cell Edge Center

The prolongation operator can be computed with the same formula as Equation 5.12 as in the 2D case, but the arrangement of the local matrix is different. Consider a one ring neighborhood of a fine node i, $\mathcal{N}_i$ that contains 26 nodes. Without loss of generality, we assume node i lies on a coarse node edge aligned with the x axis. Therefore if the node i has geometric index (x_i, y_i, z_i) in the Cartesian grid. We are interpolating its value from node c_L with geometric $(x_i - 1, y_i, z_i)$ and node c_r with geometric $(x_i + 1, y_i, z_i)$. If we denote the DOF of the coarse nodes as:

$$\mathbf{u}^c = \begin{pmatrix} u_x \\ u_y \\ u_z \\ r_x \\ r_y \\ r_z \end{pmatrix}$$

$\mathbf{u}_l^c$ is the vector of nodal values for node c_L and $\mathbf{u}_R^c$ is the vector of nodal values for node c_R. The fine node DOF at geometric index $(x_i - 1, y_i, z_i)$

and $(x_i + 1, y_i, z_i)$ is computed using injection:

$$\mathbf{u}^f(x_i - 1, y_i, z_i) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \mathbf{u}_L^c \tag{5.16}$$

Similarly $\mathbf{u}^f(x_i + 1, y_i, z_i)$ can be computed. With regards to the rotation DOF of a given coarse node, it is important to note that they influence only the fine nodes in $\mathcal{N}_i$ that are connected to the coarse node through an edge and not through the center fine node i. For instance, rotational DOF of node located at coarse node c_L with geometric index $(x_i - 1, y_i, z_i)$, influence the following 4 fine nodes in $\mathcal{N}_i$.

$$\mathcal{R}_L = \{(x_i - 1, y_i + 1, z_i), (x_i - 1, y_i - 1, z_i), (x_i - 1, y_i, z_i + 1), (x_i - 1, y_i, z_i - 1)\} \tag{5.17}$$

$\mathcal{R}_L$ is the set of nodes influenced by rotational DOF of node c_L. Similarly, we can define $\mathcal{R}_R$. As an example, take node $(x_i - 1, y_i - 1, z_i)$. Because of symmetry and without loosing generality, we can set this nodes nodal value in the local problem as the vector between the node $(x_i - 1, y_i, z_i)$ and $(x_i - 1, y_i - 1, z_i)$ is $(0, -h, 0)$. Again h here is the cell size. If the rotational DOF are (r_x, r_y, r_z). Then we can write the linearized rotational matrix as:

$$\mathbf{R}_L^c = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & r_z \end{bmatrix} \tag{5.18}$$

Then displacement of node $(x_i - 1, y_i - 1, z_i)$ caused by the rotation around the coarse node is then:

$$\mathbf{R}_L^c \begin{bmatrix} 0 \\ -h \\ 0 \end{bmatrix} = \begin{bmatrix} hr_z \\ 0 \\ -hr_x \end{bmatrix} \tag{5.19}$$

Here the 2nd dimension of the displacement is not influenced by the rotation of the coarse node. Therefore we set it as a free variable. Now we can write the displacement of node $(x_i - 1, y_i - 1, z_i)$ on the fine grid as:

$$\mathbf{u}^f(x_i - 1, y_i, z_i) = \begin{bmatrix} 1 & 0 & 0 & 0 & h & 0 \\ * & * & * & * & * & * \\ 0 & 0 & 1 & 0 & -h & 0 \end{bmatrix} \mathbf{u}_L^c \tag{5.20}$$

The symbol $*$ here indicates that the DOF associated with the row is set to be a free variable in the local problem. Now that we have prescribed a selection of fine node DOF in the local problem, we can rearrange the local matrix $\mathbf{L}_i$ similar to equation 5.9, and compute the prolongation operator similarly to equation 5.12.

Interpolating Fine Nodes that Lay on a Coarse Cell Face Center

A face center fine node i, Figure 5.5, assuming its geometric index is (x_i, y_i, z_i), is adjacent with 4 nodes that coincide with a coarse node and 4 nodes that lie on a coarse cell edge and can be prolongated by the 4 coarse node within the one ring neighbor hood of i. We call them *dependent nodes*. Without loss of generality, assume the fine node lies on a xy aligned face. The adjacent coarse node set is then:

$$\mathcal{C} = \{(x_i - 1, y_i - 1, z_i), (x_i - 1, y_i + 1, z_i), (x_i + 1, y_i - 1, z_i), (x_i + 1, y_i + 1, z_i)\} \tag{5.21}$$

The set of the dependent nodes that can be prolongated from nodes in $\mathcal{C}$ is then:

$$\mathcal{D} = \{(x_i - 1, y_i, z_i), (x_i + 1, y_i, z_i), (x_i, y_i - 1, z_i), (x_i, y_i + 1, z_i)\} \tag{5.22}$$

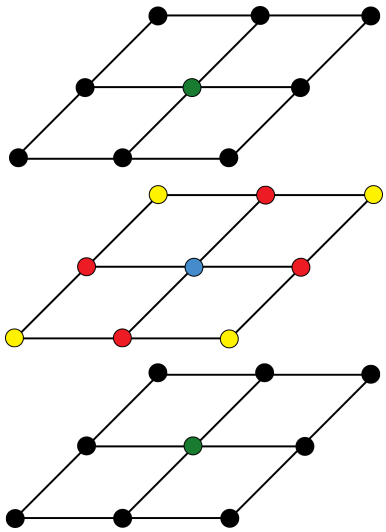


Figure 5.5: A face centered fine node, shaded blue; surrounded by 4 coarse nodes, shaded yellow; 4 dependent nodes, shaded red; 16 edge nodes that rotate along the coarse nodes and dependent nodes, shaded black; and 2 free nodes in the center of the coarse cells, shaded green.

We define the set of fine nodes that is influenced by the rotational DOF of each nodes in $\mathcal{C}$ as:

$$\mathcal{R}_1 = \{(x_i - 1, y_i - 1, z_i - 1), (x_i - 1, y_i - 1, z_i + 1)\} \tag{5.23}$$

$$\mathcal{R}_2 = \{(x_i - 1, y_i + 1, z_i - 1), (x_i - 1, y_i + 1, z_i + 1)\} \tag{5.24}$$

$$\mathcal{R}_3 = \{(x_i + 1, y_i - 1, z_i - 1), (x_i + 1, y_i - 1, z_i + 1)\} \tag{5.25}$$

$$\mathcal{R}_4 = \{(x_i + 1, y_i + 1, z_i - 1), (x_i + 1, y_i + 1, z_i + 1)\} \tag{5.26}$$

Those are the nodes with prescribed displacement in the local problem. With this we can again arrange the local matrix $\mathbf{L}_i$ similarly to equation 5.9, and compute the prolongation operator similarly to equation 5.12.

If we define $\mathbf{S^P}_{c,j}$, $j \in \mathcal{D}$ the prolongation stencil, a 3×3 matrix, from node j to the center node c. Similarly, define $\mathbf{S^P}_{j,k}$, $j \in \mathcal{D}$ or $j = c$ and $K \in \mathcal{C}$ the prolongation stencil, a 3×6 matrix, from coarse node k to a

embedded node j. The final prolongation stencil from coarse node k to the center node c, denoted as $\mathbf{P}_{c,k}$, can therefore be written as:

$$\mathbf{P}_{c,k} = \mathbf{S^P}_{c,k} + \sum_{j \in \mathcal{D}} \mathbf{S^P}_{c,j} \mathbf{S^P}_{j,k}, k \in \mathcal{C} \tag{5.27}$$

5.6 Construction of the Multigrid Hierarchy

Now with the computation of the prolongation operator $\mathbf{P}$, which is a 27 point stencil in 3D with each spoke a 3×6 matrix. We can construct a *two-level* multigrid using a standard Galerkin coarsening process. The multigrid solver is same as Algorithm 2.1.

5.7 Prolongation at Coarse Level

At coarse level, the building of the prolongation operator is quite different from the top level for three main reasons: first, the number of DOF per node has changed from 2 to 3 in 2D, and 3 to 6 in 3D. Second, the element cell matrix is no longer easily accessible, they can be computed using Galerkin coarsening of the cell matrices, but when building the local problems, same cell matrix may be reused up to 8 times. Storing them can void the overhead of recomputing the cell stiffness repeatedly but it would require $8 \times 8 \times 6 \times 6 = 2304$ doubles per element, which is prohibitively expensive. Third and most importantly, the local problems sometimes can has a nullity as many as 12, which is significant higher than the local problem at the finest level. But also, $\mathbf{L}_{ff}^{(p)}$ may always be invertible in Equation 5.8. Or worse, $\hat{e}_1$ may not be $\in$ Range$(\mathbf{L}_{ff}^{(p)})$ for Theorem 5.1. It is worth noting that based on Theorem 5.1, for some of the nodes $K_{i,p} = \infty$. It implies a point based smoother, Gauss-Seidel or Jacobi, can potentially

stagnate. Therefore, a smoother of a larger radius should be used. Now, to reconcile the three problems, and to build the prolongation matrix, here are two possible remedies:

1. Still using the local problem created by combining cell matrices, but project $\hat{\epsilon}_1$ onto Range($\mathbf{L}_{ff}^{(p)}$), and use a psudo-inverse of $\mathbf{L}_{ff}^{(p)}$, while still maintains the orthogonality condition in Equation 5.6.

2. Redefining the local problem so that $\hat{\epsilon}_1 \in$ Range($\mathbf{L}_{ff}^{(p)}$) and $\mathbf{L}_{ff}^{(p)}$ is invertible.

Using remedy 1, not only requires either store or compute the cell matrix on the fly, it also requires the singular value decomposition of the local matrix to compute the psudo-inverse of $\mathbf{L}_{ff}^{(p)}$ and correct project prolongation operator so that it can correct prolongate null energy modes. The potential high cost of remedy 1 made it very impracticable. Alternatively, we adopted remedy 2 that similar to Dendy (1982), in which the name black box multigrid is used. But here I refer the construction of the prolongation operator as *stencil collapse*, the reason will be obvious in the following section.

5.8 Building Prolongation Operator using Stencil Collapse

I want start to clarify that the method introduced in this section, though can correctly interpolate rigid body motion, it imposes significantly more artificial boundary condition in the local problem and can potentially creates prolongation operator that is worse than remedy 1. But it is significantly cheaper and in our examples has demonstrated that it is sufficient to have a good convergence behavior when paired with a good smoother.

The 2D Case

Let's consider 2D first. If the fine node is coincided with a coarse node geometrically in the undeformed space, the prolongation operator is a simple injection. If the fine node lies in the center of the coarse node, the expression for the prolongation operator is identical to Equation 5.15. The case that is different, is when the fine node lies on a coarse cell edge center, see Figure 5.3. In such case, we will assume that node *LT*, *L*, and *LB* are undergoing the same rigid body motion. So is *CT*, *C*, and *CB*. And *RT*, *R*, and *RB*. Therefore we can write:

$$\mathbf{u}^{f}_{*T} = \begin{pmatrix} u^{c}_{*}(x) - u^{c}_{*}(\theta)h \\ u^{c}_{*}(y) \\ u^{c}_{*}(\theta) \end{pmatrix}$$

$$\mathbf{u}^{f}_{*} = \begin{pmatrix} u^{c}_{*}(x) \\ u^{c}_{*}(y) \\ u^{c}_{*}(\theta) \end{pmatrix}$$

$$\mathbf{u}^{f}_{*B} = \begin{pmatrix} u^{c}_{*}(x) + u^{c}_{*}(\theta)h \\ u^{c}_{*}(y) \\ u^{c}_{*}(\theta) \end{pmatrix}$$

* here can be *L*, *C*, or *R*. h is the length of the cell element, which **doubles with each multigrid level**. We can see that each is a linear transformation from the center node. Therefore alternatively, we can write it as:

$$\mathbf{u}^{f}_{*T} = \mathbf{T}_{*T}\mathbf{u}^{f}_{*}$$

$$\mathbf{u}^{f}_{*} = \mathbf{T}_{*}\mathbf{u}^{f}_{*}$$

$$\mathbf{u}^{f}_{*B} = \mathbf{T}_{*B}\mathbf{u}^{f}_{*}$$

$\mathbf{T}$ is the linear transformation that transforms each node based on the rigid body defined by the central node. $\mathbf{T}_{*}$ is just the identity matrix. Now if

we define $\mathbf{L}_i(j)$ is the stencil between node i and j, i as the output node. Therefore, we may write the equilibrium local problem as:

$$\sum_{j\in\{LT,L,LB\}} \mathbf{L}_i(j)\mathbf{T}_j\mathbf{u}_L^f + \sum_{j\in\{RT,R,RB\}} \mathbf{L}_i(j)\mathbf{T}_j\mathbf{u}_R^f + \sum_{j\in\{CT,C,CB\}} \mathbf{L}_i(j)\mathbf{T}_j\mathbf{u}_C^f = 0 \tag{5.28}$$

If we collapse the stencils and define:

$$\begin{aligned}\mathbf{S}_L &= \sum_{j\in\{LT,L,LB\}} \mathbf{L}_i(j)\mathbf{T}_j \\ \mathbf{S}_C &= \sum_{j\in\{CT,C,CB\}} \mathbf{L}_i(j)\mathbf{T}_j \\ \mathbf{S}_R &= \sum_{j\in\{RT,R,RB\}} \mathbf{L}_i(j)\mathbf{T}_j\end{aligned}$$

Otherwise, we can write it as:

$$\mathbf{u}_C^f = -\mathbf{S}_C^{-1}\mathbf{S}_L\mathbf{u}_L^f - \mathbf{S}_C^{-1}\mathbf{S}_R\mathbf{u}_R^f \tag{5.29}$$

the prolongation stencil of the left node to the center node i is then:

$$\mathbf{P}_i(L) = -\mathbf{S}_C^{-1}\mathbf{S}_L \tag{5.30}$$

Similarly the right side:

$$\mathbf{P}_i(R) = -\mathbf{S}_C^{-1}\mathbf{S}_R \tag{5.31}$$

The 3D Case

For the 3D case, we can classify the fine node to the 4 different categories:

1. If the fine node is coincided with a coarse node geometrically in the undeformed space, the prolongation operator is a simple injection.

2. If the fine nodes lies on a coarse cell **edge center**, the derivation is

extremely similar to the 2D case. We collapse the stencils of the three faces. Each face consider to have the same rigid body motion defined by the the face center DOF.

3. If the fine nodes lies on the coarse cell **face center**, due to that the four dependent nodes can be correctly interpolated from the surrounding coarse nodes, see Equation 5.22, we consider the nine edges, perpendicular to the faces, each has its own rigid body motion defined be the center node of the edge. Then, the same stencil collapsing method can be applied.

4. If the fine node lies on the coarse **cell center**, the expression for the prolongation operator is identical to Equation 5.15.

Prolongation of Rigid Body Motion

Lemma 5.3. *If the center collapsed stencil $\mathbf{S}_C$ is invertible, the prolongation operator created by Equation 5.30 can correctly prolongate rigid body motion.*

Proof. If $\mathbf{S}_C$ is invertible, Equation 5.29 is the unique solution to Equation 5.28. rigid body motion is a null energy mode. Therefore a rigid body motion will also satisfies Equation 5.28. Given that the solution is unique, solution provided by Equation 5.29 is rigid body motion, if $\mathbf{u}_L^f$ and $\mathbf{u}_R^f$ is prescribed with the same rigid body motion. □

5.9 Dirichlet Condition Coarsening

To coarsening Dirichlet conditions, we choice the geometric coarsening strategy introduced in Section 2.3, with some modifications. Dirichlet condition are imposed at cell granularity. At top level Dirichlet cells are marked *X Dirichlet*, *Y Dirichlet*, *Z Dirichlet*, or any combination of the three

based on which degrees of freedom are fixed. If a cell is marked Dirichlet, all of its nodes are marked Dirichlet as well.

For the coarse level, the additional rotational degrees of freedom also needs to be marked as Dirichlet. For 2D, if a cell is fixed, either in X or Y, it will not be able to rotate, therefore the rotational degrees of freedom at the coarse level will also be marked as Dirichlet. For 3D, if a cell is fixed in X dimension, the cell would not be able to rotate in Y and Z axis, but still allows to rotate in X axis. Therefore, in this case, only the rotational degrees of freedom associated with Y axis rotation and Z axis rotation are marked as Dirichlet, the X axis rotation remains free.

5.10 Choice of Smoother

In Section 5.7, a defect of the method was noted, that the coarse level, $\hat{e}_1$ may not be in Range($\mathbf{L}_{ff}^{(p)}$) for Theorem 5.1. This implies that for some nodes, the local convergence metric can be $K_{i,p} = \infty$. In such cases, a point based smoother may not sufficient. Therefore, for all the coarse levels, a colored box smoother of radius 2 is used. This problem does not exist at the finest level, therefore, an eight-colored Guass-Seidel smoother is used at the finest level that is the same as presented in Section 4.6.

Because of that our Dirichlet condition is geometrically coarsened, to ensure the multigrid is stable, we will need extra boundary smoothing iterations. The boundary nodes are defined as: within **3** stencil distance from a Dirichlet node. For box smoother, we use boundary cells, defined as: within **2** cell radius in either axis of a Dirichlet cell. When using this stencil aware multigrid scheme as a preconditioner for conjugate gradient, we found a standard 3-1-3 scheme: 3 boundary smoothing, 1 interior smoothing, and 3 boundary smoothing again, is enough to produce good convergence. In practice, we also found that for boundary smoothing, Gauss-Seidel smoother is sufficient. While for interior smoothing, box

smoother is needed for better convergence.

5.11 Stencil Aware Multigrid as Preconditioner

For better convergence behavior, instead of using the stencil aware multigrid as a standalone solver, we used a symmetric implementation of it to use it as a preconditioner for conjugate gradient.

To improve convergence and avoid null space issues in the local problems, we also choose to pad the domain at the finest level with cells with minimal stiffness (1e-9) , but only for the preconditioner, so that all finest cells contained within the coarse cells at the lowest level has a none-zero stiffness.

5.12 Solver Convergence Analysis

To test the solver convergence in comparison to standard multilinear interpolated multigrid, we conducted tests that some are synthetic and some are naturally emerged from the topology optimization process.

Two Level Convergence Test

The two level convergence test is conducted on a synthetic domain of size 32^3. Figure 5.6 is a 2D demonstration the the synthetic domain. Figure 5.7 shows the convergence rate of different methods for the synthetic domain. We can see that multi-linear multigrid (MLMG), convergence rate has drastically slowed down after 10 iterations, while stencil aware multigrid (SAMG) maintained the convergence rate of 0.5, i.e. the residual reduces by half every iteration. Using stencil aware multigrid as a preconditioner for conjugate gradient (SAMGPCG), can even further improves convergence rate.

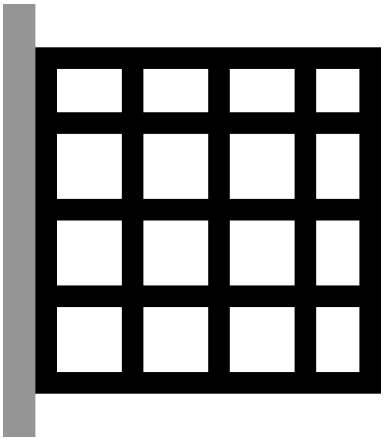


Figure 5.6: An illustration of the synthetic testing domain of 32^2 in 2D. The left side is fixed as Dirichlet condition. Black region are solid materials, the white space in between are soft material of 10^{-9} stiffness. Material properties are: $\mu = 100$ and $\lambda = 200$. A uniform downward force is applied to the right face.

Multi-level Convergence Test

The convergence tests conducted were aimed to support the following hypothesis:

1. The convergence rate of *multi-linear multigrid (MLMG)* preconditioned conjugate gradient combined with a *colored Gauss-Seidel smoother*, slows down with the increasing resolution as more complex topology emerges.

2. The convergence rate of *multi-linear multigrid (MLMG)* preconditioned conjugate gradient combined with a *colored box smoother*, slows down with the increasing resolution as more complex topology emerges.

3. The convergence rate of *stencil aware multigrid (SAMG)* preconditioned conjugate gradient combining with a *colored Gauss-Seidel smoother*,

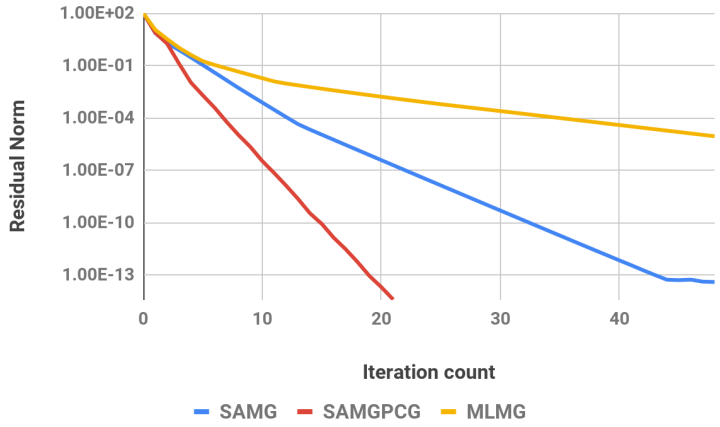


Figure 5.7: Two level convergence rate of multilinear multigrid(MLMG), stencil aware multigrid(SAMG), and stencil aware multigrid preconditioned conjugate gradient(SAMGPCG) on the 32^3 synthetic domain.

slows down with the increasing resolution as more complex topology emerges.

4. The convergence rate of *stencil aware multigrid (SAMG)* preconditioned conjugate gradient combining with a *colored box smoother* is more consistent across different resolutions.

The test is conducted using the bird beak topology optimization example, Figure 4.1, at resolution $500\times400\times300$, $750\times600\times450$, and $1000\times800\times600$. The numbers of active voxels are 5M, 14M, and 40M. The last iteration of topology optimization is used.

Resolution Scale Test

Figure 5.8, top left plot, illustrates is the convergence rate of the MLMGPCG proposed in Chapter 4. The convergence rate of the algorithm decays with the increasing of the resolution. For 5M active voxels, the algorithm takes 120 iterations to reduce the initial residual by 5 orders of magnitude. While

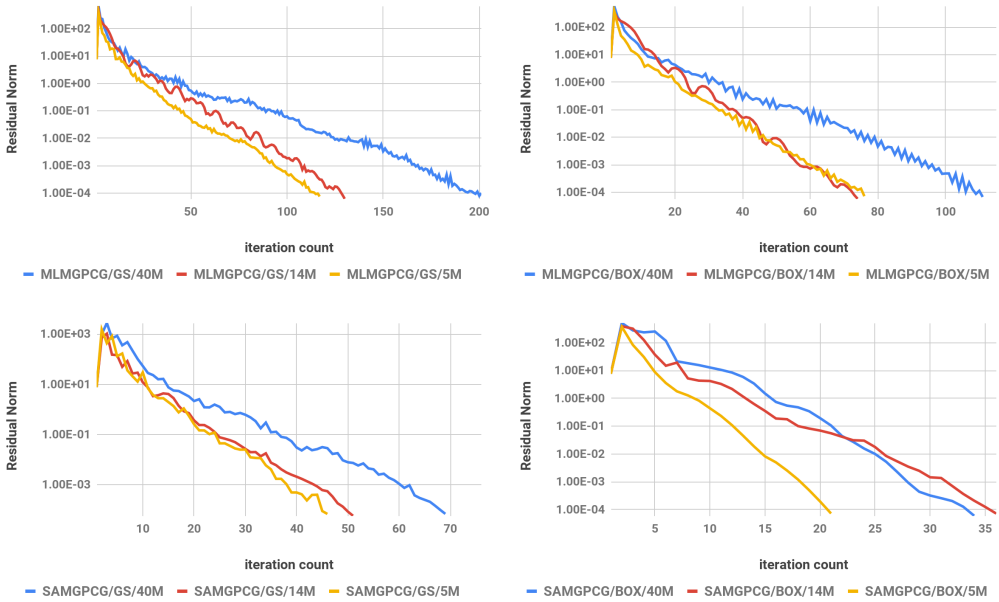


Figure 5.8: Convergence plot for the bird beak example. In all cases multigrid hierarchy contains 4 levels, and 3-1-3 smoothing iterations were applied. Top left: MLMGPCG with Gauss-Seidel smoother with 40M/14M/5M active voxels.Top right: MLMGPCG with box smoother with 40M/14M/5M active voxels.Top left: SAMGPCG with Gauss-Seidel smoother with 40M/14M/5M active voxels.Top right: SAMGPCG with box smoother with 40M/14M/5M active voxels.

with 40M active voxels, it takes about 200 iterations to reduce the initial residual by 5 orders of magnitude.

Figure 5.8, top right plot, shows that by using a box smoother, the total number of iterations for convergence has reduced by 30%. But with increment of resolution, we still observes a slow down in convergence.

Figure 5.8, bottom left plot, using the the SAMGPCG, even with just Gauss-Seidel smoothing, the highest resolution example was able to converge in 70 iterations with the 40M example. But the convergence rate has slowed down from 14M to 40M.

Figure 5.8, bottom right plot, using the the SAMGPCG, with a box

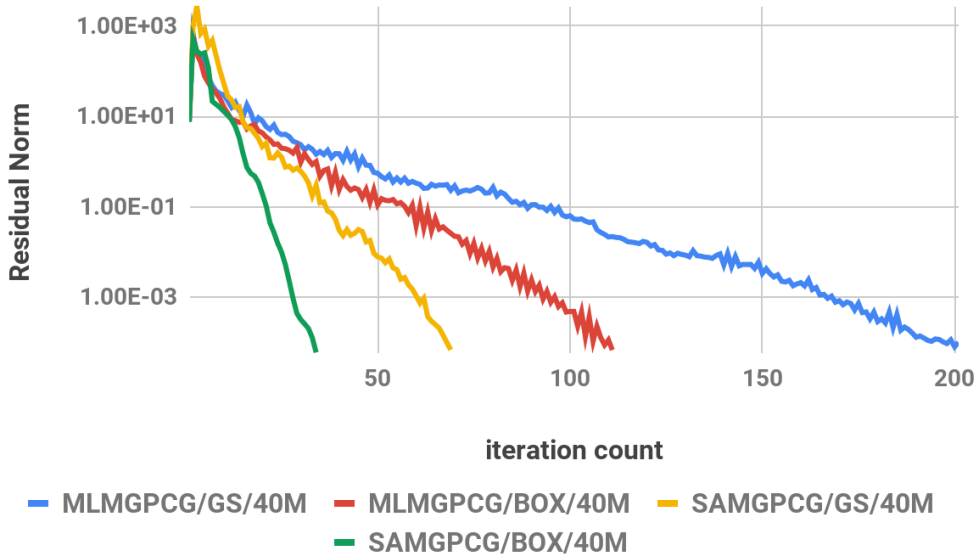


Figure 5.9: Convergence plot for the 40M bird beak example using the 4 different methods.

smoother, the 40M example and 14M example both converged in approximately 30 iterations. The 5M example converged in 22 iterations.

Conclusion

Our new multigrid solver, stencil aware multigrid preconditioned conjugate gradient(SAMGPCG) has proven to be able to drastically improve convergence rate for complex domains, Figure 5.9. But due to the memory constraint, it is difficult to test larger examples given the total amount of memory on our testing system, while small examples are unable to represent the complexity of the domain. It remains to be further validate that SAMGPCG can maintain its convergence rate with resolution scaling.

5.13 Limitations and Future Work

Though the solver presented in this chapter, SAMGPCG, has proven to have an exceptionally fast convergence rate, it requires significant more memory in comparison to standard multi-linear MGPCG, as it requires explicit storage of not only the coarse grid matrix but also the transfer operator. In practice, SAMGPCG has 3 times larger memory footprint in comparison to multi-linear MGPCG.

Further more, the derivation of the rotational degrees of freedom in this work is restricted to Cartesian grid discretization. It is possible to extend the method to compensate irregular mesh, but the interpretation of the rotational degrees of freedom will be much different. How this method applies to general mesh discretization requires further investigation.

6 DISCUSSION

In this last chapter, I will conclude this document by revisiting the challenges of designing numerical solvers in the context of the modern hardware, and how these challenges have inspired this thesis work.

6.1 Modern Hardware Features

The direction at which modern hardware advances has shifted. The CPU frequency scaling has slowed down, while other features, such as SIMD and accelerated hardware, have been expanded. The works presented in the thesis were inspired by the following three key features:

- **Increased number of computation cores** The scaling of CPU core speed in recent years has slowed down, while more computational cores were added to single CPUs. AMD Ryzen Threadripper series feature CPUs with maximally 32 cores, and Intel Skylake-SP series feature CPUs with 28 cores.

- **Wider SIMD Width** Same Instruction Multiple Data (SIMD) or vectorization, is the method of increasing the computation throughput without increasing the core count. GPUs usually feature a vector width of 32. CPUs, on the other hand, have evolved from SSE (vector width of 4), to AVX (vector width of 8), and recently AVX-512 (vector width of 16).

- **Heterogeneous Memory Hierarchy** In recent years, deeper memory hierarchies, such as L3 caches, were introduced to compensate for the gap between memory latency and CPU speed. There are also configurations, such as non-uniform memory access (NUMA) or multi-GPU compute platforms, that feature more computational

power in terms of FLOPS and aggregate memory bandwidth. But not all memory can be accessed with the same speed by the different compute units.

6.2 Program Design Consideration

Inspired by the hardware features, the solvers presented in this thesis are influenced by the following design considerations:

- **Parallel Computing** Designing algorithms that minimize read and write hazards is the first step in utilizing modern hardware. For a well designed compute bound parallel algorithm, its performance can scale linearly with the number of cores, for it is difficult to saturate the memory bandwidth without vectorization on CPUs.

- **Vectorization** Vectorization utilizes the SIMD feature in modern hardware. It drastically increases the number of floating point operations performed per second. For algorithms that are compute bound, for instance the matrix-free Galerkin coarsening introduced in Chapter 4, vectorization can greatly boost performance.

- **Heterogeneous Computing** To best utilize the full computational power of a heterogeneous platform the algorithms, especially memory bound operations, need to be made aware of the different levels of the memory hierarchy, and the fact that not all can be accessed at the same rate. This is done so that performance will not be bounded by the slowest data path in the hierarchy. In Chapter 3, this was achieved by increasing the localized work load and reducing the communication need between the heterogeneous components.

6.3 Tuning Numerical Elliptic PDE Solvers for Modern Hardware

Given these design considerations, this thesis has presented three solvers that are tailored to the new programming paradigms. Now we will state in detail, the design process of those solvers.

Analyze Algorithmic Performance Bound

When designing numerical algorithms, it always requires reading some amount of data, doing some amount work, then write something back. The ratio between the the amount of data to read/write, and the computations operated on the data dictates the bottleneck of the algorithm on a given platform, given that the data can be reasonably cached. As an example, a Skylake-SP Gold 6140 system, using AVX-512 instructions, has about 2.3 TFLOPS computation power and 128GB/s memory bandwidth. This creates a ratio of 67 FLOPS/Float. It means that we can perform 67 floating point operations for each float we read, before saturating the memory bandwidth. Without vectorization, this ratio is reduced to 5 FLOPS/Float (the CPU runs at a higher clock rate without vectorization). For GPUs, GTX 1080ti for instance, features 11.3 TFLOPs and 484 GB/s memory bandwidth. That creates a ratio of 93 FLOPS/Float. This ratio provides a good guidance in optimization as it not only indicates the upper bound of the best performance that can be achieved on a platform, but also helps us to understand when an algorithm is memory bound or compute bound.

For the Poisson problem targeted in Chapter 3, the Jacobi smoother, requires 7 fused multiply adds (FMA) for one float `read` and one float `write`. This operation is clearly bound by memory bandwidth therefore, even without vectorization, Setaluri et al. (2014) have achieved very good performance. As for the linear elasticity problem in Chapter 4, the multiply kernel requires 1152 fused multiply adds for three float reads and three

float writes. This high ratio between computation and memory access implies that this kernel is compute bound and using vectorization can drastically increase the performance.

The ideal computation power of a platform gives the theoretical upper bound of the best performance an algorithm can hope to achieve. But in reality, many other aspects will influence the final performance of the algorithm. For instance, we assumed perfect caching in our analysis. In reality, caching behavior depends on the memory access pattern and the data layout. It is extremely difficult to achieve a perfect caching performance. Another example of factors that can limit the algorithms from achieving ideal performance is latency. A modern Intel CPU can issue two FMA per cycle, and yet FMA will take 5 cycles to finish. If one of the FMA instructions has dependency on the results that is within 12 instruction before, the execution would stall to wait for result. Another source of latency is when data is not immediately available from L1 cache, the execution will stall to wait for the data to be fetched, either from L2 or L3 cache, or from the main memory. For GPUs, they can hide latency through massive multi-threading (up to 32 warps a compute unit (SM), each warp is a 32 wide SIMD vector). While for CPUs, hyper-threading can levitate this issue a little, but it will also increase the pressure on the caches. Instead of hiding latency, optimization for CPU prefers reducing latency through compiler optimization that reduces instruction dependency, out of order issuing that reduces instruction stall, and pre-fetching that predicts and fetch data into cache before it is requested.

Although the theoretical upper bound of a platform is a unrealistic goal, it can be used as a very good metric when evaluating the performance of a numerical kernel. It provides a good confidence that a kernel is very close to optimal, as it was used in Chapter 3 and Chapter 4.

Choosing Data Structures

The choice of data structure can not only influence the complexity of an algorithm. When tailored towards hardware, it can also influence cache performance, instruction latency, and many other aspects of optimization. In this thesis, SPGrid has been chosen as the primary data structure for the following reasons:

1. SPGrid can represent sparse and adaptive geometry. For large scale simulations, in many cases, the simulation domain is only filled with a small fraction of materials. Sparse data structure allows higher resolution by eliminating the need for storing inactive regions.

2. In comparison to other sparse data structure, OpenVDB for instance, SPGrid does not require maintain a tree structure that needs to be traversed when accessing the data. Instead, SPGrid utilized hardware translation lookaside buffer (TLB) for data lookup. This feature made SPGrid data structure more friendly to SIMD optimization and CPU pre-fetcher.

3. SPGrid organize data into Morton encoded blocks, which ensures that if data is geometrically close, they are more likely to be close in memory as well. It helps improving cache hit rate when using stencil operations that only requires data from geometric neighbors.

These features of SPGrid have enabled the optimization and algorithmic design presented throughout the thesis.

Vectorizing Computation Kernels

Vectorization is an effective way to achieve higher computation throughput on modern hardware. GPU programs require to be designed with full SIMD support. While for CPUs or Xeon Phi processors, they allow mixture

of SIMD instructions and scalar instructions. This flexibility makes it a lot easier to adopt existing algorithms using SIMD instructions on CPUs than on GPUs. To achieve a good performance gain with SIMD, developers need to be aware of the data layout and its access pattern.

Aligned Memory Access SIMD instructions operate on vectors of data. As hardware always reads data in the unit of cache lines. If the data of the same vector is aligned and consecutive in memory, loading a vector from memory is trivia. When the data is scattered, gathering of a vector can be rather expensive. In Section 4.6, we rearranged the data layout for the colored Gauss-Seidel smoother to compensate the data alignment for this reason.

Minimizing Memory Access SIMD instructions, in general, require more data for operation than scalar instructions. This can often cause more pressure on the cache. For memory bound operations, good performance relies on good caching behavior. Therefore, it is important not to increase cache pressure when using SIMD. For this reason, in Section 4.6, we designed the Galerkin Coarsening algorithm by using a SIMD vector representing the 8 nodal degrees of freedom instead of coarsening 8 cells at ones.

With the considerations above, choosing a data structure that can efficiently support SIMD operations is essential to optimization. In Chapter 4, we expended SPGrid to support larger blocks for better memory alignment, and added *VectorGet* operations to avoid the need for gathering instructions.

Algorithm for Multi-Accelerator Equipped Platform

Having multiple accelerators is an easy way to increase the performance throughput of a system. But accessing memories from other accelerators and the main system memory from a accelerator is slow and has a higher

latency. In many cases, it also requires to copy the data to the accelerator's local memory first. Even with the promotion of NVLink, which features 80GB/s to 150GB/s shared bandwidth across GPUs, it is still significantly slower than the GPU's local memory, which can feature over 500GB/s bandwidth. When designing algorithms for a heterogeneous system, minimizing communication between accelerators is the key to avoid been limited by the low bandwidth and high latency of the communication channel.

In Chapter 3, we presented a divide-and-conquer method that divides the simulation domain into isolated pieces. It allows the accelerators to solve each domain separately without the need for communication. Then an interface solve of reduced problem sized is used to merge the results from the subdomain solvers. By diminishing the need of constant communication, our solver has proven to be effective both in terms of increased convergence rate and improved performance.

6.4 Challenges of Large Scale Simulation

This thesis has presented techniques that enable large scale simulation on a single work station. It is also the first time that we are able to easily examine how resolution may influence the solver performance at a large scale. As the domain complexity increases, we observed the degradation of the solver convergence rate, multigrid method specifically, due to two main reasons: numerical cancellation, and poor coarse grid representation that were constructed from an inappropriate prolongation operator. Our numerical solvers aim to solve a discretized partial differential equation. At heart, the discrete operator is a differential operator. For large scale simulation, as the discretized solution converges to the continuous solution, the difference between the displacements of each adjacent samples becomes smaller. With a second order accurate discretization, the difference

between samples reduces at $O(h^2)$ speed, while the force at each sample point reduces at $O(h)$ speed. h here is the cell length. This discrepancy between force and displacement demands higher numerical precision with increased simulation resolution. Second consideration, with the resolution scaling, it is important for the numerical solver to maintain an approximately linear complexity. Standard multilinear based multigrid method is proven a linear complexity solver only for cubic domains. At higher resolution, thinner features, varying material distribution, and more complex domains impose significant challenges to the convergence of standard multigrid method.

Mixed Precision Solver Higher floating point precision can be used to mitigate the issue of numerical cancellation. For our application and targeted size, double precision has proven to be sufficient for reducing residual by 4 orders of magnitude. But double precision computation requires twice the memory footprint as well as twice the computation time. We developed a mix precision scheme that achieved the same accuracy as using double precision but only a fraction of the variables were stored in double.

Improved Multigrid Method To overcome the convergence issue of the standard multigrid method, we developed a stencil aware multigrid method. This multigrid method leverage a local smoothness metric to maximize the quality of coarse grid correction and drastically improved convergence for complex domains.

6.5 Limitations and Future Work

Unified Computational Framework

This thesis presents methods that specifically tailored to utilizing modern hardware solving elliptic PDEs. It has demonstrated the potential of the compute platforms and provided some guidelines regarding how to design efficient numerical algorithms for those computational platforms. But to achieve the performance we have demonstrated, we have to perform very detailed optimization that is specially optimized for each platform. There has been efforts in recent years to use a unified memory to abstract out the memory hierarchy from the programmer, such as CUDA unified memory and XEON Phi kight's landing series HBM memory in cache mode. Though that those design are convenient and provide reasonable performance for some algorithms. It is difficult to achieve close to hardware optimal in the general cases. The question of whether there is a generic framework that is capable of utilizing the modern hardware without the need of special optimization remains an open problem.

Scaling to Cluster

In work by Aage et al. (2017), 8000 cores were utilized to solve a giga-voxel topology optimization problem. In this thesis, we demonstrated that we can solve the same amount of degrees of freedom in a single machine. When scaling the solver to a cluster, the algorithm needs to take into the consideration that remote memory is very expensive to access. For I/O bound algorithms or compute bound algorithms, it is less of an issue, but designing numerical solvers that is memory bound in many cases for cluster, without significantly compromising performance remains a challenge.

Numerical Stability

For the problems we have targeted, double precision was sufficient. But with the increasing of scale, there will be a point at which double precision will be proved to be insufficient. Deriving a numerically stable discretization and algorithm can be important in the future.

Second Order Accurate Discretization

The work presented here were specially designed for Cartesian grid discretization benefiting from its regularity. Such regularity was exploited for our efficient SIMD implementation and construction of the stencil aware multigrid hierarchy. The major drawback of a Cartesian grid discretization is that it often fails to capture the domain boundaries to second order accuracy. The most common way to achieve this accuracy is using conforming mesh, that introduces highly irregular pointer based data structures that are unfriendly to modern hardware. But there has been works that achieved second order accuracy on a Cartesian grid. Zhu et al. (2012) used a virtual node method that captured the second order accuracy along domain boundaries on a Cartesian grid discretization. Aanjaneya et al. (2017) achieved second order accuracy around free surface boundaries by utilizing a power diagram. The numerical solvers presented in these works feature a first order preconditioner paired with an second order accurate operator. But all those methods introduce auxiliary data structures that breaks the regularity of the discretization. How to incorporate higher order discretization into the modern age of the hardware evolution still remains an open question.

REFERENCES

Aage, Niels, Erik Andreassen, Boyan S. Lazarov, and Ole Sigmund. 2017. Giga-voxel computational morphogenesis for structural design. *Nature* 550 7674:84–86.

Aage, Niels, Erik Andreassen, and Boyan Stefanov Lazarov. 2015. Topology optimization using petsc: An easy-to-use, fully parallel, open source topology optimization framework. *Structural and Multidisciplinary Optimization* 51(3):565–572.

Aanjaneya, Mridul, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. 2017. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Transactions on Graphics (TOG)* 36(4): 140.

Adalsteinsson, D, and J.A Sethian. 1999. The fast construction of extension velocities in level set methods. *Journal of Computational Physics* 148(1):2 – 22.

Adams, Bart, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. 2007. Adaptively sampled particle fluids. In *Acm siggraph 2007 papers*. SIGGRAPH '07, New York, NY, USA: ACM.

Ament, Marco, Günter Knittel, Danviel Weiskopf, and Wolfgang Straßer. 2010. A parallel preconditioned conjugate gradient solver for the poisson problem on a multi-gpu platform. In *Proceedings of the 18th euromicro conference on parallel, distributed and network-based processing*, 583–592.

Ando, Ryoichi, Nils Thuerey, and Chris Wojtan. 2015a. A stream function solver for liquid simulations. *ACM Trans. Graph.* 34(4):53:1–53:9.

Ando, Ryoichi, Nils Thürey, and Chris Wojtan. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph.* 32(4):103:1–103:10.

Ando, Ryoichi, Nils Thürey, and Chris Wojtan. 2015b. A dimension-reduced pressure solver for liquid simulations. *EUROGRAPHICS 2015*.

Bender, Jan, and Dan Koschier. 2015. Divergence-free smoothed particle hydrodynamics. 147–155. SCA '15.

Brandt, Achi. 1977. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation* 31(138):333–390.

———. 1986. Algebraic multigrid theory: The symmetric case. *Applied mathematics and computation* 19(1-4):23–56.

Brezina, Marian, Andrew J Cleary, Robert D Falgout, Van Enden Henson, Jim E Jones, Thomas A Manteuffel, Stephen F McCormick, and John W Ruge. 2001. Algebraic multigrid based on element interpolation (amge). *SIAM Journal on Scientific Computing* 22(5):1570–1592.

Bro-nielsen, Morten, and Stephane Cotin. 1996. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In *Computer graphics forum*, 57–66.

Brun, Emmanuel, Arthur Guittet, and Frédéric Gibou. 2012. A local level-set method using a hash table data structure. *Journal of Computational Physics* 231(6):2528–2536.

Challis, Vivien J, Anthony P Roberts, and Joseph F Grotowski. 2014. High resolution topology optimization using graphics processing units (gpus). *Structural and Multidisciplinary Optimization* 49(2):315–325.

Chen, Zhili, Byungmoon Kim, Daichi Ito, and Huamin Wang. 2015. Wetbrush: Gpu-based 3d painting simulation at the bristle level. *ACM Trans. Graph.* 34(6):200:1–200:11.

Chentanez, Nuttapong, Bryan E. Feldman, François Labelle, James F. O'Brien, and Jonathan R. Shewchuk. 2007. Liquid simulation on lattice-based tetrahedral meshes. 219–228. SCA '07, Switzerland.

Chentanez, Nuttapong, and Matthias Müller. 2011. Real-time Eulerian water simulation using a restricted tall cell grid. 82:1–82:10. SIGGRAPH '11.

Christiansen, Asger Nyman, J Andreas Bærentzen, Morten Nobel-Jørgensen, Niels Aage, and Ole Sigmund. 2015. Combined shape and topology optimization of 3d structures. *Computers & Graphics* 46:25–35.

Christiansen, Asger Nyman, Morten Nobel-Jørgensen, Niels Aage, Ole Sigmund, and Jakob Andreas Bærentzen. 2014. Topology optimization using an explicit interface representation. *Structural and Multidisciplinary Optimization* 49(3):387–399.

Cohen, Jonathan, Sarah Tariq, and Simon Green. 2010. Interactive fluid-particle simulation using translating Eulerian grids. In *Acm siggraph symp. on interactive 3d graphics and games*, 15–22.

Da, Fang, Christopher Batty, Chris Wojtan, and Eitan Grinspun. 2015. Double bubbles sans toil and trouble: Discrete circulation-preserving vortex sheets for soap films and foams. *ACM Trans. Graph.* 34(4):149:1–149:9.

Deaton, Joshua D, and Ramana V Grandhi. 2014. A survey of structural and multidisciplinary continuum topology optimization: post 2000. *Structural and Multidisciplinary Optimization* 49(1):1–38.

Dendy, JE. 1982. Black box multigrid. *Journal of Computational Physics* 48(3):366–386.

Dick, Christian, Joachim Georgii, and Rüdiger Westermann. 2011a. A real-time multigrid finite hexahedra method for elasticity simulation using cuda. *Simulation Modelling Practice and Theory* 19(2).

Dick, Christian, Joachim Georgii, and RÃ¼diger Westermann. 2011b. A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simulation Modelling Practice and Theory* 19(2):801 – 816.

Dick, Christian, Marcus Rogowsky, and Rüdiger Westermann. 2016. Solving the fluid pressure poisson equation using multigrid – evaluation and improvements. *IEEE Trans. Visualization & Computer Graphics*.

Dobashi, Yoshinori, Yasuhiro Matsuda, Tsuyoshi Yamamoto, and Tomoyuki Nishita. 2008. A fast simulation method using overlapping grids for interactions between smoke and rigid objects. *Computer Graphics Forum* 27(2):477–486.

Dohrmann, Clark R. 2007. Interpolation operators for algebraic multigrid by local optimization. *SIAM Journal on Scientific Computing* 29(5):2045–2058.

Donea, Jean, S Giuliani, and Jean-Pierre Halleux. 1982. An arbitrary lagrangian-eulerian finite element method for transient dynamic fluid-structure interactions. *Computer methods in applied mechanics and engineering* 33(1-3):689–723.

Edwards, Essex, and Robert Bridson. 2015. The discretely-discontinuous Galerkin coarse grid for domain decomposition. *CoRR* abs/1504.00907.

English, R. Elliot, Linhai Qiu, Yue Yu, and Ronald Fedkiw. 2013a. Chimera grids for water simulation. 85–94. SCA '13.

English, R Elliot, Linhai Qiu, Yue Yu, and Ronald Fedkiw. 2013b. Chimera grids for water simulation. In *Proceedings of the 12th acm siggraph/eurographics symposium on computer animation*, 85–94. ACM.

Enright, D., D. Nguyen, F. Gibou, and R. Fedkiw. 2003. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *Proc. 4th asme-jsme joint fluids eng. conf.*

Enright, Douglas, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. 2002a. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics* 183(1):83–116.

Enright, Douglas, Frank Losasso, and Ronald Fedkiw. 2005. A fast and accurate semi-Lagrangian particle level set method. *Comput. Struct.* 83(6-7):479–490.

Enright, Douglas, Stephen Marschner, and Ronald Fedkiw. 2002b. Animation and rendering of complex water surfaces. *ACM Trans. Graph.* 21(3):736–744.

Eschenauer, H. A., V. V. Kobelev, and A. Schumacher. 1994. Bubble method for topology and shape optimization of structures. *Structural optimization* 8(1).

Ferstl, Florian, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. 2016. Narrow band flip for liquid simulations. In *Computer graphics forum*, vol. 35, 225–232. Wiley Online Library.

Ferstl, Florian, Rüdiger Westermann, and Christian Dick. 2014a. Large-scale liquid simulation on adaptive hexahedral grids. *IEEE Trans. Visualization & Computer Graphics* 20(10):1405–1417.

Ferstl, Florian, Rüdiger Westermann, and Christian Dick. 2014b. Large-scale liquid simulation on adaptive hexahedral grids. *IEEE transactions on visualization and computer graphics* 20(10):1405–1417.

Foster, N., and R. Fedkiw. 2001. Practical animation of liquids. In *Proc. of acm siggraph 2001*, 23–30.

Foster, Nick, and Dimitri Metaxas. 1996. Realistic animation of liquids. *Graph. Models Image Process.* 58(5):471–483.

Frisken, Sarah F., Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on computer graphics and interactive techniques*, 249–254. SIGGRAPH '00.

Gao, Ming, Nathan Mitchell, and Eftychios Sifakis. 2014. Steklov-poincaré skinning. 139–148. SCA '14.

de Goes, Fernando, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. Power particles: An incompressible fluid solver based on power diagrams. *ACM Trans. Graph.* 34(4):50:1–50:11.

Golas, Abhinav, Rahul Narain, Jason Sewall, Pavel Krajcevski, Pradeep Dubey, and Ming Lin. 2012. Large-scale fluid simulation using velocity-vorticity domain decomposition. *ACM Trans. Graph.* 31(6):148:1–148:9.

Griebel, Michael, Daniel Oeltz, and Marc Alexander Schweitzer. 2003. An algebraic multigrid method for linear elasticity. *SIAM Journal on Scientific Computing* 25(2):385–407.

Hecht, Florian, Yeon Jin Lee, Jonathan R. Shewchuk, and James F. O'Brien. 2012. Updated sparse cholesky factors for corotational elastodynamics. *ACM Trans. Graph.* 31(5):123:1–123:13.

Henderson, Ronald D. 2012. Scalable fluid simulation in linear time on shared memory multiprocessors. In *Proceedings of the digital production symposium*, 43–52. ACM.

Henson, Van Emden, and Panayot S Vassilevski. 2001. Element-free amge: General algorithms for computing interpolation weights in amg. *SIAM Journal on Scientific Computing* 23(2):629–650.

Houston, Ben, Michael B. Nielsen, Christopher Batty, Ola Nilsson, and Ken Museth. 2006. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Trans. Graph.* 25(1):151–175.

Ihmsen, Markus, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. 2014. Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics* 20(3):426–435.

Irving, Geoffrey, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. 2006. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. 805–811. SIGGRAPH '06.

Jiang, Chenfanfu, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The affine particle-in-cell method. *ACM Trans. Graph.* 34(4):51:1–51:10.

Jung, Hwi-Ryong, Sun-Tae Kim, Junyong Noh, and Jeong-Mo Hong. 2013. A heterogeneous CPU-GPU parallel approach to a multigrid poisson solver for incompressible fluid simulation. *Computer Animation and Virtual Worlds* 24(3-4):185–193.

Klingner, Bryan, Bryan Feldman, Nuttapong Chentanez, and James O'Brien. 2006. Fluid animation with dynamic meshes. 820–825. SIGGRAPH '06.

Ladický, L'ubor, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.* 34(6):199:1–199:9.

Lentine, Michael, Wen Zheng, and Ronald Fedkiw. 2010. A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Trans. Graph.* 29(4):114:1–114:9.

Liu, Beibei, Gemma Mason, Julian Hodgson, Yiying Tong, and Mathieu Desbrun. 2015. Model-reduced variational fluid simulation. *ACM Trans. Graph.* 34(6):244:1–244:12.

Liu, Haixiang, Nathan Mitchell, Mridul Aanjaneya, and Eftychios Sifakis. 2016. A scalable schur-complement fluids solver for heterogeneous compute platforms. *ACM Transactions on Graphics (TOG)* 35(6):201.

Long, Benjamin, and Erik Reinhard. 2009. Real-time fluid simulation using discrete sine/cosine transforms. In *Proceedings of the 2009 symposium on interactive 3d graphics and games*, 99–106. I3D '09.

Losasso, Frank, Ronald Fedkiw, and Stanley Osher. 2005. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids* 35:2006.

Losasso, Frank, Frédéric Gibou, and Ron Fedkiw. 2004. Simulating water and smoke with an octree data structure. In *Acm transactions on graphics (tog)*, vol. 23, 457–462. ACM.

Losasso, Frank, Jerry Talton, Nipun Kwatra, and Ronald Fedkiw. 2008. Two-way coupled SPH and particle level set fluid simulation. *IEEE Transactions on Visualization and Computer Graphics* 14(4):797–804.

Macklin, Miles, and Matthias Müller. 2013. Position based fluids. *ACM Trans. Graph.* 32(4):104:1–104:12.

McAdams, Aleka, Eftychios Sifakis, and Joseph Teran. 2010. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 acm siggraph/eurographics symposium on computer animation*, 65–74. Eurographics Association.

McAdams, Aleka, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Transactions on Graphics (TOG)* 30(4):37.

McCormick, SF. 1984. Multigrid methods for variational problems: further results. *SIAM journal on numerical analysis* 21(2):255–263.

———. 1985. Multigrid methods for variational problems: general theory for the v-cycle. *SIAM Journal on Numerical Analysis* 22(4):634–643.

McCormick, SF, and JW Ruge. 1982. Multigrid methods for variational problems. *SIAM Journal on Numerical Analysis* 19(5):924–929.

Molemaker, Jeroen, Jonathan M. Cohen, Sanjit Patel, and Jonyong Noh. 2008. Low viscosity flow simulations for animation. In *Proceedings of the 2008 acm siggraph/eurographics symposium on computer animation*, 9–18. SCA '08, Aire-la-Ville, Switzerland: Eurographics Association.

Moulton, J David, Joel E Dendy Jr, and James M Hyman. 1998. The black box multigrid numerical homogenization algorithm. *Journal of Computational Physics* 142(1):80–108.

Museth, Ken. 2013. Vdb: High-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics (TOG)* 32(3):27.

Nocedal, Jorge, and Stephen J Wright. 2006. Conjugate gradient methods. *Numerical optimization* 101–134.

Nvidia, CUDA. 2014. Cusparse library. *NVIDIA Corporation, Santa Clara, California*.

Quarteroni, A., and A. Valli. 1999. *Domain decomposition methods for partial differential equations*, vol. 10. Clarendon Press.

Raveendran, Karthik, Nils Thuerey, Chris Wojtan, and Greg Turk. 2012. Controlling liquids using meshes. 255–264. SCA '12.

Raveendran, Karthik, Chris Wojtan, and Greg Turk. 2011. Hybrid smoothed particle hydrodynamics. 33–42. SCA '11.

Reisch, Jon, Stephen Marshall, Magnus Wrenninge, Tolga Göktekin, Michael Hall, Michael O'Brien, Jason Johnston, Jordan Rempel, and Andy Lin. 2016. Simulating rivers in the good dinosaur. In *Acm siggraph 2016 talks*, 40. ACM.

Rozvany, George IN. 2009. A critical review of established methods of structural topology optimization. *Structural and multidisciplinary optimization* 37(3):217–237.

Saad, Youcef, and Martin H Schultz. 1986. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing* 7(3):856–869.

Schmidt, Stephan, and Volker Schulz. 2011. A 2589 line topology optimization code written for the graphics card. *Computing and Visualization in Science* 1–8.

Selle, Andrew, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. 2008. An unconditionally stable MacCormack method. *J. Sci. Comput.* 35(2-3):350–371.

Setaluri, Rajsekhar, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. Spgrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Transactions on Graphics (TOG)* 33(6):205.

Sethian, James A, and Peter Smereka. 2003. Level set methods for fluid interfaces. *Annual review of fluid mechanics* 35(1):341–372.

Sigmund, Ole, and Kurt Maute. 2013. Topology optimization approaches. *Structural and Multidisciplinary Optimization* 48(6):1031–1055.

Sigmund, Ole, and S Torquato. 1999. Design of smart composite materials using topology optimization. *Smart Materials and Structures* 8(3):365.

Smith, Barry F., Petter E. Bjørstad, and William D. Gropp. 1996. *Domain decomposition: Parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press.

Sokolowski, J., and A. Zochowski. 1999. On the topological derivative in shape optimization. *SIAM Journal on Control and Optimization* 37(4).

Solenthaler, Barbara, and Markus Gross. 2011. Two-scale particle simulation. 81:1–81:8. SIGGRAPH '11.

Stam, Jos. 1999. Stable fluids. In *Proceedings of the 26th annual conference on computer graphics and interactive techniques*, 121–128. ACM Press / Addison-Wesley Publishing Co.

———. 2002. A simple fluid solver based on the FFT. *J. Graph. Tools* 6(2): 43–52.

Tan, Jie, Xubo Yang, Xin Zhao, and Zhanxin Yang. 2008. Fluid animation with multi-layer grids. In *Sca '08 posters*.

Teng, Yun, Mark Meyer, Tony DeRose, and Theodore Kim. 2015. Subspace condensation: Full space adaptivity for subspace deformations. *ACM Trans. Graph.* 34(4):76:1–76:9.

Thürey, Nils, Chris Wojtan, Markus Gross, and Greg Turk. 2010. A multiscale approach to mesh-based surface tension flows. *ACM Trans. Graph.* 29(4):48:1–48:10.

Trottenberg, Ulrich, Cornelius W. Oosterlee, and Anton Schuller. 2001. *Multigrid*. Academic Press.

Vaněk, Petr, Jan Mandel, and Marian Brezina. 1996. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing* 56(3):179–196.

Wadbro, Eddie, and Martin Berggren. 2009. Megapixel topology optimization on a graphics processing unit. *SIAM review* 51(4):707–721.

Wojtan, Chris, Nils Thürey, Markus Gross, and Greg Turk. 2010. Physics-inspired topology changes for thin fluid features. *ACM Trans. Graph.* 29(4):1–8.

Wu, Jun, Christian Dick, and Rudiger Westermann. 2016a. A system for high-resolution topology optimization. *IEEE Transactions on Visualization and Computer Graphics* 22(3):1195–1208.

Wu, Jun, Christian Dick, and Rüdiger Westermann. 2016b. A system for high-resolution topology optimization. *IEEE transactions on visualization and computer graphics* 22(3):1195–1208.

———. 2016c. A system for high-resolution topology optimization. *IEEE transactions on visualization and computer graphics* 22(3):1195–1208.

Wu, Xiaofeng, Rajaditya Mukherjee, and Huamin Wang. 2015. A unified approach for subspace simulation of deformable bodies in multiple domains. *ACM Trans. Graph.* 34(6):241:1–241:9.

Yadav, Praveen, and Krishnan Suresh. 2014. Large scale finite element analysis via assembly-free deflated conjugate gradient. *Journal of Computing and Information Science in Engineering* 14(4):041008.

Yang, Ulrike Meier, et al. 2002. Boomeramg: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics* 41(1):155–177.

Zhang, Xinxin, and Robert Bridson. 2014. A PPPM fast summation method for fluids and beyond. *ACM Trans. Graph.* 33(6):206:1–206:11.

Zheng, Wen, Bo Zhu, Byungmoon Kim, and Ronald Fedkiw. 2015. A new incompressibility discretization for a hybrid particle mac grid representation with surface tension. *Journal of Computational Physics* 280: 96–142.

Zhu, Bo, Wenlong Lu, Matthew Cong, Byungmoon Kim, and Ronald Fedkiw. 2013. A new grid structure for domain extension. *ACM Transactions on Graphics (TOG)* 32(4):63.

Zhu, Bo, Ed Quigley, Matthew Cong, Justin Solomon, and Ronald Fedkiw. 2014. Codimensional surface tension flow on simplicial complexes. *ACM Trans. Graph.* 33(4):111:1–111:11.

Zhu, Bo, Xubo Yang, and Ye Fan. 2010a. Creating and Preserving Vortical Details in SPH Fluid. *Computer Graphics Forum*.

Zhu, Yongning, and Robert Bridson. 2005. Animating sand as a fluid. 965–972. SIGGRAPH '05.

Zhu, Yongning, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010b. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Transactions on Graphics (TOG)* 29(2):16.

Zhu, Yongning, Yuting Wang, Jeffrey Hellrung, Alejandro Cantarero, Eftychios Sifakis, and Joseph M Teran. 2012. A second-order virtual node algorithm for nearly incompressible linear elasticity in irregular domains. *Journal of Computational Physics* 231(21):7092–7117.