IMPLEMENTATION AND ANALYSIS OF A PARALLEL VERTEX-CENTERED FINITE ELEMENT SEGMENTAL REFINEMENT MULTIGRID SOLVER

A Thesis
Presented to
The Academic Faculty

by

Stefan Klaus Wilhelm Henneking

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in
Computational Science and Engineering

School of Computational Science and Engineering
Georgia Institute of Technology
May 2016

IMPLEMENTATION AND ANALYSIS OF A PARALLEL VERTEX-CENTERED FINITE ELEMENT SEGMENTAL REFINEMENT MULTIGRID SOLVER

Approved by:

Professor Richard Vuduc, Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

Professor Hao-Min Zhou
School of Mathematics
Georgia Institute of Technology

Professor Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Mark F. Adams
Scalable Solvers Group
Lawrence Berkeley National Laboratory

Date Approved: 18 April 2016

ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor, Richard Vuduc, for his support and his guidance over the past two years, and for giving me the chance to work for him as a research assistant in the HPC garage.

Secondly, I would like to thank Mark Adams for many insightful discussions and for helping me understanding many details of segmental refinement multigrid solvers.

Thirdly, I would like to thank the remaining committee members, Edmond Chow and Hao-Min Zhou, for their patience in scheduling and observing my thesis defense.

Special thanks go to Jed Brown who developed the HPGMG finite element multigrid solver that the segmental refinement implementation presented in this work is based on, as well as to Achi Brandt who ingeniously developed the theory of modern multigrid algorithms, including segmental refinement, and to Ulrich Rüde who introduced and got me interested in multigrid methods.

I would also like to thank my friends and colleagues of the HPC garage and all CSE faculty and students who have made my time at Georgia Tech an incredibly valuable and memorable experience.

Lastly, I would like to thank my parents and my sister for their great support throughout my entire life and especially during my time in graduate school.

This work was supported in part by the U.S. Dept. of Energy (DOE), Office of Science, Advanced Scientific Computing Research, X-Stack 1.0 under DE-FC02-10ER26006/DE-SC0004915. We also used resources of the National Energy Research Scientific Computing Center, which is supported by the DOE Office of Science under Contract No. DE-AC02-05CH11231. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of DOE.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In a parallel vertex-centered finite element multigrid solver, segmental refinement can be used to avoid all inter-process communication on the fine grids. While domain decomposition methods generally require coupled subdomain processing for the numerical solution to a nonlinear elliptic boundary value problem, segmental refinement exploits that subdomains are almost decoupled with respect to high-frequency error components. This allows to perform multigrid with fully decoupled subdomains on the fine grids, which was proposed as a sequential low-storage algorithm by Brandt in the 1970s, and as a parallel algorithm by Brandt and Diskin in 1994. Adams published the first numerical results from a multilevel segmental refinement solver in 2014, confirming the asymptotic exactness of the scheme for a cell-centered finite volume implementation.

We continue Brandt's and Adams' research by experimentally investigating the scheme's accuracy with a vertex-centered finite element segmental refinement solver. We confirm that full multigrid accuracy can be preserved for a few segmental refinement levels, although we observe a different dependency on the segmental refinement parameter space. We show that various strategies for the grid transfers between the finest conventional multigrid level and the segmental refinement subdomains affect the solver accuracy. Scaling results are reported for a Cray XC30 with up to 4 096 cores. The segmental refinement multigrid implementation is based on Jed Brown's HPGMG finite element solver; the code is publicly accessible at https://bitbucket.org/shenneking/hpgmg-fe-sr.

CHAPTER 1

INTRODUCTION

Multigrid methods are among the fastest and most efficient algorithms known for solving elliptic partial differential equations. Applying a multilevel hierarchy of discretizations and using solutions on coarse grids as an accelerator for the solution on the finest grid is the idea behind a full multigrid method (FMG). Usually, one full cycle of the FMG algorithm is sufficient to reduce the algebraic error to discretization accuracy. FMG is provably work-optimal for many problems; on the other hand, it entails large data dependencies due to its grid hierarchy.

In the context of modern computer architectures, memory efficiency becomes very important as intra-process and inter-process data movement state a severe limiting constraint to the runtime of an algorithm and also affect power consumption. *Segmental refinement*, a low-memory technique developed by Brandt in the 1970s, can be applied to a FMG full approximation scheme (FMG-FAS) algorithm to improve data locality and reduce data dependencies in the algorithm. Brandt recognized that the multigrid smoothing only serves to reduce high-frequency error components on the grid. Additionally, by looking at multigrid from a "dual point of view", fine grids can be seen as a correction to the coarse grid problem instead of regarding the coarse grid as an accelerator to the fine grid convergence. With this notion, relaxation can be done separately in subdomains on the fine grid to resolve high frequencies that cannot be resolved on the coarse grid. Errors from the interpolation cannot be eliminated at the boundaries of the decoupled subdomains; to prevent that they degrade the accuracy of the solution, segmental refinement extends each subdomain with overlapping buffer regions. Thus, the algorithm does redundant computation and requires extra-storage in the overlapping parts of the grid. In 1994, this scheme was extended to a parallel solver by Brandt and Diskin; decoupled processing of subdomains increases

asynchrony by eliminating all inter-process communication on the finest grids.

We report the first numerical results from a vertex-centered multilevel segmental refinement solver. The algorithm is presented in detail, including illustrations of the overlapping segmental refinement spaces and the modified communication operators on the transition level, which marks the transitioning from conventional multigrid to segmental refinement. We investigate accuracy, computational work, communication complexity, and scalability of the method. In particular, we show how these metrics compare to a standard FMG-FAS solver. In the analysis, we focus on the scheme's accuracy that depends on variables such as the size of the subdomains, the process grid, the size of the buffers, and the number of segmental refinement levels. We discuss how decisions in the algorithm design affect the accuracy of the method; special emphasis is given to the construction of the buffer regions and the grid transfers on the transition level.

CHAPTER 2

RELATED WORK

Most of the state of the art multigrid algorithms are based on Brandt's work in the 1970s [6, 7, 8, 9]. Since then, various types of multigrid algorithms have been developed, and there exist many advanced techniques to apply geometric and algebraic multigrid to different kinds of problems: from simple linear equations to complex nonlinear systems. Application areas involve, for example, sparse systems of algebraic equations (typically arising from discretized elliptic boundary-value problems), flow and elasticity problems, diffusion with discontinuous coefficients, or singular perturbation [9, 12, 18]. Multigrid methods can be used on complex domains, and they can solve source functions with highly localized features very efficiently by applying adaptive mesh refinement [14]. Especially for the numerical solution to discretized elliptic partial differential equations (PDEs) multigrid is known as one of the fastest if not the fastest solver. Often one full multigrid cycle is sufficient to reach discretization accuracy. Additionally, multigrid has been proven to have optimal work complexity of $\mathcal{O}(n)$ for solving Laplace's equation with n unknowns [18]. This efficiency is sometimes referred to as textbook multigrid efficiency [10].

In 1994, Brandt and Diskin presented a parallel multigrid algorithm [11]. Follow-up work showed that parallel multigrid efficiency is much harder to obtain because of the communication involved [15, 16], and several papers have studied domain decomposition and parallelization techniques for multigrid [13, 17]. Though there has been broad interest in parallel multigrid over the past decades, and a substantial body of literature exists that shows its efficient applicability to many problems, *segmental refinement* remains an open research field in which many problems have not been solved or even addressed yet.

Segmental refinement was originally introduced by Brandt as a low-storage technique that uses decoupled subdomains on the fine grids [7, 8, 12]. The motivation for such algorithms were problems where the fine grids are too large to fit in the computer memory. As memory became cheaper, these methods attracted less interest - maybe also due to the complexity involved with the implementation. Instead, a parallel algorithm based on the same idea was proposed by Brandt and Diskin in 1994 [11]. The algorithm's inherent asynchrony, which comes from communication-avoiding vertical processing of subdomains on the finer grids, makes it attractive for parallel distributed memory computing when communication is a limiting constraint. Mohr und Rüde discuss the Brandt-and-Diskin algorithm and analyze communication patterns [17]. Numerical experiments for a two-level segmental refinement scheme are conducted in [11, 17]. In 2014, Adams presented the first published multilevel numerical results of a cell-centered segmental refinement implementation [4]. Adams proposes a new segmental refinement data model and shows experimental results from a finite volume scheme analyzing the asymptotic exactness of the solver.

To the best of our knowledge, no numerical results have been published on multilevel segmental refinement methods that are vertex-centered.

CHAPTER 3

MULTIGRID BACKGROUND

This chapter is meant to give a concise overview of the main components of a conventional full multigrid solver, and it introduces the notation used for the relevant operators in this work. An excellent in-depth introduction to multigrid methods is given in [18].

3.1 Discretized Problem

We consider general elliptic boundary value problems of the form

$$Lu(x) = f(x) \qquad (x \in \Omega)\,, \tag{1}$$

where L is a linear or nonlinear operator, u the solution, and f is a forcing function or simply called right-hand side in an open domain Ω with boundary $\partial\Omega$. We assume that Dirichlet boundary conditions are given on $\partial\Omega$. The continuous differential problem is discretized using a hierarchy of grids $\Omega_h \subset \Omega$. The discretized problem is

$$L_h u_h(i) = f_h(i) \qquad (i \in \Omega_h)\,, \tag{2}$$

where u_h is the vector of unknowns, $i = 1, 2, \ldots, n$, and the unknowns are stored at grid nodes. The grid hierarchy is given by a sequence of M grids $\Omega_1, \Omega_2, \ldots, \Omega_M$, where Ω_1 is the coarsest and Ω_M the finest grid.

3.2 Grid Transfer Operators

To transfer the solution and the residual from one grid to another, grid transfer operators are applied. A transfer from Ω_k to Ω_{k-1} is usually called restriction and a transfer from Ω_{k-1} to Ω_k prolongation.

3.2.1 Injection

Injection is a simple operator that can be used as a transfer operator in either direction: from coarse to fine or from fine to coarse grid. Injection transfers only values from those grid points that exist on both coarse and fine grid. We call those points C-points on the fine grid. Fine grid points that do not exist on the coarser grid – F-points – do not participate in the operation. A restriction by injection is denoted by

$$v_{k-1} = \hat{I}_k^{k-1} v_k \ , \tag{3}$$

where v_k is a vector defined on Ω_k. A prolongation by injection is

$$v_k = \hat{I}_{k-1}^{k} v_{k-1} \ . \tag{4}$$

3.2.2 Restriction

While injection can be used to restrict the numerical solution to the coarser grid level, the residual is typically transferred using a weighted restriction:

$$v_{k-1} = I_k^{k-1} v_k \ .$$

That is, values from the F-points are taken into consideration as well. This can be implemented by aggregating values into C-points from their local neighborhood on the fine grid followed by an injection operation.

3.2.3 Prolongation

A prolongation operator is typically applied to make corrections to the fine grid solution. For example,

$$v_k = v_k + I_{k-1}^{k} c_{k-1} \ , \tag{5}$$

where c is a correction to v_k. In a full multigrid algorithm, a second operator is used for setting the initial guess on the finer grid:

$$v_k = \Pi_{k-1}^{k} v_{k-1} \ . \tag{6}$$

Often, this is a higher-order prolongation, but it may also be the same as I_{k-1}^{k}. Prolongation can be implemented by doing an injection first and then applying an appropriate interpolation to fill the F-points on the fine grid.

3.3 Smoother and Coarse Grid Solver

The smoother plays an important role for the multigrid algorithm. It smoothes the solution by eliminating the higher-frequency errors on the respective grid. Forward and Colored (e.g., Red-Black) Gauss-Seidel smoothers are a popular choice. Another is polynomial smoothing, e.g., with Chebyshev polynomials. See [2] for a comparison of different types of smoothers. In multigrid, smoothing is sometimes also called relaxation. The coarse grid solver can be any appropriate iterative or direct solver. Important is that the coarse grid is sufficiently coarse such that the application of the solver is fast. Typical choices for coarse grid solvers are Krylov subspace methods.

3.4 Multigrid V-Cycle

The multigrid idea is to use coarse grids to compute far-field contributions to the solution while smoothers are used for computing near-field contributions. Most commonly, the multigrid method is implemented recursively and computes the residual equation to create coarse correction grids, which are then used to improve the solution on the finer grids. This is called *correction scheme*, because coarser grids do not compute a full solution to the original problem but only the residual equation. Coarse grids are, roughly speaking, an accelerator to the fine grid convergence. The correction scheme does not work for nonlinear problems, because solving the residual equation does not return a valid correction to the full solution in this case. For this purpose, we can use the so-called *full approximation scheme* (FAS). Instead of computing corrections on the coarse grids during the multigrid cycle, this method computes solutions to the original problem on all coarse levels. This allows to solve nonlinear problems as well.

For linear problems, FAS and the correction scheme should compute the same solutions in every stage of the algorithm. FAS also opens up the possibility to rewrite the algorithm such that a defect correction, or τ-correction, is computed explicitly and added to the coarse grid right-hand side. This is often called the *dual point of view* [12]. Instead of regarding the coarse grid as an accelerator to the fine-grid convergence, the fine grid is used as a correction to the coarse grid problem. τ-corrections are great because they allow to inexpensively raise the order of accuracy by τ-extrapolation [5]. They also play an important role in low-memory segmental refinement. For parallel segmental refinement, τ-corrections do not have to be explicitly computed and stored. Algorithm 1 outlines a basic multigrid FAS algorithm.

Algorithm 1 Multigrid FAS V-Cycle

```
MGVCycle(L_k, u_k, f_k) {
    if (k = 1) u_k ← L_k^{-1} f_k                  · Apply coarse grid solver
    else
        S^{ν1}(L_k, u_k, f_k)                      · Pre-smoothing: ν1 iterations
        r_k ← f_k − L_k u_k                        · Compute residual
        u_{k−1} ← Î_k^{k−1} u_k                    · Inject solution
        r_{k−1} ← I_k^{k−1} r_k + L_{k−1} u_{k−1}  · Restrict residual + FAS
        c_{k−1} ← u_{k−1}                          · Store coarse-grid solution
        MGVCycle(L_{k−1}, u_{k−1}, r_{k−1})        · Recursive call
        u_k ← u_k + I_{k−1}^k (u_{k−1} − c_{k−1})  · Prolongate FAS correction
        S^{ν2}(L_k, u_k, f_k)                      · Post-smoothing: ν2 iterations
}
```

One iteration of this method is called multigrid V-Cycle. If the number of recursive calls is two, then the method is called W-Cycle. The V-Cycle is applied repeatedly until convergence or some stopping other criterion is reached. Thus, the standard multigrid algorithm is an iterative solver.

3.5 Multigrid F-Cycle

A multigrid F-Cycle, on the other hand, starts solving on the coarsest grid and successively proceeds on to finer grids, computing one or multiple V-Cycles for each level. The F-Cycle can be seen as a direct solver, because one F-Cycle is applied

to solve the problem. Both recursive and iterative implementations are common. Algorithm 2 shows the recursive algorithm. Note that multiple V-Cycles can be performed on each level if necessary.

Algorithm 2 Multigrid F-Cycle

```
MGFCycle(L_k, u_k, f_k) {
    if (k = 1) u_k ← L_k^{-1} f_k          · Apply coarse grid solver
    else
        MGFCycle(L_{k-1}, u_{k-1}, f_{k-1})   · Recursive call
        u_k ← Π_{k-1}^k u_{k-1}               · Set initial guess
        MGVCycle(L_k, u_k, f_k)               · Perform V-Cycle
}
```

The main difference to an iterative multigrid V-Cycle algorithm is that the F-Cycle sets an initial guess on the fine grids by prolongating full coarse grid solutions before computing the V-Cycle. The F-Cycle is also called full multigrid (FMG) algorithm. In this work, the term *conventional* multigrid solver refers to the full multigrid algorithm. The sketch in Figure 1 illustrates the F-Cycle.

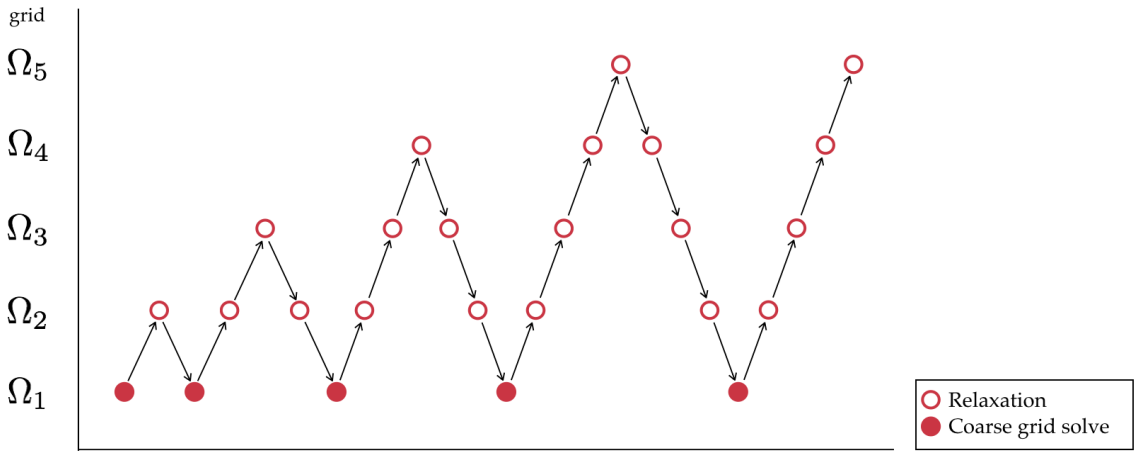

Figure 1: F-Cycle for five grid levels

CHAPTER 4

HPGMG FINITE ELEMENT

The High-Performance Geometric Multigrid (HPGMG) project is an HPC benchmarking effort based on geometric multigrid methods[1][1]. Currently, HPGMG has two implementations: a finite volume solver (HPGMG-FV) developed by Sam Williams[2], and a finite element solver (HPGMG-FE) developed by Jed Brown[3]. This chapter introduces the finite element solver which the segmental refinement implementation is based on. HPGMG-FE is a FMG-FAS solver for constant- and variable-coefficient elliptic problems on mapped coordinates using $\mathcal{Q}_1$ or $\mathcal{Q}_2$ elements. In this introduction, we consider $\mathcal{Q}_1$ elements in the computational domain, and we assume a grid refinement ratio of two. Similar concepts apply to $\mathcal{Q}_2$ or higher-order elements and different refinement ratios as well. For $\mathcal{Q}_1$ and $\mathcal{Q}_2$, the code vectorizes across elements, which is not necessary on current architectures for $\mathcal{Q}_3$ or higher. The $\mathcal{Q}_1$ reference element is shown in A.1. The implementation currently requires PETSc[4].

4.1 Domain Decomposition

Subdomains have to be defined for every grid level in the multigrid hierarchy. In a vertex-centered finite element solver, the grid partitioning can be viewed from an element space or a vertex space. For example, the grid can be decomposed in non-overlapping element partitions, which is done in this implementation, or in non-overlapping vertex partitions. Additionally, the domain decompositioning depends on the process grid on each level.

[1] http://hpgmg.org
[2] Computational Research Division, Lawrence Berkeley National Laboratory
[3] Mathematics and Computer Science Division, Argonne National Laboratory
[4] http://www.mcs.anl.gov/petsc

4.1.1 Process Grids

In a conventional distributed multigrid algorithm, different process grids are typically used to improve performance on different grid scales. This means that on coarse grid levels, processors may be idle if they do not participate in the respective process grid. The refinement ratio for process grids is two. The active process grid is restricted using Z-order so that grid transfer operators mostly involve processes that are "nearby". If the process grid is the same for two grid levels, then the domain decomposition looks the same as well. The domain decomposition describes a global view of the grid partitioning; in addition to that, every processor has a process-specific local view of its subdomain and its neighbors' subdomains, which may include ghost layers at the subdomain boundaries.

4.1.2 Global View vs. Local View

To illustrate the differences between global and local spaces, we are going to use a one-dimensional example with a constant process grid of two processors and three different grid levels Ω_1, Ω_2, and Ω_3, of dimension (extent) $4, 8$, and 16, respectively.

In the vertex-centered finite element scheme, computation is associated with cells while degrees of freedom (dofs) are associated with vertices. Figure 2 depicts the global view on each grid Ω_1, Ω_2, and Ω_3. The dashed line that separates processor 0 (P_0) and processor 1 (P_1) marks the subdomains for a dof-partition. That means, for example, on Ω_3 – grid level three – vertex-based values are stored in P_0 for vertices $\{0, 1, \ldots, 7\}$, and they are stored in P_1 for vertices $\{8, 9, \ldots, 16\}$.

But, as mentioned before, the partitioning used is a non-overlapping element space. With respect to subdomains, it means that on Ω_3, the subdomain "owned" by P_0 is $[0, 8)$, and P_1 owns $[8, 16]$. By convention, a processor owns the vertices at the low boundary of its subdomain (Ω_3: P_0 owns node $\{0\}$, P_1 owns node $\{8\}$). By implication, a processor does not own the vertices at its high boundary (Ω_3: P_0 does

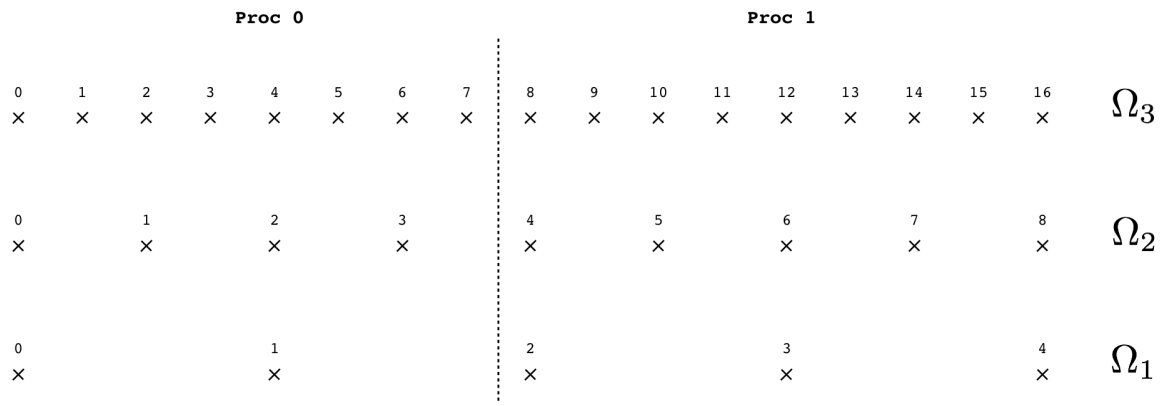

Figure 2: Global spaces

not own node $\{8\}$). An exception to that are the processors at the global high boundary that do not have a "right neighbor". They own the global boundary vertices, too (Ω_3: P_1 owns node $\{16\}$).

As a consequence, overlapping writes for the vertex-based residuals are needed for the grid nodes at the interface. This is illustrated with the local view of the processors' subdomains shown in Figure 3. The local domains ${}^p\Omega_k$ are the owned part of the grid plus any ghost points. At the same time, we define a local computing domain ${}^p\Omega_k^c$, which is essentially the same as the local domain except the global boundaries. For example, on grid level three, P_0 computes cells in ${}^0\Omega_3^c = (0, 8]$ (global space: $(0, 8]$), which means that no vertex-based values are written to the global boundary node $\{0\}$. P_1 computes in ${}^1\Omega_3^c = [0, 8)$ (global space: $[8, 16)$), and vertex $\{8\}$ (global space: $\{16\}$) remains unchanged. To compute a correct vertex-based residual, the values from P_0 and P_1 at node $\{8\}$ (in global space) have to be added; this is the overlapping write that was mentioned before. Thus, the algorithm switches from global to local space to apply the discrete operator, and then it goes back to global space and adds up vertex-based values at the inner boundaries of the non-overlapping element partition. This notion changes for segmental refinement spaces (5.2), since updating ghost points and overlapping writes both require communication between neighboring processors.

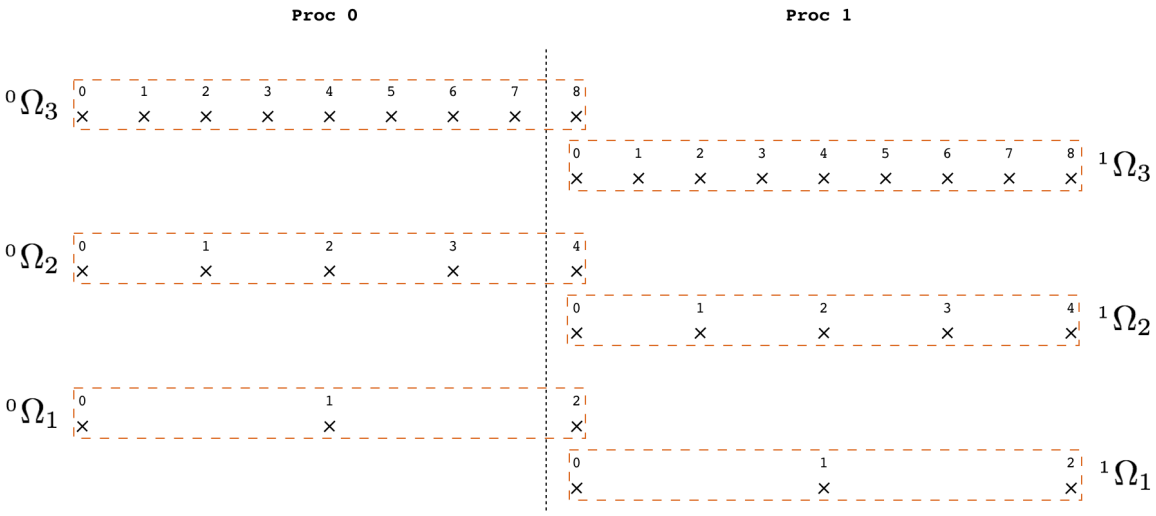

Figure 3: Local spaces

4.1.3 Communication

Communication is required for almost all operations in the full multigrid algorithm. Switching from local to global or global to local space requires information from neighboring processors, and so do the application of the preconditioner and discrete operator, pre- and post-smoothing, prolongation of the solution or restriction of the residual. For these communication routines, the multigrid solver uses PETSc *Star Forests*; a star being a simple tree consisting of a root connected to $n \geq 0$ leaves. This model is less general than an arbitrary bipartite graph, but it is sufficient for the communication needed in HPGMG-FE. A Star Forest that can be used for prolongation and restriction is discussed in the next section. Additional Star Forests for neighbor-scatter (exchange of ghost regions) and global injection are illustrated in A.2.

4.2 Grid Transfer

Grid transfer operators are needed to switch between global and local space as well as for prolongation and restriction. Injection (3.2.1) from local fine space to global coarse space is done by the Star Forest depicted in Figure 4. The *roots* of this Star

Forest are the coarse grid points in the global space and the *leaves* are the C-points on the local fine grid, including those that are ghost points. This communication routine can be used for prolongation where values are broadcasted from roots to leaves or for restriction where values are reduced from leaves to roots.

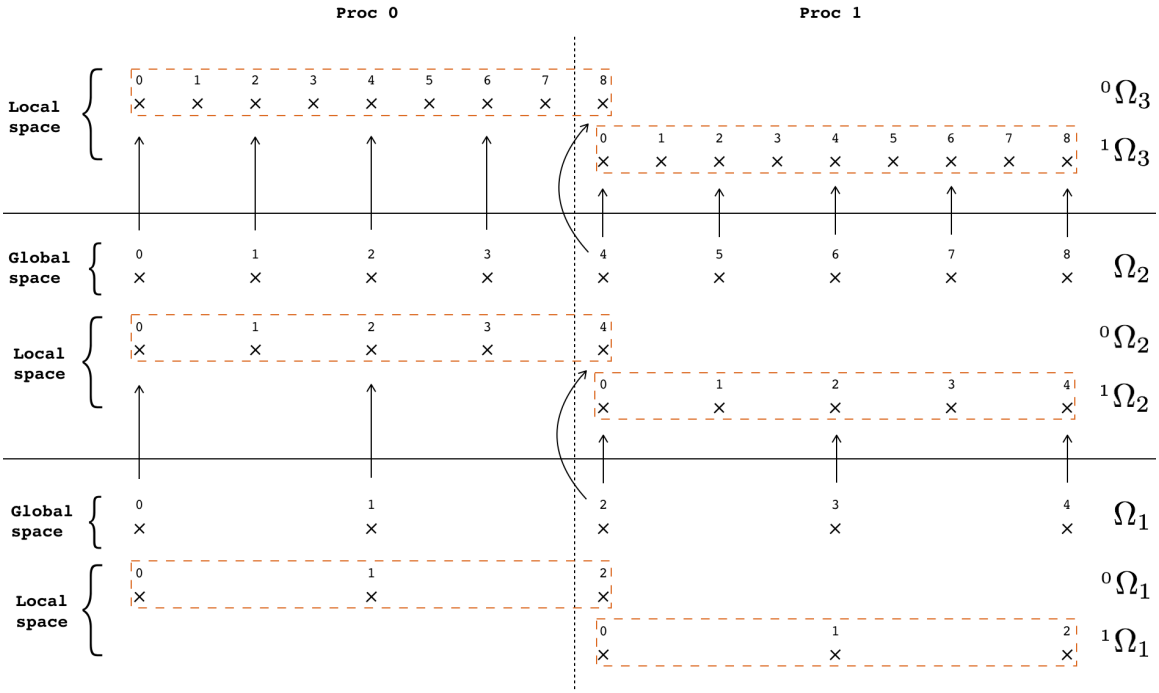

Figure 4: Star Forest for local fine to global coarse injection

4.3 Problem and Solver Setup

The model problem is the Poisson equation. The coarse grid solver is the preconditioned conjugate gradient method; preconditioning is done with the diagonal (Jacobi preconditioner). The diagonal is precomputed during the setup phase and this is not part of the timed benchmark. Smoothing is done with Chebyshev polynomials, preconditioned by the diagonal. Furthermore, the implementation uses Gauss-Legendre quadrature, which is 5th-order accurate for $\mathcal{Q}_2$ elements and 3rd-order accurate for $\mathcal{Q}_1$ elements.

The Poisson test problems are constructed, smooth problems with known solutions. The following three test cases are considered:

Wave:

$$u(x_1, x_2, x_3) = \prod_{i=1}^{3} \left(x_i^4 - x_i^2 + 2x_i^3 - 2x_i^5 \right) \tag{7}$$

Sine:

$$u(x_1, x_2, x_3) = \prod_{i=1}^{3} \sin\left(R(i)\pi x_i \right), \ R = \{1, 2, 3\} \tag{8}$$

Hump:

$$u(x_1, x_2, x_3) = (\tanh(x_1) + \log(1 + x_2) + \exp(-x_3)) \prod_{i=1}^{3} \sin\left(\pi x_i \right) \tag{9}$$

Two-dimensional plots of the exact solution with independent variables x_1, x_2 in $\Omega = (0, 1)^2$ for *Wave*, *Sine*, and *Hump* are shown in Figure 5, 17, and 18, respectively.

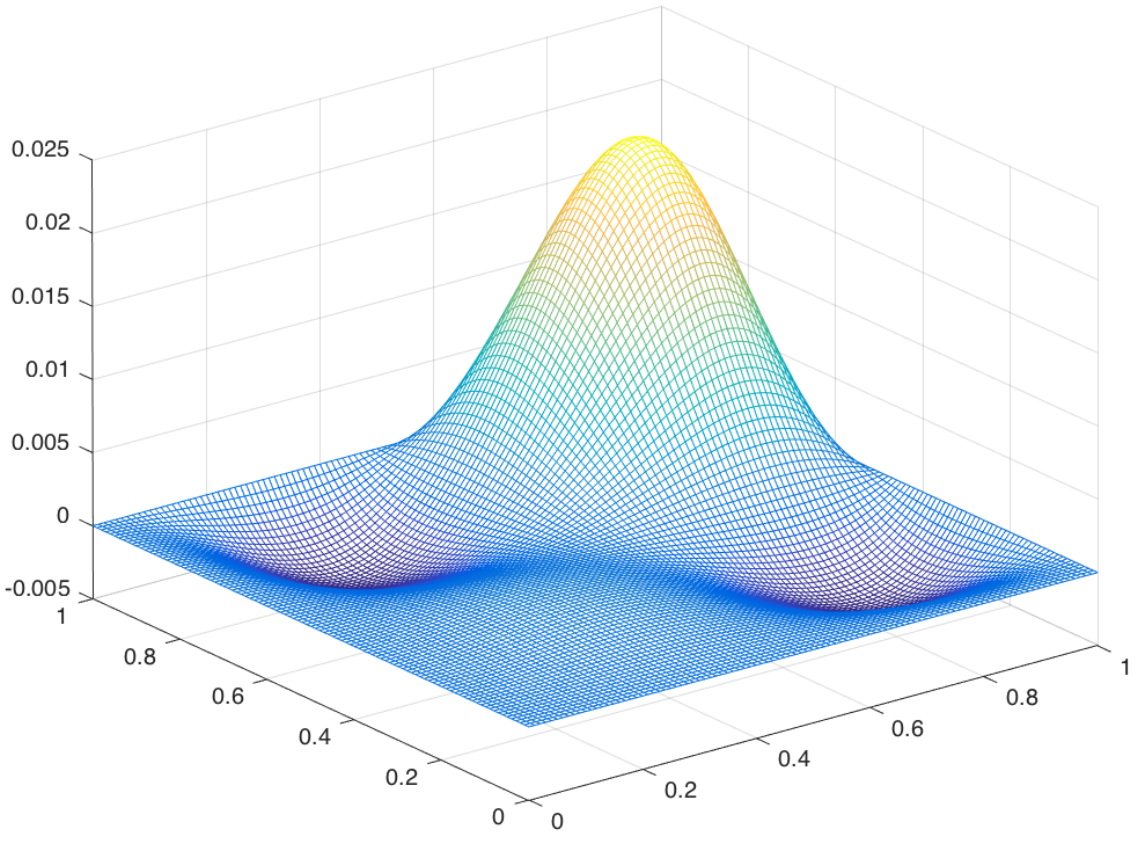

Figure 5: Solution of the two-dimensional *wave* problem

CHAPTER 5

SEGMENTAL REFINEMENT: IMPLEMENTATION

In this chapter, the segmental refinement multigrid scheme is introduced conceptually. Then, the scope of the implementation and the notation for segmental refinement spaces are defined. The main part is a detailed description of the segmental refinement implementation, including a discussion of the buffer schedule, the concept of frozen nodes, and specifics of the grid transfer between segmental refinement and conventional multigrid levels.

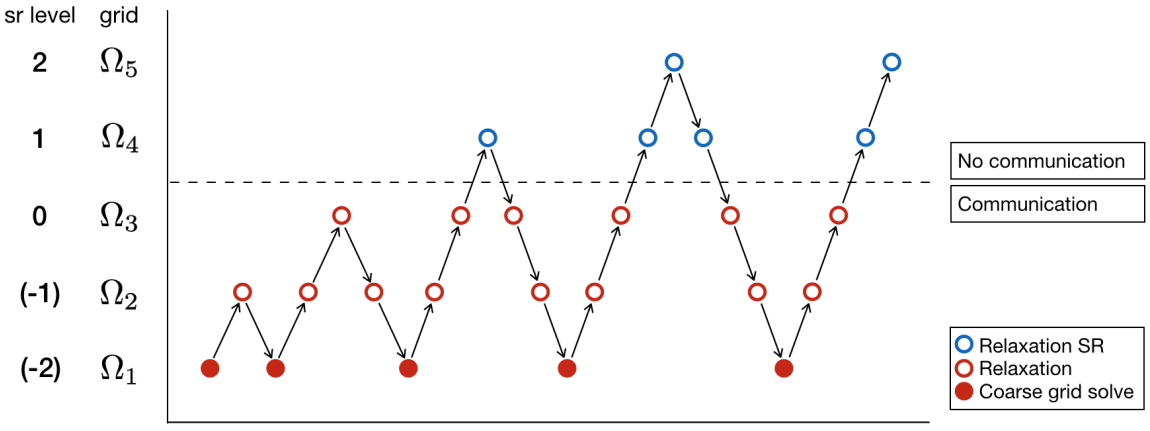

Figure 6: Segmental refinement F-Cycle

Figure 6 shows a sketch of the segmental refinement F-Cycle similar to the conventional full multigrid scheme in Figure 1. In this example, the grid hierarchy consists of five levels in total. The dashed line separates the three conventional levels Ω_1, Ω_2, and Ω_3 from the two segmental refinement levels Ω_4 and Ω_5. In general: let S denote the number of segmental refinement levels. Then, there are $M - S$ non-segmental refinement or conventional levels, and the S finest levels are segmental refinement levels, i.e., grid levels $\Omega_M, \Omega_{M-1}, \ldots, \Omega_{M-S+1}$. The *Transition Level* is the finest conventional level: Ω_{M-S}, e.g., Ω_3 in Figure 6. In addition to the normal grid index

$(k = 1, 2, \ldots, M)$, a segmental refinement index $(q = 0, 1, \ldots, S)$ is defined, which is zero on the transition level by convention - as shown in Figure 6 on the left.

In this implementation, the dashed line also separates the part of the algorithm where communication is allowed and where it is avoided. In fact, on the transition level and below, the implementation computes the conventional algorithm without any changes in the code. Segmental refinement only requires changes in the code on segmental refinement levels and in the transfers between transition level and the first segmental refinement level. Whether communication is allowed (or necessary) in those grid transfers depends on the definition of the segmental refinement spaces and the local spaces on the transition level. Here, communication is used for prolongation and restriction between transition level and first segmental refinement level, which will be discussed in more detail in 5.3.

5.1 Scope of the Implementation

The structural layout for the segmental refinement implementation should be similar to the original HPGMG finite element code so that the conventional scheme and segmental refinement can be run in the same framework. This approach drives the scope and design of the implementation along with some additional restrictions for this first version of the code:

- Segmental refinement is currently implemented for $\mathcal{Q}_1$ elements but not for higher-order discretizations. Extending the implementation to $\mathcal{Q}_2$ elements will be part of the second version.

- Results for one buffer strategy are presented: a maximum buffer schedule that is fully defined by the size of the buffer on the transition level and which grows corresponding to the multigrid refinement ratio. Additional static and dynamic buffer strategies are discussed in 5.2.2. The setup principally enables the use of advanced buffer schedules; preliminary results are not included in this work.

- The buffer size is restricted such that the exchange of the buffer points between the transition level and the first segmental refinement level can be done without increasing the number of locally "known" processes defined for a conventional multigrid solve. This is necessary to limit the scope of the Star Forest communication routines to communication within a local neighborhood of 27 processors. Consequently, for every level and in each dimension, the buffer may not be larger than the size of any neighboring (owned) subdomain in the respective level and dimension. Note that, at least in the context of the maximum buffer schedule, this is a reasonable restriction to the buffer size anyway.

- The process grid is not further refined on segmental refinement levels; that is, one segmental refinement subdomain is assigned to exactly one processor and will not be split between processes later during the computation. This strategy eliminates *all* inter-processor communication on segmental refinement levels but naturally it limits the total number of segmental refinement levels in practice due to storage limitations for each individual process. An alternative strategy would be a data model where the segmental refinement domain is split up again on a local process grid, which is a compromise with regard to reducing communication in the scheme, but it allows to further increase the number of segmental refinement levels in practice.

5.2 Subdomains and Notation

Since grids on the transition level and below are defined as conventional multigrid levels, the same subdomains and notation can be used for this part of the algorithm. On segmental refinement levels, the global spaces introduced in 4.1 for the conventional solver also remain unchanged. New spaces and notation only have to be introduced for the local spaces that contain the buffer or ghost region.

5.2.1 Local Space

In the conventional scheme, the local spaces were fully defined by the local domain ${}^p\Omega_k$ and the compute region ${}^p\Omega_k^c$. In segmental refinement, two new spaces are added: the valid region ${}^p\Omega_k^v$ and the ghost region ${}^p\Omega_k^g$. The valid region corresponds to the owned part of the grid defined by the partitioning of the global space; it contains the *valid* part of the global solution while the buffer region outside can be seen as "outdated" during the computation. The ghost region contains everything of the local domain outside the valid region, that is, ${}^p\Omega_k^g \equiv {}^p\Omega_k n {}^p\Omega_k^v$. The definition of the compute region and the local domain itself has not changed, but they are both enlarged by the growth of the ghost region, which was previously only a single layer of vertices at the high boundary. Table 1 gives an overview of the spaces that define the subdomains on segmental refinement levels.

Table 1: Definition of global and local spaces on segmental refinement levels

Ω_k	Global domain on level k
${}^p\Omega_k$	Local domain of processor p on level k
${}^p\Omega_k^c$	Compute region
${}^p\Omega_k^v$	Valid region
${}^p\Omega_k^g$	Ghost region

Figure 7 is a sketch of three grid levels: a transition level Ω_1 and two segmental refinement levels Ω_2 and Ω_3. For Ω_3, the definition of the local spaces is illustrated for both processors P_0 and P_1. Compared to the original local spaces (cf. Figure 3), the local segmental refinement spaces are extended by the new ghost region. The spaces are element-partitioned, but values for the solution and residuals are vertex-based. Thus, each space we compute on also defines whether vertex-based values at the low and high boundary are included or excluded when writing values to vertices, which is marked by a filled square or framed square, respectively, in Figure 7. For example, the compute region excludes vertices at the global boundaries, because those values

are not overwritten at any point and the vertex-based residuals are zero. A similar case applies to frozen boundary nodes, which is discussed in 5.2.3. Table 2 shows the exact numbers for P_0's spaces on all three grid levels in Figure 7.

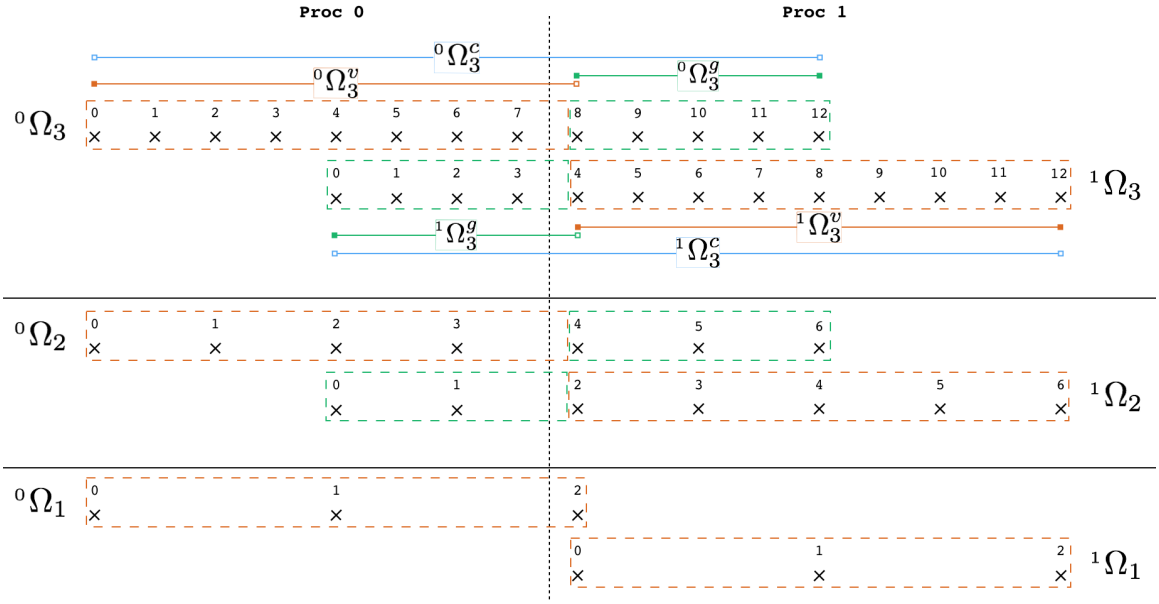

Figure 7: Local spaces on segmental refinement levels

Table 2: P_0's global and local spaces on Ω_1, Ω_2, and Ω_3 (Figure 7)

Ω_3	$[0, 16]$	Ω_2	$[0, 8]$	Ω_1	$[0, 4]$
$^0\Omega_3$	$[0, 12]$	$^0\Omega_2$	$[0, 6]$	$^0\Omega_1$	$[0, 2]$
$^0\Omega_3^c$	$(0, 12)$	$^0\Omega_2^c$	$(0, 6)$	$^0\Omega_1^c$	$(0, 2)$
$^0\Omega_3^v$	$[0, 8)$	$^0\Omega_2^v$	$[0, 4)$		
$^0\Omega_3^g$	$[8, 12]$	$^0\Omega_2^g$	$[4, 6]$		

5.2.2 Buffer Schedule

Having defined the local spaces, we can now discuss one of the most important parts of the segmental refinement multigrid scheme: *The Buffer Schedule.*

The buffer schedule defines the size of the ghost regions for each segmental refinement level. Therefore, it impacts almost every aspect of the segmental refinement solver. The solver accuracy depends on the errors that propagate from the finite

buffer into the valid part of the domain, and the additional computational work directly corresponds to the overhead associated with buffer computation. The goal is to find buffer schedules that minimize the computational overhead while still providing an accurate multigrid scheme. We would also like to find a buffer strategy that allows for an asymptotically exact solver, which is addressed in 6.2.

A general expression for the buffer schedule is

$$J(q) = a + bq + cq^2 + \ldots + e(m^q),$$

where $J(q)$ denotes the extent of the buffer in each dimension on the segmental refinement level q. The "components" of the buffer schedule are defined in Table 3:

Table 3: Components of the buffer schedule

a	Static buffer
bq	Linear buffer
cq^2	Quadratic buffer
$\vdots$	
$e(m^q)$	Exponential buffer

The coeffcients $a, b, c, \ldots$, can be any integer number: positive, negative, or zero; in practice, certain requirements have to be met to assert a valid buffer schedule. For example, the buffer must not grow faster than the refinement ratio of the multigrid scheme:

$$\frac{J(q+1)}{J(q)} \leq r \ ,$$

where r is the grid refinement ratio. Otherwise, the local domain on the segmental refinement level would have to grow during the prolongation, which in turn required communication. Thus, this simply means that for each processor the fine grid cannot cover a larger subdomain than the coarse grid. Of course, the buffer must also not be zero or negative. Additionally, we defined the requirement that the buffer must not be larger than any neighboring processor's valid region to limit communication

to a local neighborhood on the process grid (cf. 5.1). One last requirement that is not necessary but recommended is that the size of the buffer is a multiple of the grid refinement ratio; in other words, the buffer region should end with a C-point. For the common grid refinement ratio of two, this implies that the extent of the buffer should be an even number. The reason for that has to do with the accuracy of the frozen nodes and is discussed in 5.2.3.

The *Maximum Buffer Schedule* causes the buffer to grow at the same rate as the grid refinement: $J(q) = e(r^q)$. This is simply a special case where all coefficients $a, b, c, \ldots$ of $J(q)$ are zero except for the exponential buffer ($e > 0$), and $m = r$. That implies that each segmental refinement level spans the same subdomain of the grid. Hence,

$$\text{Maximum Buffer Schedule} \equiv \text{Constant Buffer Domain.}$$

Any other buffer schedule can be called a dynamic buffer schedule, because the part of the domain spanned by the segmental refinement buffer changes from level to level:

$$\text{Dynamic Buffer Schedule} \equiv \text{Variable Buffer Domain.}$$

If a is the only non-zero coefficient, then we get a constant buffer size, that is, the number of grid points in the buffer is constant for each level. If a dynamic buffer schedule is selected, we need to define another local space in the segmental refinement scheme, which may be called the *support region* ${}^{p}\Omega_k^f$ [4]. This would be the region of the subdomain that is covered by both coarse and fine grid, i.e., ${}^{p}\Omega_k^f \equiv {}^{p}\Omega_k \cap {}^{p}\Omega_{k+1}$. The support region includes only a part of the segmental refinement buffer on ${}^{p}\Omega_k$, since the next finer grid ${}^{p}\Omega_{k+1}$ has a smaller buffer domain than the grid on level k due to the dynamic buffer schedule. It is called support region, because for this part of the domain, the fine grid can support the coarse grid with information such as the fine grid residual. The "unsupported" part of the buffer, which is given by ${}^{p}\Omega_k \backslash {}^{p}\Omega_k^f$, cannot compute τ-corrections or receive residuals from the fine grid, which may cause

additional errors in the solution. From here on, only the maximum buffer schedule will be considered, if not mentioned otherwise.

5.2.3 Frozen Nodes

A special case are the interior boundaries of the segmental refinement subdomains which are not global boundaries. As any communication is avoided on segmental refinement levels, the computation at segment boundaries has to be considered carefully, because this is where the additional errors originate due to the lack of updates. For example, in the one-dimensional case (see Figure 7), at the right boundary of P_0, the computation of a vertex-based residual would require information from the element to the left and the element to the right. Since the right element is not available at this point, the residual cannot be computed correctly. To deal with this issue, the value of the outer boundary points will be "frozen" during the segmental refinement computation. That is, they are set during the prolongation from the transition level to the first segmental refinement level and remain unchanged on finer levels. In the implementation, this is realized by setting the vertex-based residual to zero at frozen nodes and "pretending" that the solution there was accurate. The reason why the frozen nodes should be C-points comes from considerations about the accuracy of the scheme. Since values at the frozen nodes should be as accurate as possible, it is recommended not setting those values by interpolation in order to avoid the interpolation error. The frozen nodes are updated exactly once per V-Cycle in the prolongation operator after receiving the corrected values from the neighboring processors during the conventional solve on the transition level.

5.3 Transition Level

The transition level plays a special role in the segmental refinement scheme. While the discretized problem on the transition level itself is solved with the conventional multigrid algorithm, there are several points that need to be addressed in the grid

transfer between the transition level and the first segmental refinement level.

5.3.1 Prolongation

In the finite element code, the prolongation operator is implemented by performing an injection for the C-points followed by interpolation for the F-points on the fine grid. The first special case is the initial guess on the first segmental refinement level. This case is relatively simple to handle: the injection operator uses a modified Star Forest that populates all C-points on the fine grid, including those in the ghost region and the frozen nodes. The roots are the coarse grid points in the global space similar to the conventional injection operation. An example is illustrated in Figure 8. Note that, in this example, node $\{6\}$ in ${}^0\Omega_2$ and node $\{0\}$ in ${}^1\Omega_2$ are frozen nodes, which are set in this prolongation and then kept throughout the computation until this prolongation is done again in the FAS correction of the next V-Cycle. The FAS correction on the first segmental refinement level is more subtle than setting the initial guess and is discussed in detail in 5.3.3.

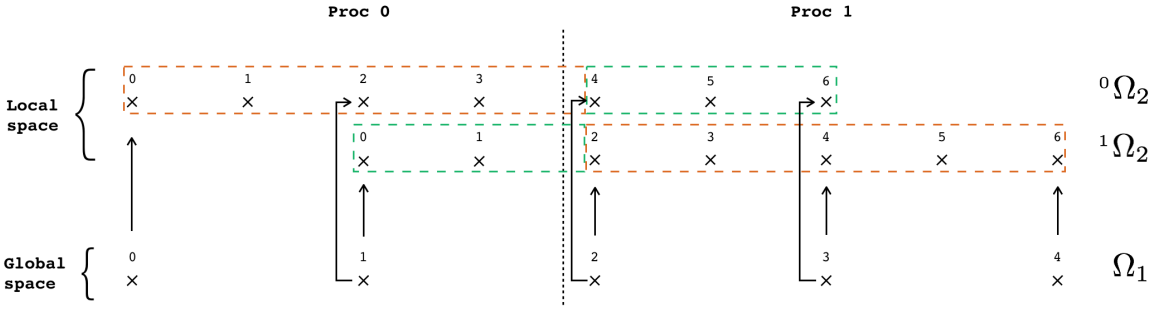

Figure 8: Injection between transition level and first segmental refinement level

5.3.2 Restriction

In the restriction, there are also two different cases that need to be considered: firstly, the restriction of the state vector (solution u) by injection, and, secondly, the restriction of the residual. In both cases, the restriction can be done entirely without

communication *or* with some communication for the vertices at the interface. In Figure 8, node $\{2\}$ on the global grid on the transition level Ω_1 is at the interface between P_0 and P_1. The state vector can be restricted by using either value $\{4\}$ or $\{2\}$ on the local fine grids ${}^0\Omega_2$ or ${}^1\Omega_2$, respectively, or by averaging those values. Communication is only avoided completely by choosing the local fine grid value that resides on the processor that owns node $\{2\}$ on the transition level (P_1). This corresponds to injecting the C-points in the valid region of the local fine grid into the global coarse grid. In the experiments, we could not find a significant difference in the accuracy of the proposed techniques and hence decided to use the communication-avoiding option. This choice may be revisited if found to have an impact on the accuracy of the scheme.

In the restriction of the residual, the point at the interface needs information from the respective C-point on the fine grid and the neighboring F-points, too. For node $\{2\}$ in Figure 8, the information could be collected from vertices $\{3\}, \{4\}$, and $\{5\}$ on ${}^0\Omega_2$, or from vertices $\{1\}, \{2\}$, and $\{3\}$ on ${}^1\Omega_2$, or from both grids in some way. While using solely information from ${}^1\Omega_2$ would avoid all communication, it was found in the experiments that the better results are obtained by aggregating values from all subdomains at the interface. In this example, the vertex-based residual would be calculated based on the values at node $\{3\}$ on ${}^0\Omega_2$ as well as vertices $\{2\}$ and $\{3\}$ on ${}^1\Omega_2$.

The accuracy of the restriction in all these cases certainly depends on several factors such as the buffer schedule or the smoothness of the solution at the interface. A more detailed study on the effects of the different choices presented did not seem justified due to the apparently small effect on the overall accuracy of the solver, at least for linear problems. A much more significant impact appears to come from the strategy for the FAS correction on the first segmental refinement level.

5.3.3 FAS Correction

To discuss the issue with the FAS correction on the first segmental refinement level, we need to revisit the multigrid FAS algorithm and look at what happens in this specific case. Recall that, after restricting the solution, the FAS V-Cycle solves the following equation on the transition level Ω_k:

$$L_k u_k = L_k(\hat{I}^k_{k+1} u_{k+1}) + I^k_{k+1} r_{k+1} \ , \tag{10}$$

where r_{k+1} is the residual computed on the fine grid Ω_{k+1}, which is the first segmental refinement level in this case. Hence, new components in the solution u_k will be computed based on the restricted fine grid residual. After the V-Cycle, FAS correction typically corrects the fine grid solution with the correction equation

$$u_{k+1} = u_{k+1} + I^{k+1}_k (u^{new}_k - u^{old}_k) \ , \tag{11}$$

where u^{old}_k and u^{new}_k are the solution before and after the V-Cycle, respectively. The problem here are the segmental refinement buffers. We already discussed that the frozen nodes should have a full update of the solution, but the remaining nodes in the buffer may be inaccurate, too, if the FAS correction is applied this way.

Figure 9 illustrates the computation of the restricted fine grid residual for node $\{5\}$ on the transition level Ω_k. Since the residual at that point is based on the residual values at nodes $\{5\}, \{6\}$, and $\{7\}$ in ${}^1\Omega_{k+1}$, the FAS correction computed at $\{5\}$ in Ω_k is *intended* as a correction to the values at nodes $\{5\}, \{6\}$, and $\{7\}$ in ${}^1\Omega_{k+1}$. The old values in the ghost region of ${}^0\Omega_{k+1}$ at nodes $\{9\}, \{10\}$, and $\{11\}$ receive the same corrections in the prolongation operator (cf. Figure 8). But usually, the values in the buffer region are significantly more inaccurate than the values in the valid region, which means, they have different residuals and thus would need different corrections than the values in the valid region.

In order to address this problem, one could either compute different corrections for ghost and valid region or update the ghost values and apply the same corrections.

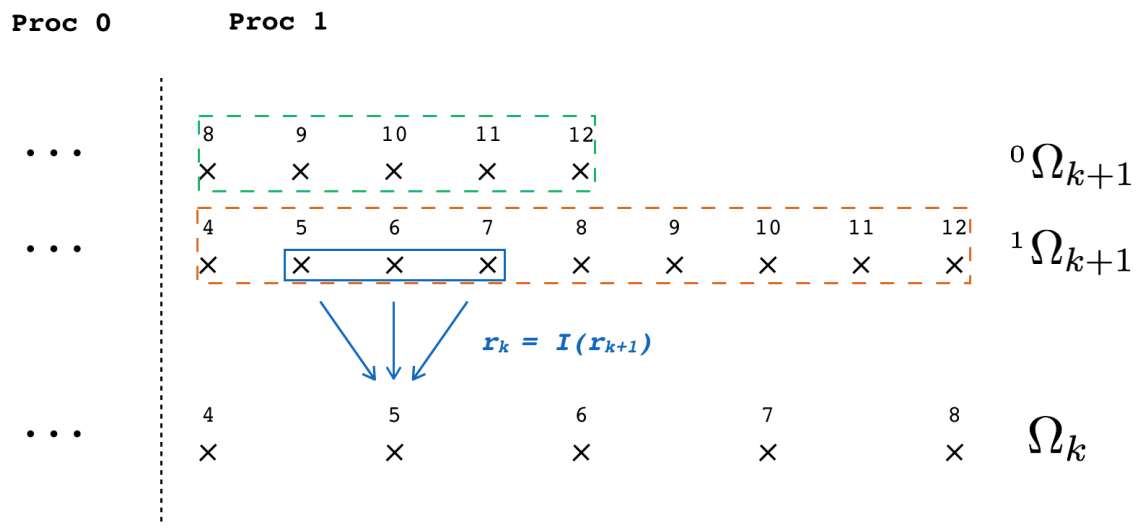

Figure 9: Fine grid residual restriction on the transition level

In both cases, the ghost region would then receive accurate FAS corrections. In the ghost update, one also has the choice to either only update the C-points and thereby avoid communication on the first segmental refinement level but introduce additional interpolation errors in the F-points or update the F-points in the ghost region as well to set them equal to the nodes in the valid region for the best accuracy.

Based on the experiments, we recommend to do a full update of at least all the C-points in the buffer region of the first segmental refinement grid instead of using the conventional FAS correction.

5.4 Algorithm

Finally, the algorithm can be formally defined, including the spaces that each operator works on. Both parts of the scheme, the recursive F-Cycle and the recursive FAS V-Cycle, are executed on segmental refinement levels only, and they call the conventional algorithm when they reach the transition level ($k = M - S$ or $q = 0$). Note that the details of restriction and prolongation on the transition level are not explicitly dealt with in the V- and F-Cycle algorithms in 5.4.1 and 5.4.2 but are expected to be implemented as part of the respective operator.

5.4.1 Segmental Refinement V-Cycle

Algorithm 3 Multigrid segmental refinement FAS V-Cycle

`MGVCycleSR`(L_k, u_k, f_k, q) {

if $(q = 0)$ `MGVCycle`(L_k, u_k, f_k)	· Call conventional V-Cycle
else	
$S^{\nu_1}(L_k, u_k, f_k)$	· Pre-smoothing, ${}^{p}\Omega_k^c$
$r_k \leftarrow f_k - L_k u_k$	· Compute residual, ${}^{p}\Omega_k^c$
$u_{k-1} \leftarrow \hat{I}_k^{k-1} u_k$	· Inject solution, ${}^{p}\Omega_{k-1}$
$r_{k-1} \leftarrow I_k^{k-1} r_k + L_{k-1} u_{k-1}$	· Restrict residual + FAS, ${}^{p}\Omega_{k-1}^c$
$c_{k-1} \leftarrow u_{k-1}$	· Store coarse grid, ${}^{p}\Omega_{k-1}^c \cup {}^{p}\Omega_{k-1}^g$
`MGVCycleSR`$(L_{k-1}, u_{k-1}, r_{k-1}, q-1)$	· Recursive call
$u_k \leftarrow u_k + I_{k-1}^k (u_{k-1} - c_{k-1})$	· FAS correction, ${}^{p}\Omega_k^c \cup {}^{p}\Omega_k^g$
$S^{\nu_2}(L_k, u_k, f_k)$	· Post-smoothing, ${}^{p}\Omega_k^c$

}

5.4.2 Segmental Refinement F-Cycle

Algorithm 4 Multigrid segmental refinement F-Cycle

`MGFCycleSR`(L_k, u_k, f_k, q) {

if $(q = 0)$ `MGFCycle`(L_k, u_k, f_k)	· Call conventional F-Cycle
else	
`MGFCycleSR`$(L_{k-1}, u_{k-1}, f_{k-1}, q-1)$	· Recursive call
$u_k \leftarrow \Pi_{k-1}^k u_{k-1}$	· Set initial guess, ${}^{p}\Omega_k^c \cup {}^{p}\Omega_k^g$
`MGVCycleSR`(L_k, u_k, f_k, q)	· Perform V-Cycle

}

CHAPTER 6

SEGMENTAL REFINEMENT: ANALYSIS

In this chapter, we intend to provide insight into the relations and asymptotics of the segmental refinement parameter space through collecting experimental data. We also address the communication and computation complexity in the segmental refinement algorithm. The $\mathcal{Q}_1$ finite element scheme is second-order accurate. The results presented in this chapter were computed for the three-dimensional Poisson *wave* problem defined in 4.3, eq. (7).

6.1 Parameter Space

The results we are interested in are the accuracy, scaling, running time, and communication and computational work of the scheme. While the accuracy of the algorithm fully depends on a set of independent variables specified in the segmental refinement scheme, the scaling and running time for example both need to be considered in the light of the machine model, which will be discussed in 7.

The independent variables of the scheme are the buffer schedule $J(q)$, the process grid P, the finest grid size M, and the number of segmental refinement levels S. Because the buffer is the maximum buffer schedule, it is uniquely defined by the size of the coefficient e (or the size of the buffer on the first segmental refinement level: $2e$). Hence the parameter space is four-dimensional. Other variables that may be of interest, such as the subdomain size on the transition level, are unequivocally specified through the other variables.

The accuracy of the scheme is measured with the error ratio e_r, defined as

$$e_r = e_{sr}/e_{conv}, \tag{12}$$

where e_{sr} and e_{conv} are the error of the segmental refinement solver and the conventional solver, respectively, in some norm. Note that the preferred method to

determine the segmental refinement accuracy may be application-specific. Here, we consider the solver to be sufficiently accurate, if the error of the segmental refinement scheme is not more than 10% above the conventional solver error, i.e., $e_r \leq 1.1$. The error is computed as the difference between the numerical solution and the known exact solution. The norm is not specified closer at this point and will be mentioned explicitly for the presented results. Generally, the error norms used are a weighted $L2$ norm and the maximum (L^∞) norm. Both norms clearly show a reduction of the error according to the second-order accuracy in the conventional scheme with a $V(3,1)$-Cycle. This choice is used for the computation in the conventional and the segmental refinement solver.

6.2 Experiments and Observations

We first look at the results for a fixed process grid $P = [2\ 2\ 2]$, three segmental refinement levels, and a fixed small buffer schedule of $e = 1$. The results for varying subdomain sizes are shown in Table 4. We show the size of the valid region of the subdomain on the finest grid denoted by M_p. The global grid size is denoted by M. Both variables represent the extent of the respective grid in each of the three dimensions. That is, $M = 32$ refers to the global grid $M = [32\ 32\ 32]$ that has 33^3 vertices. The variable p is used in the same way, i.e., $p = 4$ refers to the process grid $P = [4\ 4\ 4]$.

The dependency on the subdomain size follows a similar trend for all three problems and the range of tested process grids as well as different buffers and numbers of segmental refinement levels. We observe that the relative error e_r increases with increasing subdomain size.

Adams [4] observed a dependency on the subdomain size on the transition level, too, but the dependence is just the other way around; that is, in [4], the error ratio e_r improves for increasing subdomain sizes. Possible explanations for this discrepancy

Table 4: Error ratio $e_r = e_{sr}/e_{conv}$ in the $L2$ norm: $S = 3, e = 1, P = [2\ 2\ 2]$

$M = 32$	$M = 64$	$M = 128$	$M = 256$
$M_p = 16$	$M_p = 32$	$M_p = 64$	$M_p = 128$
1.00	1.10	1.23	1.46

could be that segmental refinement may be particularly sensitive to the order of prolongation, because it is used to set the frozen nodes in the ghost region, or to the computation of the segment boundaries. This point should therefore be investigated further as it clearly affects the accuracy of the scheme.

The dependence on the subdomain size can be seen in Table 5 as well. This table also shows some of the expected dependencies on the size of the buffer and the number of segmental refinement levels: generally, the more segmental refinement levels are used, the larger the buffer must be for the scheme to be accurate. Corresponding to the subdomain size dependency, the buffer has to grow slightly for a constant number of segmental refinement levels to accomodate the increase of the error ratio for increasing M. For example, with three segmental refinement levels, the scheme yields accurate results for $M = 128, e = 3$, but as the subdomain size is increased and $M = 512$, the scheme needs a buffer of $e = 4$ to obtain sufficient accuracy.

Table 5: Error ratio $e_r = e_{sr}/e_{conv}$ in the L2 norm: $P = [4\ 4\ 4]$

	$M = 64$		$M = 128$		$M = 256$		$M = 512$	
	$S = 1$	$S = 2$	$S = 2$	$S = 3$	$S = 2$	$S = 3$	$S = 3$	$S = 4$
$e = 1$	2.3	2.6	3.4	10.4	6.1			
$e = 2$	<u>1.0</u>	<u>1.0</u>	<u>1.0</u>	4.0	<u>1.0</u>	13.2		
$e = 3$				<u>1.1</u>	<u>1.0</u>	2.1	4.8	6.9
$e = 4$							<u>1.0</u>	<u>1.0</u>

Figure 10 shows for a process grid of eight processors how the error affects convergence in the segmental refinement scheme. In this example, there are four segmental refinement levels and the transition level subdomain size is 4 in each dimension. The plot shows the error on a log-log scale with the grid dimension on the x-axis. The segmental refinement solver with buffer $e = 1$ and $e = 2$ is compared to the 2nd-order accurate conventional solver.

The final error ratios are 3.08 and 1.26 for the buffer with $e = 1$ and $e = 2$, respectively. The plot shows that with the smaller buffer, the scheme is 2nd-order accurate for two segmental refinement levels, then loses some accuracy on the third and most of its accuracy on the very last level. The larger buffer yields similar results: the scheme is accurate for three segmental refinement levels and differs from the conventional scheme only on the finest grid level. This observation indicates that the errors from the buffer region need a certain time to reach and affect the valid region of the subdomains, and it matches the results we obtained with Adams' finite volume segmental refinement solver.

Figure 11 shows a similar plot for a refined process grid with 64 segments in total and three segmental refinement levels. As indicated by the results in Table 5, the solver needs the $e = 3$ buffer for sufficient accuracy. But the behavior we observed in Figure 10 for the coarser process grid is alike: convergence starts to degrade somewhat at some segmental refinement level and then degrades much stronger if the segmental refinement is continued on finer grids.

With regard to the asymptotic exactness of the solver, we aim to find a buffer strategy that *theoretically* allows unlimited adding of segmental refinement levels. Consider the approach: $M' = 4M$, $S' = S + 1$, $e' = 2e$, and a constant process grid.

This effectively doubles the subdomain size on the transition level, increases the number of segmental refinement levels by one, and doubles the size of the buffer on the first segmental refinement level which in turn preserves the ratio of buffer size to

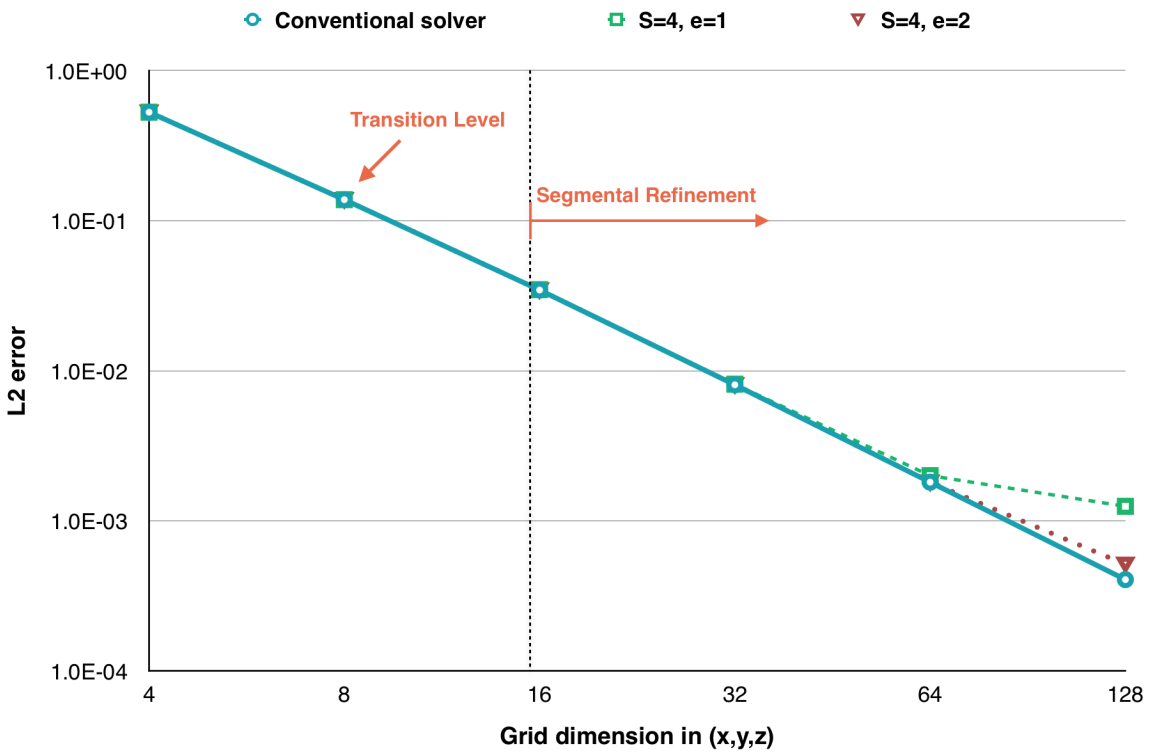

Figure 10: Error in the L2 norm for segmental refinement on process grid $P = [2\ 2\ 2]$

subdomain size and therefore has the same computational overhead on each segmental refinement level as before.

For example: consider an accurate setting with $S = 2$ segmental refinement levels and buffer $e = 2$ for a fine grid $M = 256$ with process grid $p = 4$. Then, a third segmental refinement level can be added by using $M = 1024$, $S = 3$, $e = 4$, which has the same computational overhead for each segmental refinement level as the two-level segmental refinement previously. Continuing this approach, we would add a fourth segmental refinement level for $M = 4096$, $e = 8$, then a fifth for $M = 16384$, $e = 16$, and so on. This seems to yield an asymptotically exact solver based on the data collected during the experiments. The approach is rather theoretical, because larger problem sizes will usually be tackled with more processors due to memory and time constraints. Thus, we now look at the error ratios for more practical scenarios where the number of processors increases as well.

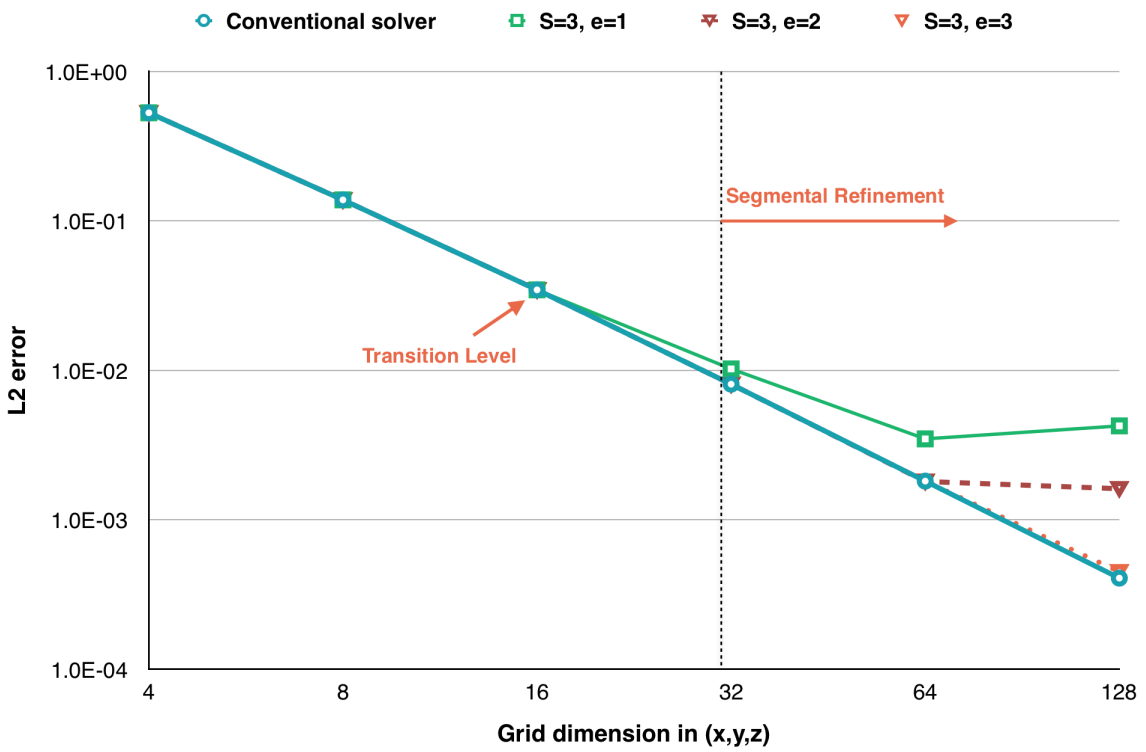

Figure 11: Error in the L2 norm for segmental refinement on process grid $P = [4\ 4\ 4]$

In this scheme, scaling up the number of processors does not allow for an asymptotically accurate solver which is adding successively more and more segmental refinement levels - at least not while having a constant subdomain size on the transition level. The reason is that for an accurate solution, the buffer will have to be increased when adding more segmental refinement levels, and this is only possible up to a point where the buffer region on the transition level covers the entire neighboring subdomain. Instead, we look at how the error depends on the process grid in the scaling scenario. Table 6 shows the error ratios for two segmental refinement levels and refined process grids with different buffer sizes. As expected, the refined process grids require larger buffers to be accurate, since with every refinement, the number of segments and segment boundaries with frozen nodes grows as well. In this example, doubling the number of segment in each dimension requires an increase of 1 in the buffer coefficient e.

Table 6: Error ratio $e_r = e_{sr}/e_{conv}$ in the L2 norm for $S = 2$

	$M = 64$ $p = 2$	$M = 128$ $p = 4$	$M = 256$ $p = 8$	$M = 512$ $p = 16$
$e = 1$	1.2	3.4		
$e = 2$	<u>1.0</u>	<u>1.0</u>	1.6	
$e = 3$	-	-	<u>1.1</u>	1.5
$e = 4$	-	-	-	<u>1.0</u>

For further refined process grids, this appears not to be necessary, because at some point the buffer should be large enough to guarantee an accurate solution for a certain number of segmental refinement levels independent of the process grid that is used. Table 7 shows some indication of that; the scheme yields equivalently accurate results for three segmental refinement levels for a buffer of $e = 4$ for both process grids $P = [8\ 8\ 8]$ and $P = [16\ 16\ 16]$.

Table 7: Error ratio $e_r = e_{sr}/e_{conv}$ in the maximum norm for $S = 3$

	$M = 128$ $p = 2$	$M = 256$ $p = 4$	$M = 512$ $p = 8$	$M = 1024$ $p = 16$
$e = 1$	<u>1.0</u>	19.3		
$e = 2$	<u>1.0</u>	8.1	38.6	
$e = 3$	-	1.2	6.5	24.3
$e = 4$	-	-	<u>1.0</u>	<u>1.0</u>
$e = 5$	-	-	-	<u>1.0</u>

6.3 Communication and Computation Complexity

Clearly, the maximum buffer schedule is not a very practical buffer, because it grows exponentially with the segmental refinement subdomain according to the grid refinement ratio. On the other hand, the relative cost, i.e., the relative overhead for buffer computation vs. computation in the valid part of the domain, does not increase. The maximum buffer is interesting, since it provides sort of a baseline that more practical, dynamic buffers should try to reach in terms of accuracy while doing substantially less work.

For typical problem sizes per processor and with the maximum buffer schedule, the subdomain size was usually extended by factors between 1.125 and 1.5 in the experiments. For three-dimensional problems, this corresponds to an increase of 1.42 up to 3.38 in computational work on the first segmental refinement level. This could likely be compensated by a non-exponential dynamic buffer schedule, where the relative overhead decreases exponentially with every segmental refinement level; with the maximum buffer schedule though, the overhead remains constant, which corroborates that this buffer schedule should be thought of as a baseline for the analysis of segmental refinement accuracy but not for improving the running time. The relative computational overhead on a segmental refinement level could then be written as $(2J(q) + {}^{p}\Omega^{v}_{M-S+q})^3/({}^{p}\Omega^{v}_{M-S+q})^3$, where ${}^{p}\Omega^{v}_{M-S+q}$ refers to the size of the valid region of the subdomain on the q-th segmental refinement level per dimension, assuming cubic subdomains.

Different segmental refinement data models have been proposed an analyzed by Brandt [12], Mohr [17], and Adams [4], including more refined approaches to distinguish between far and near-communication for different memory partitions. Here, we only briefly discuss the inter-process communication which is avoided but also added by segmental refinement. A comprehensive memory complexity analysis for multilevel segmental refinement supported by detailed timing data from different machine

models is yet to be done.

Inter-process communication is avoided during the computation on all segmental refinement levels and in the grid transfers between segmental refinement levels. Additional communication occurs in the prolongation between transition level and the first segmental refinement level for filling (or correcting) the buffer values and thus depends on $J(1)$. For example, an update of the C-points on the first segmental refinement level "costs" $({}^{p}\Omega_{M-S} + 2(J(1)/2))^2(J(1)/2)$ for one "ghost face" of a cube subdomain with refinement ratio two, assuming the same process grid on transition level and first segmental refinement level. In the three-dimensional case, for each transition level prolongation, the inter-process communication is slightly less than six times that value. As discussed in 5.3, further communication may occur on the transition level depending on the strategy for restriction of the residual and the solution.

CHAPTER 7

SCALING STUDIES

Scaling was tested on Edison[1] (Cray XC-30 at NERSC). Each node of Edison has 2 sockets containing 12-core Intel "Ivy-Bridge" (E5-2695v2) processors. Each core has a 256 bits wide vector unit. Edison uses the Cray Aries network, a high-radix dragonfly network; the MPI latency is between $0.25\mu s$ and $3.7\mu s$ and the MPI bandwidth is about $8\text{GB}/s$. With this configuration, Edison performs very well on conventional multigrid algorithms in terms of scaling compared to other supercomputers that have a slower network. For example, the HPGMG finite volume benchmark found that Edison performs up to *three* times better than NREL's Peregrine[2] while the HPL benchmark[3] showed a difference of only 20% in performance. Both machines use an identical Xeon-based node architecture, but where Peregrine uses an Infiniband fat-tree network, Edison uses a Cray Aries Dragonfly network. Therefore, it is important to consider which machine the communication-avoiding implementations are tested on, because the scaling results may highly depend on it. In this section, we show results only for Edison; for a larger scaling study, different machine models should be taken into consideration.

In the HPGMG finite element implementation, the computation of the diagonal has not been optimized and is not a timed part of the benchmark. For measuring scalability, the solver is preloaded with one solve to "warm up" followed by eight timed solves.

Figure 12 shows weak scaling results on Edison. In this weak scaling study, the extent of the local fine grid per process is 32^3. The number of cores reaches from

[1]http://www.nersc.gov/users/computational-systems/edison/configuration
[2]http://hpc.nrel.gov/users/systems/peregrine
[3]http://www.netlib.org/benchmark/hpl

64 to 4096. The y-axis shows parallel efficiency; process grid $P = [4\ 4\ 4]$ serves as a baseline, because it includes "interior" processes that do not reside at any global boundary on the finest grid. Segmental refinement multigrid was run with two and three segmental refinement levels. Adams [4] found that segmental refinement caused a modest increase in scalability which appears to be the case for sufficiently large process grids here as well. The cross-over between two and three segmental refinement levels in Figure 12 may be due to limited sample size.

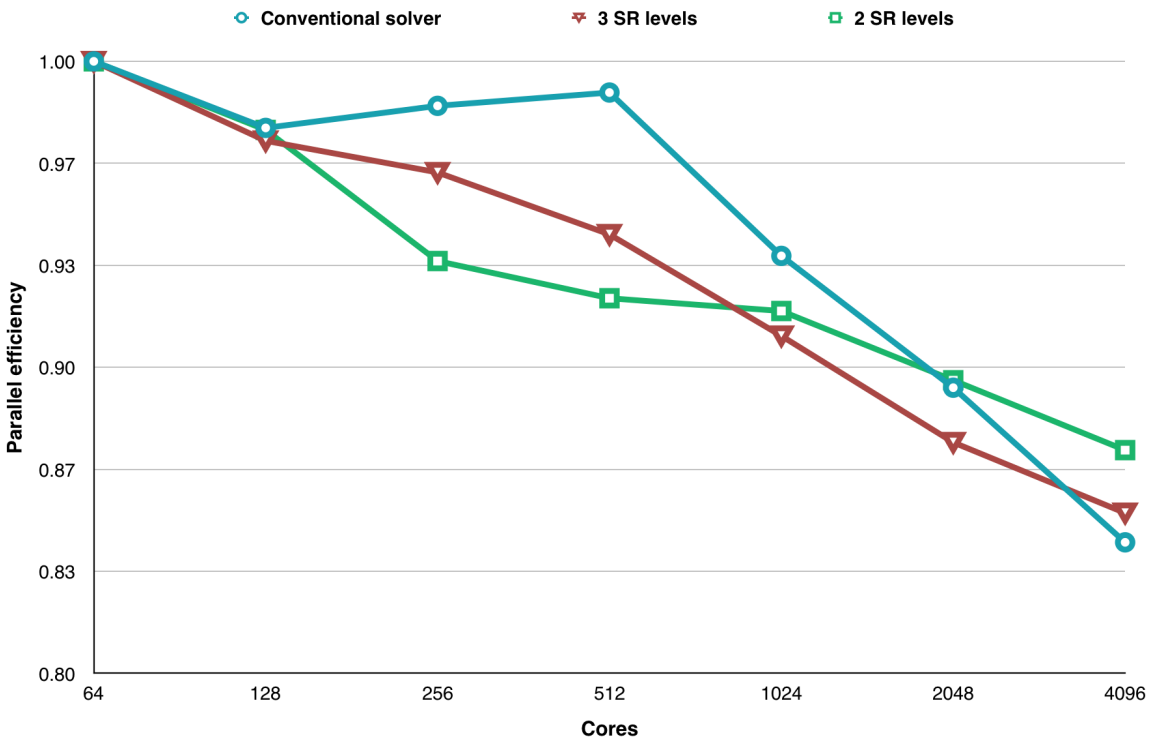

Figure 12: Weak scaling on Edison

For the same study, we show the running time in Figure 13. The plot shows that for small subdomains ($M = 32$), the maximum buffer schedule increases the runtime significantly, especially for three or more segmental refinement levels, because of the computational overhead (cf. 6.3). Adams [4] finite volume solver, which is more memory bandwidth-intensive, achieved better runtime results with the dynamic buffer schedule on Edison, but the segmental refinement solver also does not outcompete

conventional full multigrid.

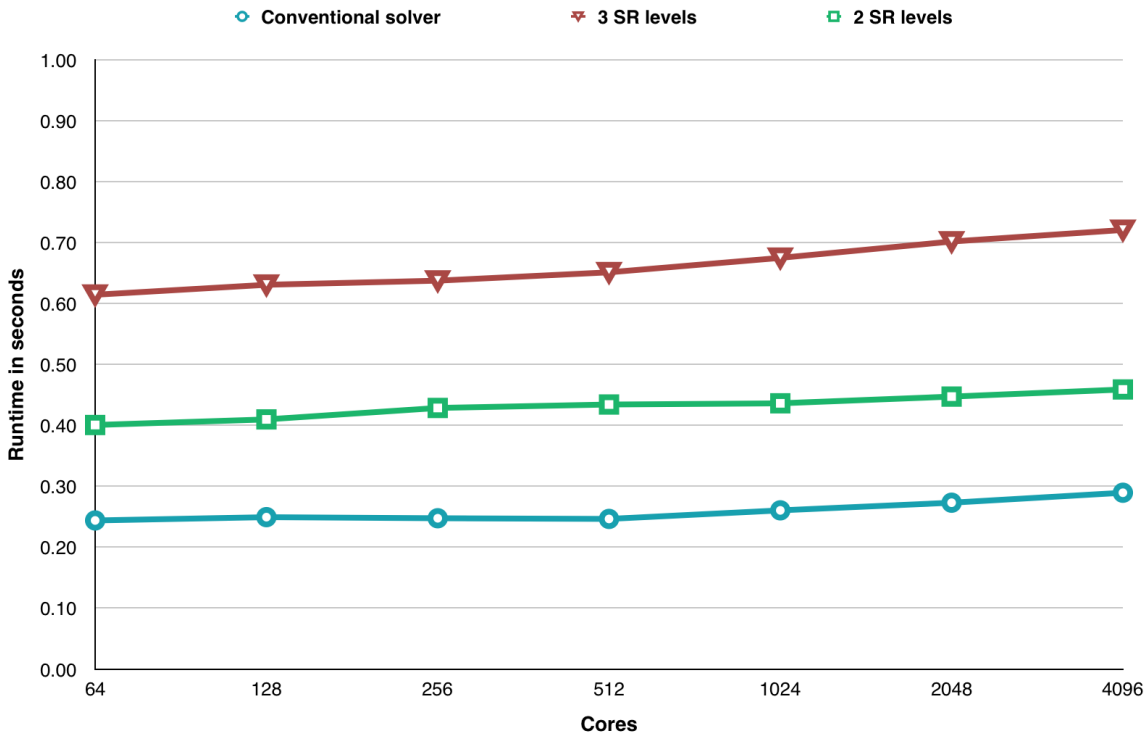

Figure 13: Runtime on Edison

CHAPTER 8

CONCLUSION

We have showed that segmental refinement can be used to avoid inter-process communication on the finest grids in a parallel multigrid F-Cycle algorithm. The first version of the vertex-centered finite element implementation is publicly accessible[1]. The implementation is currently under development; the second version will support segmental refinement with higher-order discretization ($\mathcal{Q}_2$ elements) as well as a dynamic buffer schedule. A preliminary version of one-level evanescent data segmental refinement - a low-memory version of the method - is also available.

An interesting approach for future work would be to quantify savings in time from the avoided communication vs. the overhead in computational work for different machine models that either reward compute-intense or memory bandwidth-intense applications. This may allow to specify the range of machine models that are likely to perform better for conventional or segmental refinement multigrid. The scaling results that we observed indicate that a competitive segmental refinement solver would probably require a dynamic buffer schedule.

Additional research is required to further investigate the dependency of the accuracy on the segmental refinement parameters, particularly with regard to the subdomain size on the transition level, where we observed a different dependence than Adams for the cell-centered finite volume solver. This could be approached by examining the effect of higher-order prolongation to set frozen nodes in the buffer region and analyzing the propagation of errors in the buffer depending on different grid transfer operators and the local smoothness of the solution. A comparable study for nonlinear test problems may provide new insight as well.

[1] http://bitbucket.org/shenneking/hpgmg-fe-sr

APPENDIX A

FINITE ELEMENT IMPLEMENTATION

A.1 $\mathcal{Q}_1$ Reference Element

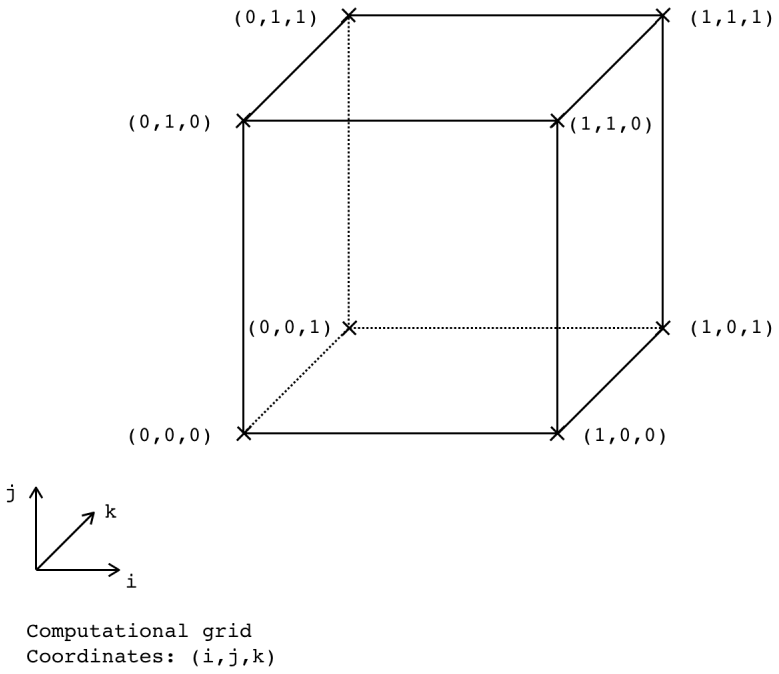

Figure 14: $\mathcal{Q}_1$ reference element

A.2 Star Forests

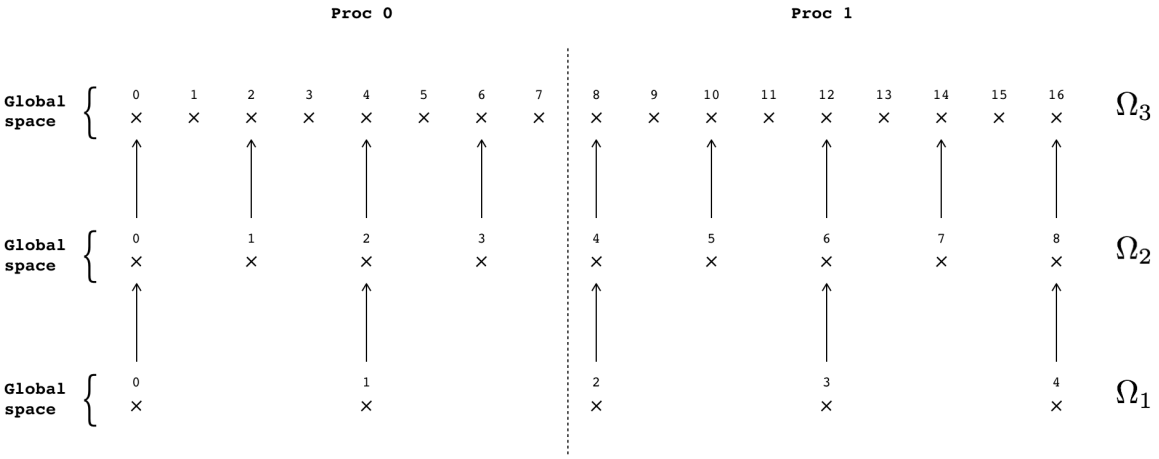

Figure 15: Star Forest for global injection

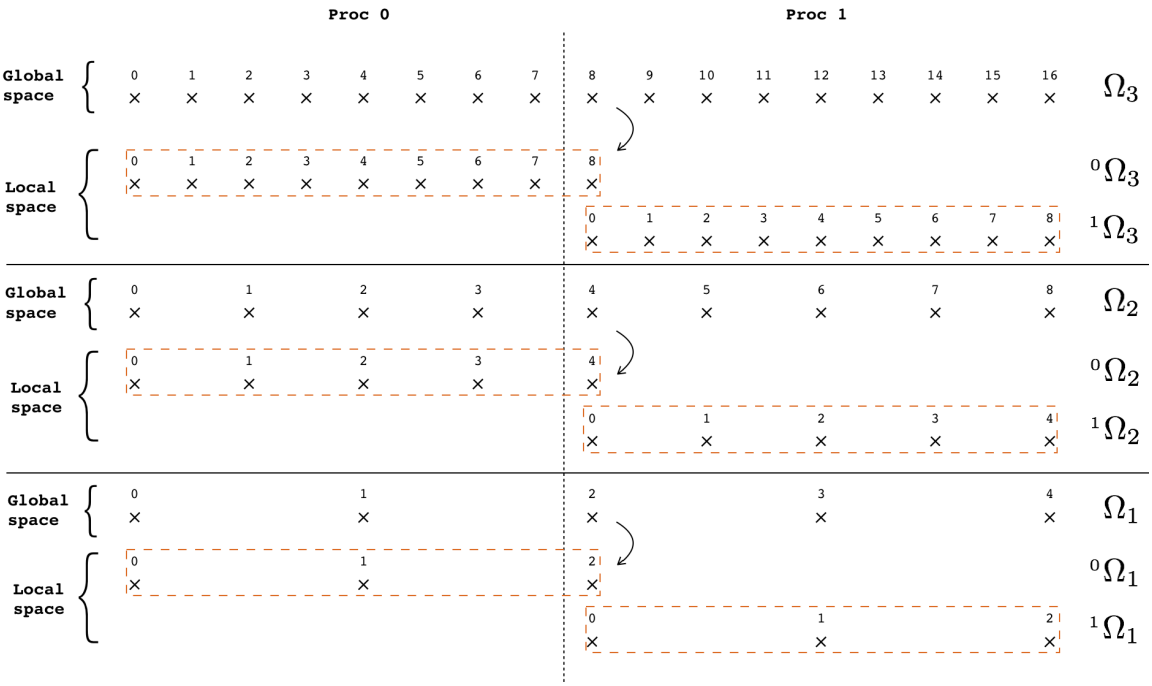

Figure 16: Star Forest for neighbor scatter: exchange of ghost regions

A.3 Poisson Test Problems

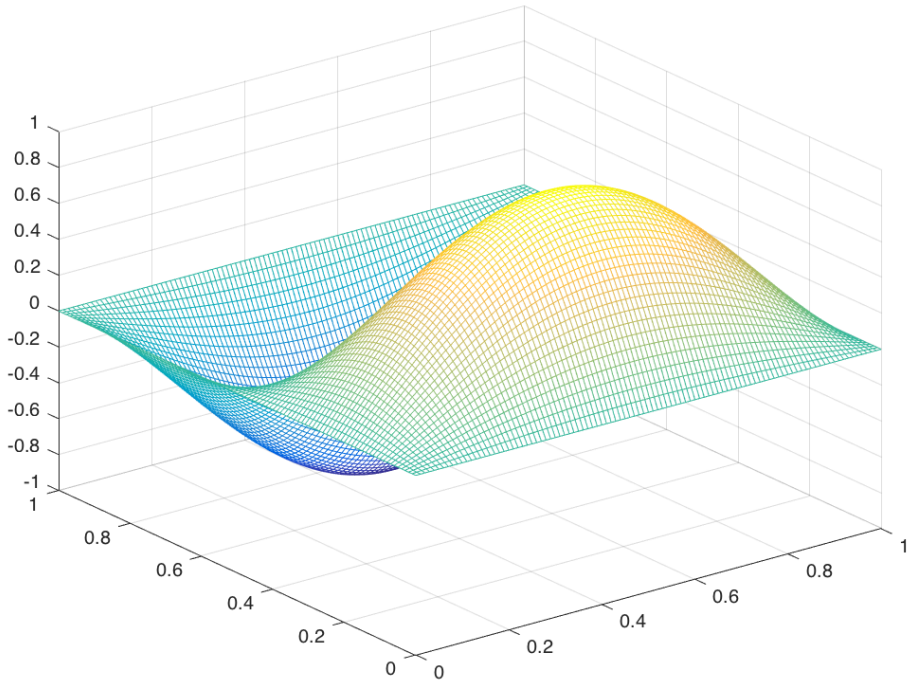

Figure 17: Solution of the two-dimensional *sine* problem

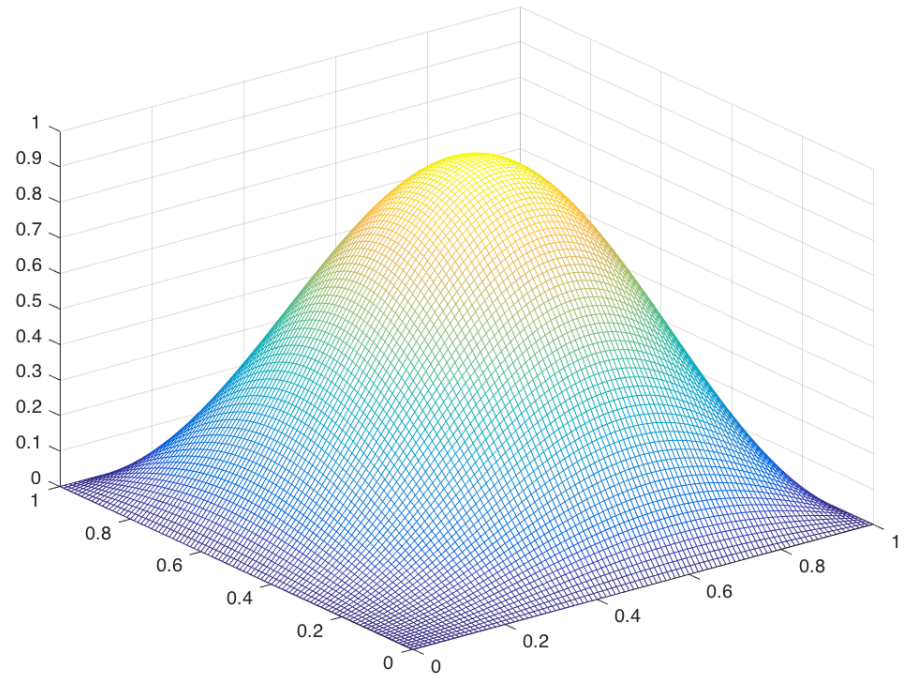

Figure 18: Solution of the two-dimensional *hump* problem

REFERENCES

[1] Adams, M., "Hpgmg 1.0: a benchmark for ranking high performance computing systems," 2014.

[2] Adams, M., Brezina, M., Hu, J., and Tuminaro, R., "Parallel multigrid smoothing: polynomial versus gauss–seidel," *Journal of Computational Physics*, vol. 188, no. 2, pp. 593–610, 2003.

[3] Adams, M. F., "A low memory, highly concurrent multigrid algorithm," *arXiv preprint arXiv:1207.6720*, 2012.

[4] Adams, M. F., Brown, J., Knepley, M., and Samtaney, R., "Segmental refinement: A multigrid technique for data locality," *arXiv preprint arXiv:1406.7808*, 2014.

[5] Bernert, K., Jung, M., and Rüde, U., "Multigrid tau–extrapolation for nonlinear partial differential equations," in *ICOSAHOM*, vol. 95, pp. 543–557.

[6] Brandt, A., "Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems," in *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, pp. 82–89, Springer, 1973.

[7] Brandt, A., "Multi-level adaptive solutions to boundary-value problems," *Mathematics of computation*, vol. 31, no. 138, pp. 333–390, 1977.

[8] Brandt, A., "Multi-level adaptive techniques (MLAT) for partial differential equations: ideas and software," *Mathematical Software*, vol. 3, pp. 277–318, 1977.

[9] Brandt, A., "Multi-level adaptive techniques (MLAT) for singular-perturbation problems," 1978.

[10] Brandt, A., "Barriers to achieving textbook multigrid efficiency (tme) in cfd," 1998.

[11] Brandt, A. and Diskin, B., "Multigrid solvers on decomposed domains," *Contemporary Mathematics*, vol. 157, pp. 135–135, 1994.

[12] Brandt, A. and Livne, O. E., *Multigrid techniques: 1984 guide with applications to fluid dynamics*, vol. 67. SIAM, 2011.

[13] Chow, E., Falgout, R. D., Hu, J. J., Tuminaro, R. S., and Yang, U. M., "A survey of parallelization techniques for multigrid solvers," *Parallel processing for scientific computing*, vol. 20, pp. 179–201, 2006.

[14] Gholaminejad, A., Malhotra, D., Sundar, H., and Biros, G., "FFT, FMM, or Multigrid? A comparative study of state-of-the-art Poisson solvers," in *Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium*, vol. 2014, 2014.

[15] Gmeiner, B., Rüde, U., Stengel, H., Waluga, C., and Wohlmuth, B., "Towards textbook efficiency for parallel multigrid," *Numerical Mathematics: Theory, Methods and Applications*, vol. 8, no. 01, pp. 22–46, 2015.

[16] Hülsemann, F., Kowarschik, M., Mohr, M., and Rüde, U., "Parallel geometric multigrid," in *Numerical Solution of Partial Differential Equations on Parallel Computers*, pp. 165–208, Springer, 2006.

[17] Mohr, M. and Rüde, U., "Communication reduced parallel multigrid: Analysis and experiments," 1998.

[18] Trottenberg, U., Oosterlee, C. W., and Schuller, A., *Multigrid*. Academic press, 2000.