Parallel Efficiency-based Adaptive Local Refinement

by

Lei Tang

B.S., University of Science and Technology of China, 2005

M.S., University of Waterloo, 2007

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Applied Mathematics

2010

This thesis entitled:
Parallel Efficiency-based Adaptive Local Refinement
written by Lei Tang
has been approved for the Department of Applied Mathematics

Prof. Thomas A. Manteuffel

Prof. Stephen F. McCormick

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Tang, Lei (Ph.D., Applied Mathematics)

Parallel Efficiency-based Adaptive Local Refinement

Thesis directed by Prof. Thomas A. Manteuffel

New adaptive local refinement (ALR) strategies are developed, the goal of which is to reach a given error tolerance with the least amount of computational cost. This strategy is especially attractive in the setting of a first-order system least-squares (FOSLS) finite element formulation in conjunction with algebraic multigrid (AMG) methods in the context of nested iteration (NI). To accomplish this, the refinement decisions are determined based on minimizing the predicted 'accuracy-per-computational-cost' efficiency (ACE). The nested iteration approach produces a sequence of refinement levels in which the error is equally distributed across elements on a relatively coarse grid. Once the solution is numerically resolved, refinement becomes nearly uniform. Efficiency of the algorithms are demonstrated through a 2D Poisson problem with steep gradients, and 2D reduced model of the incompressible, resistive magnetohydrodynamic (MHD) equations.

Accommodations of the ALR strategies to parallel computer architectures involve a geometric binning strategy to reduce communication cost. Load balancing begins at very coarse levels. Elements and nodes are redistributed using parallel quad-tree structures and a space filling curve. This automatically ameliorates load balancing issues at finer levels. Numerical results produced on Frost, the NCAR/CU Blue Gene/L supercomputer, are presented for a 2D Poisson problem with steep gradients, a 2D backward facing step incompressible Stokes equations and Navier-Stokes equations. The NI-FOSL-AMG-ACE approach is able to provide highly resolved approximations to rapidly varying solutions using a small number of work units. Excellent weak and strong scalability of parallel ALR are demonstrated on up to 4,096 processors for problems with up to 15 million biquadratic elements.

Dedication

To Mom and Dad.

Acknowledgements

This work was influenced by a great number of people. First, I want to thank my advisors, Tom Manteuffel and Steve McCormick, for their advice and support. I have learned from them how to become a motivated, dedicated, and professional researcher. They have taught me how to enjoy the excitement of applied mathematics and how to think independently and creatively. Additionally, I have also learned from them how to communicate and collaborate with other people and how to enjoy life outside of research. Thank you very much for all of the bike rides and hikes, and the sharing of puns during the group meetings, (although it is quite hard to understand some of the puns as a foreign national). I would also like to thank John Ruge and Marian Brezina for all their help. Without their patient explanations in great detail of the algorithms, data structures, and bugs, I would never have been able to finish this thesis. I am grateful for all the suggestions to my code. Second, I want to thank my committee member, Xiao-chuan Cai, for teaching me parallel numerical methods for PDEs and for sharing his knowledge and experience with parallel software development. Third, I want to thank my former advisor Hans De Sterck for all his support during my master degree. You introduced me to the area of scientific computing and always encouraged me to go further. Last, I want to send a thank you to the rest of the Grandview gang: to Josh Nolting for his senior advice and preliminary work on this project, to James Adler for teaching me FOSLS and FOSPACK tricks, to Geoff Sanders for discussions on linear solvers, to Christian Ketelsen for correcting my thesis, and to Jacob Schroder, Minho Park, and Kuo Liu for all the insightful discussions and moral support. Finally, I want to thank my parents for raising me and teaching me how to become a righteous man.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

Adaptive finite element methods (AFEMs) are being used extensively to approximate solutions of partial differential equations (PDEs) containing local features; see, e.g., [6, 24, 26, 38, 43]. Consider a PDE, or a system of PDEs, written abstractly as

$$\mathcal{P}u = f \qquad \text{in } \Omega \subset \mathbb{R}^d, \tag{1.1}$$

with $u \in \mathcal{V}$ and appropriate boundary conditions. Let $\mathcal{T}$ be a regular partition [10] of the domain, Ω, into elements. Define the mesh size, $h = \max\{\text{diam}(\tau), \tau \in \mathcal{T}\}$. The refinement process starts on a coarse grid, $\mathcal{T}_0$ (level $= 0$), and iteratively refines and approximates the PDE on levels $\ell = 1, 2, ...$ until the error satisfies a certain criterion. At each level, some elements are refined in h by splitting them into sub-elements, and some are refined in p by increasing the element order. In [32, 33], this concept is described in the following form:

$$\text{Solve} \rightarrow \text{Estimate} \rightarrow \text{Mark} \rightarrow \text{Refine}. \tag{1.2}$$

The goal of adaptive mesh refinement is to construct a sequence of grids that converge to an optimal grid. A grid, $\mathcal{T}_{opt}$, (or the associated finite element space $\mathcal{V}_{opt}$), is called optimal if it is capable of approximating the solutions of a given PDEs to certain accuracy with the least number of degree of freedom (DOF). Equivalently, denoted by $u_{\mathcal{T}} \in \mathcal{V}_{\mathcal{T}}$ the numerical solutions to a given system of PDEs. Let $\mathcal{E}_{\mathcal{T}} = ||u - u_{\mathcal{T}}||_{\mathcal{V}}$ be norm of the error. Then, the $\mathcal{T}_{opt}$ is considered as an **optimal grid** if for a given number of DOF, write $\mathbb{N} > 0$,

$$\mathcal{T}_{opt} = \arg \min_{|\mathcal{T}|=\mathbb{N}} ||u - u_{\mathcal{T}}||_{\mathcal{V}}. \tag{1.3}$$

Here, $|\mathcal{T}|$ is the dimension of the finite element space, $\mathcal{V}_{\mathcal{T}}$, that is associated with $\mathcal{T}$. In one dimension, it has been proved that this is accomplished by equally distributing the error over all elements; [26]. It is believed that this also holds for higher dimensions. Based on this premise, a simple method to mark elements for refinement was introduced by Babuška [26]: an element is marked for refinement if its local error is within a certain factor of the largest local error at that level. In [24], a more complicated algorithm, called the threshold-based marking, was proposed:

ALGORITHM 1 (Threshold-based Marking)**.** *Given a parameter* $0 < \theta \leq 1$*, construct a minimal subset* $\hat{\mathcal{T}}$ *of* $\mathcal{T}$ *such that*

$$\sum_{\tau \in \hat{\mathcal{T}}} \epsilon_\tau^2 \geq \theta \sum_{\tau \in \mathcal{T}} \epsilon_\tau^2, \tag{1.4}$$

and mark all elements in $\hat{\mathcal{T}}$ *for refinement.*

The AFEMs in [32, 33] start with this approach, then further mark elements based on oscillation terms. This marking approach often produces satisfactory results. With a proper choice of θ, one can establish the convergence of the AFEM as well as near optimality of the finest grid. However, the real computational cost was not addressed. Also, the proper choice of θ is different for various problems and unknowns **a priori**. We argue that the goal of AMR should take into account the real computational cost, meaning, the goal should be to achieve the optimal grid with the least amount of work. To do that would require the value of θ to be free to change on each level. A large fraction of elements may need to be refined at the coarser levels when the grids are too coarse to resolve the local features of the solution. Then, at intermediate levels, refinement should concentrate on the elements containing relatively large error. Lastly, at finer levels, once the error is equally distributed, near global refinement is preferred.

A new approach, described in [8], was developed to address this issue. The algorithm refines elements that minimize a 'work-times-error-reduction' efficiency factor (WEE) at each refinement level. Later, in [22], it was shown that the WEE algorithm was inefficient for problems with spatial dimension, d, less than the polynomial degree, p, of the finite element space. Another

algorithm, which determined the fraction of elements to be refined, r, by optimizing the 'accuracy-per-computational-cost' efficiency factor (ACE) was proposed and analyzed in [3,22,34]. The ACE algorithm was applied to first-order system least-sqares (FOSLS) finite element methods in the conjunction with algebraic multigrid (AMG) in the context nested iteration (NI). The NI-FOSLS-AMG approach yields measures that allows us to estimate the error reduction and computational cost, which can be used to make the refinement decisions based on optimizing computational efficiency. The results show that the ACE algorithm is capable of effectively and efficiently detecting the solution's local features.

Even with efficient and effective AMR, more computing resources are almost always desired to achieve the ever-increasing demands on solution resolution. Currently, massively parallel distributed memory machines are being built to accommodate this continual need for greater computing power. For this reason, this thesis focuses developing the parallel efficiency-based adaptive refinement techniques and the software implementation of the NI-FOSLS-AMG approach in parallel. The main contribution of this thesis is to provide the parallel PDE solver package, Parallel FOSPACK (*pFOSPACK*). The package uses the FOSLS methodology for discretization, and employs AMG, (accelerate by Conjugate Gradient (CG)), to solve the discretized equations. The design goal is to develop a scalable, efficient, and easy-to-use PDE solver to support numerical simulations for large scale applications. Extending efficiency-based refinement strategies in parallel utilizes geometric binning strategies. Elements are grouped into bins based on local error, then refinement decisions can be made based on treating each bin as abstract element. Once a bin is marked for refinement, all elements in that bin are refined. This thesis demonstrates that the parallel ACE(pACE)-like algorithms based on geometric binning produces results similar to the original serial algorithms, but greatly reduces the communication cost. Scaling dynamical AMR in parallel is challenging. Difficulties in communications, load balancing, and mesh interactions must be overcome. In pFOSPACK, load balancing starts on coarser grids, where the computation and communication are relatively cheap. At each refinement level, a space filling curve (SFC) and parallel tree structures are used to redistribute nodes and elements among processors in order to

preserve locality of the new partition and, thus, reduce communication cost. This thesis shows that the NI-FOSLS-AMG-pACE algorithm yields equal distribution of error on finer levels, which leads to near uniform refinement. Uniform refinement requires no further load balancing. Tests on Frost, the CU/NCAR Blue Gene/L super computer, demonstrate excellent strong and weak scalabilities.

This thesis is organized as follows. In chapter 2, the basic concepts of the NI-FOSLS-AMG approach are described. Notations used in this thesis are also introduced. The efficiency-based adaptive local refinement strategies are formulated in chapter 3. Numerical results of applying the NI-FOSLS-AMG-ACE approach to a 2D Poisson equation with steep gradients and a 2D reduced model of the incompressible, resistive magnetohydrodynamic (MHD) equations are discussed. We show that, by using ALR strategies, we are able to resolve the physics using only 10% of the computational cost used to approximate the solutions on a uniformly refined mesh within the same error tolerance. Next, accommodation of the efficiency-based ALR strategies in parallel is discussed in chapter 4. Details of mesh partitioning, load balancing strategies, and communication issues come afterward. Numerical efficiency and parallel scalability of the NI-FOSLS-AMG-pACE approach are demonstrated by various tests on Frost. Lastly, conclusions are formulated in chapter 5.

Chapter 2

Preliminaries

This chapter describes the basic concepts behind the NI-(Newton)-FOSLS-AMG approach and introduces notations used in the rest of the thesis. The FOSLS approach yields locally sharp error indicator which is almost computationally free and fits in the ALR framework. Using AMG in the context of nested iterations, measures can be computed to estimate both error reduction and computational work. This makes NI-FOSLS-AMG a great candidate for developing efficient multi-level adaptive PDE solvers. This chapter provides theoretic foundations for developing efficiency-based ALR schemes in Chapter 3. Although we focus on NI-FOSLS-AMG, it is worthwhile to point out that ACE-like ALR schemes can be used together with other descretization methods and linear solvers.

2.1 Discretization Methods

2.1.1 FOSLS methodology

First-order system least squares (FOSLS) is a special type of finite element method that reformulates a PDE as a system of first-order equations and poses the problem as a minimization of a functional. Here, the first-order differential terms appear quadratically and, thus, the functional norm is equivalent to a norm meaningful to the problem. To illustrate the basic concepts of FOSLS, consider the PDE written abstractly in (1.1). Introducing new variables, we arrive at a first-order system:

$$\mathcal{L}_i\mathbf{u} = f_i, \qquad i = 1, 2, ..., M. \tag{2.1}$$

Assuming $f_i \in L^2(\Omega)$ and $\mathcal{L}_i$ is linear, consider the associated FOSLS functional given by

$$\mathcal{G}(\mathbf{u}, \mathbf{f}) = \sum_{i=1}^{M} ||\mathcal{L}_i \mathbf{u} - f_i||_{0,\Omega}^2, \tag{2.2}$$

where $||u||_{0,\Omega} = \sqrt{\int_\Omega |u|^2}$ is the L^2-norm. The minimization problem is

$$\mathbf{u} = \arg\min_{\mathbf{v} \in \mathcal{V}} \mathcal{G}(\mathbf{v}; \mathbf{f}). \tag{2.3}$$

Here, $\mathcal{V}$ is an appropriate Hilbert space, usually (equivalent to) a product of H^1 spaces. The minimizer $\mathbf{u}$ satisfies $\mathcal{G}'(\mathbf{u})[\mathbf{v}] = \mathbf{0}$, which is the Fréchet derivative of $\mathcal{G}$ in the direction $\mathbf{v} \in \mathcal{V}$; that is

$$\mathcal{G}'(\mathbf{u})[\mathbf{v}] = \lim_{\alpha \to 0} \frac{\mathcal{G}(\mathbf{u} + \alpha \mathbf{v}; \mathbf{f}) - \mathcal{G}(\mathbf{u}; \mathbf{f})}{\alpha}. \tag{2.4}$$

This yields the equivalent weak form:

$$\begin{aligned} &\text{find } \mathbf{u} \in \mathcal{V} \text{ such that} \\ &\qquad < \mathcal{L}\mathbf{u}, \mathcal{L}\mathbf{v} > = < \mathbf{f}, \mathcal{L}\mathbf{v} > \qquad \forall\ \mathbf{v} \in \mathcal{V}, \end{aligned} \tag{2.5}$$

where $< \mathcal{L}\mathbf{u}, \mathcal{L}\mathbf{v} > = \sum_i < \mathcal{L}_i \mathbf{u}, \mathcal{L}_i \mathbf{v} >$ is the usual L^2 inner product in the product space.

In many cases, under general regularity assumptions, the weak form is continuous and coercive in $\mathcal{V}$; see, e.g., [16, 17]. That is, there exist positive constant c_1 and c_2 such that

$$\begin{aligned} &\text{continuity} \qquad < \mathcal{L}\mathbf{u}, \mathcal{L}\mathbf{v} > \leq c_2 ||\mathbf{u}||_{\mathcal{V}} ||\mathbf{v}||_{\mathcal{V}} \qquad \forall\ \mathbf{u},\ \mathbf{v} \in \mathcal{V}, \\ &\text{coercivity} \qquad < \mathcal{L}\mathbf{u}, \mathcal{L}\mathbf{u} > \geq c_1 ||\mathbf{u}||_{\mathcal{V}}^2 \qquad \forall\ \mathbf{u} \in \mathcal{V}. \end{aligned} \tag{2.6}$$

In other words, the FOSLS functional $\mathcal{G}(\mathbf{u}; \mathbf{f})$ is "elliptic" with respect to the $\mathcal{V}$ norm; That is, its homogeneous part, $\mathcal{G}(\mathbf{v}; \mathbf{0})$, is equivalent to the squared $\mathcal{V}$ norm:

$$c_1 \leq \frac{\mathcal{G}(\mathbf{v}; \mathbf{0})}{||\mathbf{v}||_{\mathcal{V}}^2} \leq c_2 \qquad \forall\ \mathbf{v} \in \mathcal{V}. \tag{2.7}$$

By The Riesz Representation Theorem, ellipticity guarantees the existence and uniqueness of the solution, $\mathbf{u}$. Let $\mathcal{V}^h \subset \mathcal{V}$ be a finite-dimensional subspace of $\mathcal{V}$. Often, it consists of continuous piecewise polynomials. Note that the discretization can be written as the minimization problem

$$\mathbf{u}^h = \arg\min_{\mathbf{v}^h \in \mathcal{V}^h} \mathcal{G}(\mathbf{v}^h; \mathbf{f}). \tag{2.8}$$

Well-posedness of (2.8) follows directly from the ellipticity, (2.6). Therefore, the FOSLS formulation is not restricted by any strict stability conditions such as the inf-sup or Ladyzhenskaya-Babška (LBB) condition; see, e.g., [10, 13]. While not a necessary condition, if $\mathcal{V}$ is a product of H^1 spaces, then ellipticity also enables an optimal multigrid solver of the discrete system [17]. That is, standard multigrid solvers converge with factors, $0 < \rho < 1$, bounded uniformly in mesh size, h.

The introduction of the new dependent variables increases the number of DOF, much like mixed finite element methods. However, unlike mixed methods, FOSLS yields a symmetric positive definite algebraic system that is, in general, more amenable to multilevel solution techniques.

2.1.2 Linearization

Newton-like linearization processes are often used for solving nonlinear PDEs. Starting with an initial guess, one computes the next iterate by solving a linear approximation of the nonlinear problem in a small neighborhood of the current iterate. If the initial guess is in the attraction basin, then a sequence of iterates converges to the exact solution of the nonlinear problem. In the context of least-squares methods, depending on where the linearization appears in the solution process, two types of methods are usually used. The first method, called Newton-FOSLS, linearizes the PDEs in the neighborhood of the current approximate and then applies the least-squares method to the linearized problem. The second method, called FOSLS-Newton, first constructs a nonlinear least-square problem and then approximates this with a linear problem (see [20, 37]). In literature, the Newton-FOSLS corresponds to Gauss-Newton-like approach, and the FOSLS-Newton to the Full-Newton approach. These two approaches behave similarly near the exact solution of the nonlinear problem, so the choice is based mostly on convenience of implementation. Because of the reduced expense of Gauss-Newton-like approaches, they tend to be more popular in practice. Similarly, we choose the Newton-FOSLS approach over FOSLS-Newton because of its simplicity. A more detailed description of the Newton-FOSLS method is given below. One can refer to [37] and references therein for more details.

Newton-FOSLS

To illustrate the concept of Newton-FOSLS, consider a nonlinear first-order differential operator, $\mathcal{L} : \mathcal{V} \to \mathcal{V}_{\mathcal{L}}$, where $\mathcal{V}$ is often equivalent to a subspace of a product of $H^{1+\delta}(\Omega)$ spaces for $\delta \in (0, 1)$. Here, we take the solution space in $H^{1+\delta}$ instead of purely H^1 since nonlinear terms often consists of a product of first order derivatives of $\mathbf{u} \in \mathcal{V}$. In $\mathbb{R}^2$, Sobolev embedding theorem [1] implies that

$$||\mathbf{u}||_{\infty,\Omega} \leq C||\mathbf{u}||_{1+\delta,\Omega}, \qquad \forall \mathbf{u} \in H^{1+\delta}(\Omega), \tag{2.9}$$

where $||\cdot||_{1+\delta,\Omega}$ is the fractional Sobolev space norm. This equality ensures that $\mathcal{V}_{\mathcal{L}} \subset L^2$ so that the usual L^2 least-square functional can be applied. To solve the nonlinear PDE associated with $\mathcal{L}$:

$$\mathcal{L}(\mathbf{u}) = \mathbf{f}, \qquad \text{in } \Omega, \tag{2.10}$$

one can first linearize (2.10) in a neighborhood of a given initial guess, $\mathbf{u}_0 \in \mathcal{V}$, by truncating high order terms in its Taylor series:

$$\mathcal{L}(\mathbf{u}) \approx \mathcal{L}(\mathbf{u}_0) + \mathcal{L}'(\mathbf{u}_0)[\mathbf{u} - \mathbf{u}_0], \tag{2.11}$$

where $\mathcal{L}'(\mathbf{u}_0)[\mathbf{u} - \mathbf{u}_0]$ is the Fréchet derivative of $\mathcal{L}$ at $\mathbf{u}_0$ in the direction $[\mathbf{u} - \mathbf{u}_0]$. This leads to a linear approximation of the nonlinear problem at $\mathbf{u}_0$:

$$\mathcal{L}'(\mathbf{u}_0)[\delta\mathbf{u}] = \mathbf{f} - \mathcal{L}(\mathbf{u}_0). \tag{2.12}$$

The least-square functional for (2.12) becomes

$$\mathcal{G}(\mathbf{u}_0 + \delta\mathbf{u}; \mathbf{f}) = ||\mathcal{L}'(\mathbf{u}_0)[\delta\mathbf{u}] - (\mathbf{f} - \mathcal{L}(\mathbf{u}_0))||_0^2. \tag{2.13}$$

Minimizing the linearized functional (2.13) yields

$$< \mathcal{L}'(\mathbf{u}_0)[\delta\mathbf{u}], \mathcal{L}'(\mathbf{u}_0)[\mathbf{v}] > = < \mathcal{L}'(\mathbf{u}_0)[\mathbf{v}], \mathbf{f} - \mathcal{L}(\mathbf{u}_0) > \qquad \forall\ \mathbf{v} \in \mathcal{V}. \tag{2.14}$$

The next approximation is obtained by adding the correction to the current iterate:

$$\mathbf{u}_1 = \mathbf{u}_0 + \delta\mathbf{u}. \tag{2.15}$$

The n-th Newton step can be written abstractly as

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \mathcal{L}'(\mathbf{u}_n)^{-1} \left(\mathbf{f} - \mathcal{L}(\mathbf{u}_n)\right). \tag{2.16}$$

The discrete problem is to apply FOSLS approach to solve for $\delta\mathbf{u}_n^h \in \mathcal{V}^h \subset \mathcal{V}$ such that

$$< \mathcal{L}'(\mathbf{u}_n^h)[\delta\mathbf{u}_n^h], \mathcal{L}'(\mathbf{u}_n^h)[\mathbf{v}^h] > = < \mathcal{L}'(\mathbf{u}_n^h)[\mathbf{v}^h], \mathbf{f} - \mathcal{L}(\mathbf{u}_n^h) > \qquad \forall\ \mathbf{v}^h \in \mathcal{V}^h. \tag{2.17}$$

If the weak form in (2.17) is continuous and coercive, then there exists a unique solution.

The convergence of Newton-FOSLS strongly depends on a good initial guess. An efficient and effective technique of obtaining initial guesses is the strategy of nested iteration. Details are discussed in the next section.

2.2 Algebraic Solver

Nested iteration (NI), or full-multigrid [14] (FMG) as it is called in the multigrid context, involves starting the solution process on a relatively coarse grid, where the computational cost is relatively cheap. The solution on the coarse grid is used as an initial guess for the problem on the next finer grid. Since the objective on each grid is to minimize the FOSLS functional, the coarse-grid solution should provide a good starting guess. On each refinement level, solving discrete minimization problem (2.8) involves fast iterative solvers applied to the matrix equations. If the FOSLS functional is equivalent to a product H^1 norm, then there exists an optimal multilevel solution algorithm [45]. Experience shows that, in this context, AMG also yields an approximate solution to the discrete equations associated with quasi-uniform grids in optimal time with convergence factor, ρ, bounded uniformly below 1, independent of mesh size h. AMG methods, together with the NI strategy and local refinement, provide a powerful approach for approximating solutions of PDEs. Numerical and theoretical results confirm that the overall cost of such a scheme resides predominantly in the cost of the finest-level processing. The total cost is usually cheaper than solving the problem directly on the finest grid, which generally is not even known in advance.

The NI strategy presents a special opportunity for AMG methods. In general, AMG requires a substantial setup phase. The NI approach with local refinement yields a hierarchy of quasi-

nested block-structured grids. That is, the coarsest grid may be irregular, but subsequent grids are increasingly more structured. This hierarchy, together with AMG, may reduce or eliminate the need for a setup phase at each level. This will be investigated in future work.

The NI strategy is efficient and effective for obtaining a good initial guess for Newton-FOSLS. The computed solution from the previous coarser level minimizes the nonlinear functional in the coarser space. It is likely to lie into a small neighborhood of the solution on the next finer level. Theory developed in [20] shows that, under rather mild hypotheses, for a sufficiently fine coarsest grid, the NI approach based on one Newton step per refinement level using a fixed number of multigrid cycles produces a final approximation to the solution of the first-order system that is H^1 accurate to the level of discretization error. However, fixing number of multigrid cycles and number of Newton steps at each refinement level might not be efficient in terms of accuracy-per-computational-cost. In fact, at some point, doing more work on the coarser grid might not be as efficient as moving to the finer grid. For example, to locally refine a portion of the domain containing a large amount of error might give more error reduction than continuing to solve on the coarse grid. Similarly, on the same level, it might be more efficient to relinearize and perform a new Newton step than to continue to resolve the current Newton step. The number of multigrid cycles, as well as Newton steps, should vary depending on how the grids are refined on each refinement level. Strategies to decide how accurately to solve each Newton step and when to stop the Newton iteration and proceed to the next finer grid are described in [2]. Results in [2–5], and later in this thesis, show the NI-Newton-FOSLS-AMG approach with ALR and stopping criteria based on accuracy-per-computational-cost is able to solve complicated system of PDEs, such as Navier-Stokes and MHD, in a small amount of work units. We briefly describe the NI strategy below; see [2] for more details.

The basic principle is to have the most accuracy-per-computational cost. Let $\mathcal{G}$ and G denote the nonlinear functional and linearized functional at the current Newton step, respectively. With the linearized functional and AMG convergence factor, one can estimate how much error of the linear system is being reduced and at what cost. Similarly, by checking the nonlinear functional and

the number of Newton steps being performed, one can estimate the accuracy-per-computational cost for solving the nonlinear system. These estimates are then used to establish stopping parameters for linear as well as nonlinear iterations.

For linear problems, the idea is to stop linear iterations on the coarse grid once moving to the fine grid is more efficient. Denote by G_*^{2h} and G_*^h the discretization error, measured by the functional, on grids h and $2h$, respectively. Let $\mathbf{u}_i^{2h}$ be the approximate solution at the i^{th} iteration and G_i^{2h} be the associated numerical error, measured by the functional. Furthermore, denote by d the dimension of the problem, i.e., $\mathbb{R}^2$ or $\mathbb{R}^3$. Keep in mind that the AMG convergence rate, ρ, is considered as a constant on grids $2h$ and h. On grid $2h$, one can write the linear functional at the i^{th} iteration as

$$G_i^{2h} = (1 + \epsilon_i^{2h}) G_*^{2h}, \tag{2.18}$$

where $\epsilon_i^{2h} G_*^{2h}$ is the square of the algebraic error. Assume the algebraic error converges with convergence factor ρ. Then, one more iteration would give

$$G_{i+1}^{2h} \approx (1 + \rho^2 \epsilon_i^{2h}) G_*^{2h}. \tag{2.19}$$

Instead of doing the $(i+1)^{\text{st}}$ linear iteration, using $\mathbf{u}_i^{2h}$ as the initial guess on grid h, and performing one iteration would give

$$G_1^h \approx (1 + \rho^2 \epsilon_0^h) G_*^h. \tag{2.20}$$

Doing an AMG cycle on grid h is more expensive than on grid $2h$. What should be used in (2.20) is the effective reduction

$$\rho^{\frac{1}{W_h / W_{2h}}}, \tag{2.21}$$

where $\frac{W_h}{W_{2h}}$ is the work ratio between grid h and $2h$. For uniform refinement it is $\frac{1}{2^{2d}}$. For ALR, it is given in the next chapter. The effective error reduction moving to the next grid is written

$$G_1^h \approx \left(1 + \left(\rho^{\frac{1}{W_h / W_{2h}}}\right)^2 \epsilon_0^h\right) G_*^h. \tag{2.22}$$

If $G_1^h \leq G_{i+1}^{2h}$, then it is more efficient to move to grid h.

The problem remains to determine the particular ϵ_i^{2h} such that this happens. We call this number ϵ_s^{2h}. In [2], the relation between the algebraic error on grid $2h$, $\epsilon_s^{2h}G_*^{2h}$, algebraic error on grid h, $\epsilon_0^h G_*^h$, and the difference between the discrete error among grids, $G_*^h - G_*^{2h}$, is written

$$\epsilon_s^{2h}G_*^{2h} + \left(G_*^{2h} - G_*^h\right) = \epsilon_0^h G_*^h. \tag{2.23}$$

Using this, one can compute

$$\epsilon_s^{2h} = \frac{\left(\left(\rho^{\frac{1}{W_h/W_{2h}}}\right)^2 - 1\right)(1 - \frac{G_*^h}{G_*^{2h}})}{\rho^2 - \left(\rho^{\frac{1}{W_h/W_{2h}}}\right)^2}. \tag{2.24}$$

The stopping criteria for linear problem is described

ALGORITHM 2. *Stopping Criterion for Linear Problem*

- *Compute* ϵ_s^{2h} *by (2.24)*
- *At each linear iteration, compute* G_i^{2h} *by*

$$G_i^{2h} =< A^{2h}x_i^{2h}, x_i^{2h} > -2 < b^{2h}, x_i^{2h} > +||\mathbf{f}||_0^2. \tag{2.25}$$

  Here $\mathbf{f}$ *is the right hand side of the continuous problem and* $A^{2h}x^{2h} = b^{2h}$ *is the matrix system obtained from the associated discrete problem.*

- *If*

$$G_i^{2h} \leq (1 + \epsilon_s^{2h})G_*^{2h}, \tag{2.26}$$

  then stop linear iteration and go to next refinement level.

Algorithm (2) requires that the AMG convergence rate, ρ, and functional reduction, $\frac{G_*^h}{G_*^{2h}}$, between grids $2h$ and h are known in advance. AMG convergence rate, ρ, can be measured at run time. For uniform refinement, reduction can be estimated using the standard finite element error estimate. As for ALR, reduction from the previous grid can be used. Details are discussed in Chapter 3. Discretization error, G_*^{2h}, is usually unknown, but it can be computed using three consecutive iterations. Details can be found in [2, 4, 5].

Stopping criteria for the nonlinear iteration is rather complicated. The bottom line is that once the current Newton step approximation is within a small fraction of the discrete solution to the full nonlinear problem, then more accuracy-per-computational-cost is gained by moving to a finer grid. We give the algorithm below, see [2, 3, 5] for complete details.

ALGORITHM 3. *Stopping Criterion for Nonlinear Problem*

Let $\mathbf{u}_*^{2h}$ *be the exact solution to the nonlinear problem on grid* $2h$*, and* $\mathbf{u}_i^{2h}$ *be the discrete solution to the* i^{th} *Newton step. At each Newton step*

- *Solve the linearized problem* $\mathcal{L}'(\mathbf{u}_i^{2h})[\delta\mathbf{u}_i^{2h}] = \mathbf{f} - \mathcal{L}(\mathbf{u}^{2h})$ *using FOSLS.*
- *Stop linear iteration according to the stopping criteria in algorithm 2.*
- *Compute the nonlinear functional* $\mathcal{G}(\mathbf{u}_{i+1}^{2h}) = \mathcal{G}(\mathbf{u}_i^{2h} + \delta\mathbf{u}_i^{2h})$ *and the difference functional*

$$\mathcal{G}_d(\delta\mathbf{u}_i^{2h}) = ||\mathcal{L}(\mathbf{u}_i^{2h} + \delta\mathbf{u}_i^{2h}) - \Big(\mathcal{L}(\mathbf{u}_i^{2h}) + \mathcal{L}'(\mathbf{u}_i^{2h})[\delta\mathbf{u}_i^{2h}]\Big)\,||_0^2. \tag{2.27}$$

- *If*

$$\frac{\mathcal{G}_d(\delta\mathbf{u}_i^{2h})}{\mathcal{G}(\mathbf{u}_i^{2h} + \delta\mathbf{u}_i^{2h})} < \eta, \tag{2.28}$$

  then stop solving on grid $2h$ *and move to the next grid. Here* η *is a given tolerance. A typical choice is* $\eta = \epsilon_s$*, where* ϵ_s *is computed by (2.24).*

2.3 A Sharp and Reliable A-posteriori Error Estimate

The second step in the refinement loop is to estimate the error locally. An effective local *a posteriori* error estimate is crucial to adaptive refinement. It needs to be easily computable and accurately approximate the actual error. If the error estimate is reliable, then refining elements with a large error estimate leads to equal distribution of the error, which is crucial to reducing the load balancing cost on fine levels when computing on a parallel machine. In [43], Verfüth breaks *a posteriori* error estimation into four categories:

(1) residual estimation,

(2) solution of local problems,

(3) hierarchical basis error estimates,

(4) averaging methods.

Many AFEMs are based on residual estimates. For example, in [24], the estimate,

$$\epsilon_\tau^2 = \sum_{\Gamma \in \partial\tau \setminus \partial\Omega} d_\Gamma ||[\partial_n u^h]||_{0,\Gamma}^2, \tag{2.29}$$

is used for adaptive refinement of the Garlerkin finite formulation for Poisson equation, where Γ is an edge/face of element τ, d_Γ is the edge/face size, and $[\partial_n u^h]$ is the jump of the normal derivative of u^h across Γ. Later, in [32, 33], a local equation residual is added for solving the advection-diffusion equation

$$\epsilon_\tau^2 = h_\tau^2 ||f + \nabla \cdot A\nabla u^h - \mathbf{b} \cdot \nabla u^h - cu^h||_{0,\tau}^2 + \sum_{\Gamma \in \partial\tau \setminus \partial\Omega} d_\Gamma ||[\partial_n u^h]||_{0,\Gamma}^2. \tag{2.30}$$

Our method, like the methods mentioned above, is also based on residual estimation. Because the functional value is zero at the solution, the FOSLS functional itself is a measure of the total error in a given approximation. It provides both absolute and relative error measures, as well as global and local error estimates, that are much simpler and potentially sharper than conventional error estimators. To illustrate this, for any element $\tau \in \mathcal{T}$, define the local FOSLS functional,

$$\mathcal{G}_\tau(\mathbf{u}^h; \mathbf{f}) = \sum_{i=1}^{M} ||\mathcal{L}_i \mathbf{u}^h - f_i||_{0,\tau}^2. \tag{2.31}$$

Writing $\epsilon_\tau = \sqrt{\mathcal{G}_\tau(\mathbf{u}^h; \mathbf{f})}$, the ellipticity in (2.7) implies that

$$\frac{1}{c_2}\epsilon_\tau^2 = \frac{1}{c_2}\mathcal{G}_\tau(\mathbf{u}^h - \mathbf{u}; 0) \leq ||\mathbf{u}^h - \mathbf{u}||^2_{\mathcal{V},\tau} \tag{2.32}$$

and

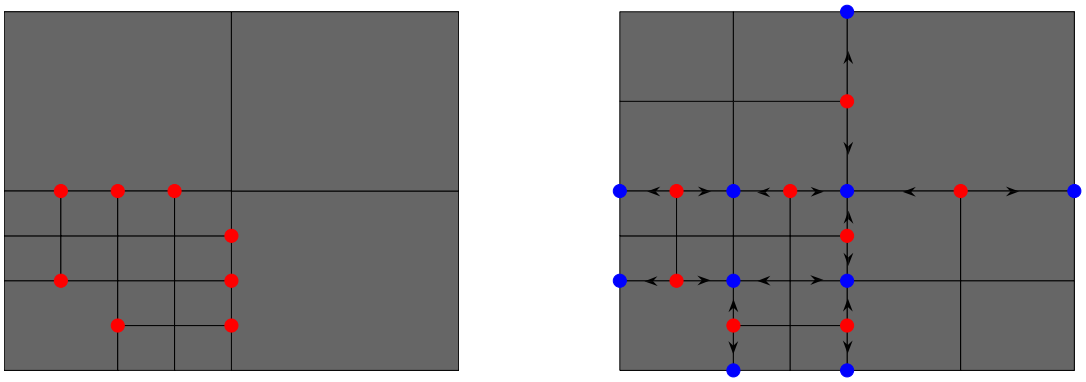
$$||\mathbf{u}^h - \mathbf{u}||^2_{\mathcal{V}} \leq \frac{1}{c_1}\mathcal{G}(\mathbf{u}^h - \mathbf{u}; 0) = \frac{1}{c_1}\sum_{\tau\in\mathcal{T}} \epsilon_\tau^2. \tag{2.33}$$

An error estimator, ϵ_τ, that satisfies an inequality of type (2.32) is called locally sharp. It implies that if ϵ_τ is large, then the error is large within that element. In the literature, an inequality of type (2.33) is called a reliability bound; see [43]. Note that a small sum of local estimators, ϵ_τ, implies a small global error.

Compared to the *a posteriori* bounds, (2.29) and (2.30), the local FOSLS functional is computationally inexpensive since it only involves a numerical integral within each element, while (2.29) and (2.30) require information from neighbor elements. This greatly simplifies implementation and communication in parallel. Another unique feature that the local FOSLS functional has is the local sharpness. It was shown in [32, 33] that either (2.29) or (2.30) can be a sharp lower bound for the local error unless the grid is already fine enough to resolve oscillatory features of the problem. In that case, an additional term, called the oscillation, is added. But, generalizing that idea to complicated systems of PDEs such as Navier-Stokes and MHD is not straightforward. The local FOSLS functional, on the other hand, has no such difficulty.

2.4 Refinement

In this section, we discuss the method for subdividing elements into subelements. The FOSLS methodology allows us to use simple bisection of elements. Here we describe the algorithm in the context of quadrilaterals in two dimensions and hexahedral elements in three dimensions. The quadrilateral elements are partitioned into four sub-elements of equal area, and the hexahedral elements are partitioned into eight sub-elements of equal volume. In the AFEM context, triangles (tetrahedron in three dimensions) are often employed as finite elements, which requires more attention in the subdivision stage to ensure conforming elements. The newest vertex routine in conjunction with simple bisection is often used there; see [33]. Although the simple methods we employ produce hanging nodes, the FOSLS method handles this situation easily. Hanging nodes are nodes along element edges or faces, in which the edge or face is shared by multiple elements, and the nodes are not defined for some elements; see Fig. 2.1. If conforming elements for FOSLS



(a) Possible grid after refinement without clean-up stage (b) grid after a clean-up stage is used to maintain 2-to-1 balance

Figure 2.1: A mesh resulting in slave nodes and master nodes; the red solid circle marks slave nodes, the blue solid circle marks master nodes. The arrows represent explicit correlations between slave and master nodes.

discretization are desired, each slave node is dependent on its master nodes at the endpoints of the edge or the face on which it is hanging through an explicit algebraic constraint. Explicit correlations between slave nodes and master nodes are established, see Fig. 2.1(b).

Although not required for FOSLS convergence, we perform an additional clean-up stage in our implementation so that two adjacent elements sharing an edge or a face should not differ in edge size by a factor greater than 2. Such a constraint is often referred to as the balance condition or 2-to-1 balance constraint [41, 42]. This clean-up stage helps to improve AMG convergence by reducing stencil size. Also, it helps track local features that are not stationary during a time-dependent simulation. Implementing the 2-to-1 balance constraint can be difficult in parallel. It depends on the data structure to represent the mesh, details are discussed in Chapter 4.

Remark. Although not pursued here, FOSLS is particularly amenable to non-conforming finite element spaces; see Berndt's lemma (Theorem 5.2) [9]

Chapter 3

Efficiency-based Adaptive Local Refinement

This chapter describes the ACE-like adaptive refinement algorithms in serial for NI-Newton-FOSLS-AMG approach. Numerical heuristics and the original ACE algorithm are introduced. Two variations of the ACE algorithm are also proposed. The first requires a fixed increase of DOF. The second forces a fixed anticipated error reduction. This is similar to the threshold-hold refinement algorithm except that an element is allowed to be refined more than once at a single level. These two variations can be generalized to circumstances in which other quantities are important in the simulation. For instance, one can force a fixed error reduction of the $||\text{div}\,\mathbf{u}||_0^2$ term to improve conservation in Stokes and Navier-Stokes problems. Finally, another algorithm, which minimizes the 'anticipated-overall-computational-work' efficiency factor (NACE) is developed. This method differs from the ACE algorithm and its variations in that the marking decision is made by the most accuracy-per-computational cost on all finer refinement levels instead of the next finer levels.

NI-Newton-FOSLS-AMG yields measures that allow us to estimate the anticipated error reduction and computational cost, which is used to make the refinement decisions based on optimizing computational efficiency. The performance of the efficiency-based adaptive refinement algorithms, applied to NI-Newton-FOSLS-AMG, is compared for a 2D Poisson problem with steep gradients and a time-dependent nonlinear 2D reduced model of the incompressible, resistive magnetohydrodynamics (MHD) equations. The numerical results show that all of the ACE-like algorithms used with NI-Newton-FOSLS-AMG are capable of approximating the solutions within the same error tolerance with much less computational cost than the threshold-based refinement and uni-

form refinement. Possibilities of extending the ACE-like algorithms to the context other than NI-Newton-FOSLS-AMG is discussed at the end of this chapter.

3.1 Efficiency-based Adaptive Local Refinement for NI-FOSLS-AMG

Recall that the goal of efficiency-based refinement strategies is to reach a certain error tolerance with the least amount of computational cost. Given a potential refinement strategy, one needs to estimate

- the reduction in the functional norm of the error and
- the computational cost of solving the resulting system of equations.

These estimates are established in the next two sections.

3.1.1 FOSLS Approximation Heuristics

Assume the solution space, $\mathcal{V}$, is a product of $H^1(\Omega)$ Sobolev spaces. For any tessellation, $\mathcal{T}_h$, with mesh size, h, let $\mathcal{V}^h$ be the finite-dimensional subspace consisting of continuous piecewise polynomials of degree p. Define $I_h\mathbf{u}$ to be the interpolation of the exact solution, $\mathbf{u}$, into the subspace $\mathcal{V}^h$. Then, there exists a constant, C, independent of $\mathbf{u}$, such that

$$||I_h\mathbf{u} - \mathbf{u}||_1 \leq Ch^s|\mathbf{u}|_{s+1} \tag{3.1}$$

for $0 < s \leq p$. Here, $||\cdot||_1$ is the $H^1(\Omega)$-norm and $|\cdot|_{s+1}$ is the $H^{s+1}(\Omega)$ semi-norm, (c.f. [10]). We further assume that the solution, $\mathbf{u}$, is smooth enough, i.e., $\mathbf{u} \in H^{p+1}(\Omega)$, so that (3.1) is valid for $s = p$. The following error bound is used to estimate the functional reduction:

$$\begin{aligned} \mathcal{G}(\mathbf{u}^h;\mathbf{f}) := \sum_{\tau\in\mathcal{T}_h} \mathcal{G}_\tau(\mathbf{u}^h;\mathbf{f}) &\leq \sum_{\tau\in\mathcal{T}_h} \mathcal{G}_\tau(I_h\mathbf{u};\mathbf{f}) \\ &\leq c_2||I_h\mathbf{u} - \mathbf{u}||_1^2 \\ &\leq c_2C^2h^{2p}|\mathbf{u}|_{p+1}^2. \end{aligned} \tag{3.2}$$

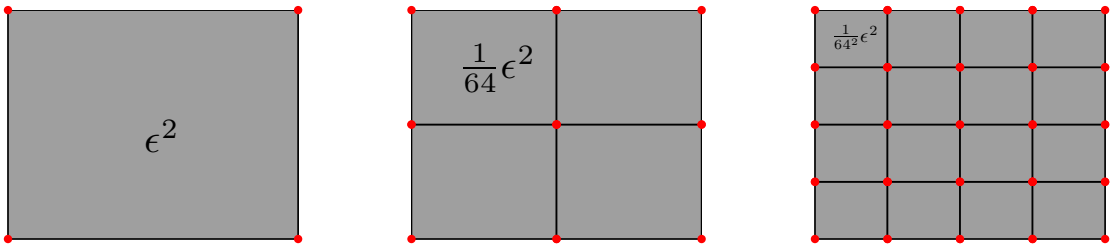
Similar bound holds for the local interpolate error:

$$\begin{aligned}\hat{\epsilon}_\tau^2 := \mathcal{G}_\tau(I_h\mathbf{u};\mathbf{f}) &\leq Dh_\tau^{2p}|\mathbf{u}|_{p+1,\tau}^2 \\ &\leq Dh_\tau^{2p}M_{p+1,\tau}H_\tau,\end{aligned} \tag{3.3}$$

where D is independent of $\mathbf{u}$ and h_τ, H_τ is the area of element τ, and $M_{p+1,\tau}H_\tau$ is a bound on $|\mathbf{u}|_{p+1,\tau}^2$. We assume that D and $M_{p+1,\tau}$ are relatively constant over element τ. Moreover, we assume $I_h\mathbf{u}$ is close enough to $\mathbf{u}^h$ so that bound (3.3) holds for local FOSLS functionals:

$$\epsilon_\tau^2 := \mathcal{G}_\tau(\mathbf{u}^h;\mathbf{f}) \approx \mathcal{G}_\tau(I_h\mathbf{u};\mathbf{f}) \leq Dh_\tau^{2p}M_{p+1,\tau}H_\tau. \tag{3.4}$$

Modifications of the assumptions might be necessary for certain situations, for example, when the solution contains a singularity (c.f. [22]) or the grid is not fine enough to resolve features of the solution. In such cases, these assumptions can be adaptively monitored so that run-time adjustments can be made.



(a) Parent element (b) Single refinement (c) Double refinement

Figure 3.1: Double refinement and local functional reduction for biquadratic element.

If element τ_i is split in two in each dimension, then we have 2^d new elements, $\tau_{i,1},...,\tau_{i,2^d}$, in $\mathbb{R}^d$. Using (3.4) as an asymptotic bound, we can estimate the local functional after refinement:

$$\sum_{j=1}^{2^d}\epsilon_{\tau_{i,j}}^2 \approx D(\frac{h_{\tau_i}}{2})^{2p}M_{p+1,\tau_i}\frac{H_{\tau_i}}{2^d}2^d \approx \frac{1}{2^{2p}}\epsilon_{\tau_i}^2. \tag{3.5}$$

Next, assume that the error is equally distributed among $\tau_{i,j}$, which yields

$$\epsilon_{\tau_{i,j}}^2 \approx \left(\frac{1}{2^{2p+d}}\right)\epsilon_{\tau_i}^2. \tag{3.6}$$

To give a little insight as to what this means, suppose quadratic elements are used in $\mathbb{R}^2$. Then the functional in each child element should be about $\frac{1}{64}$ of its parent. If an element is allowed to be refined twice, its grand-children are expected to have a local functional of about $\frac{1}{4096}$ of its grandparent; see Fig. 3.1. This suggests local errors can be reduced quickly and equally distributed if multiple refinements are correctly implemented.

3.1.2 Work estimate for NI-AMG

Here, we develop a procedure to estimate the computational work depending on the refinement decision made at each level. Let ℓ denote the refinement level, with $\ell = 0$ the coarsest and $\ell = L$ the finest grid. The following level-dependent definitions are made:

- N_ℓ =number of elements at level ℓ;
- $\mathcal{G}_\ell(\mathbf{u}^h; \mathbf{f})$ = FOSLS functional at level ℓ;
- $\epsilon_i^2 = \mathcal{G}_{\ell,\tau_i}(\mathbf{u}^h; \mathbf{f})$ = local functional on each element, τ_i, at level ℓ;
- $\mathcal{M}_\ell(\mathbf{u}^h; \mathbf{f}) = \sqrt{\mathcal{G}_\ell(\mathbf{u}^h; \mathbf{f})}$ = error at level ℓ.

The goal is to ensure that elements that contain large local error are refined first. This yields equally distributed error on the finer levels. The algorithm starts by ordering the elements at level ℓ so that

$$\epsilon_1^2 \geq \epsilon_2^2 \geq ... \geq \epsilon_{N_\ell}^2. \tag{3.7}$$

Let $r \in [0, 1]$ be the fraction of elements to be refined versus the total number of elements. Define $E_\ell(r)$, the associated fraction of the functional on the elements to be refined; that is,

$$E_\ell(r) = \frac{\sum_{i \leq rN_\ell} \epsilon_i^2}{\sum_{i=1}^{N_\ell} \epsilon_i^2}. \tag{3.8}$$

Although, as defined, the functional distribution function, $E(r)$, is piecewise constant on a mesh of size $\frac{1}{N_\ell}$. Consider a smooth interpolationof $E(r)$, then, $E(r)$ is monotonically increasing and concave down from $E_\ell(0) = 0$ to $E_\ell(1) = 1$, that is, $E'_\ell(r) \geq 0$ and $E''_\ell(r) \leq 0$. The derivative

$E'_\ell(0)$ can be used to indicate whether the functional is equally distributed. If it is large, then the functional is dominant in the first few elements. For example, in Fig. 3.2, when the error is dominated in the first two elements, the derivative at $r = 0$ is much greater than 1.

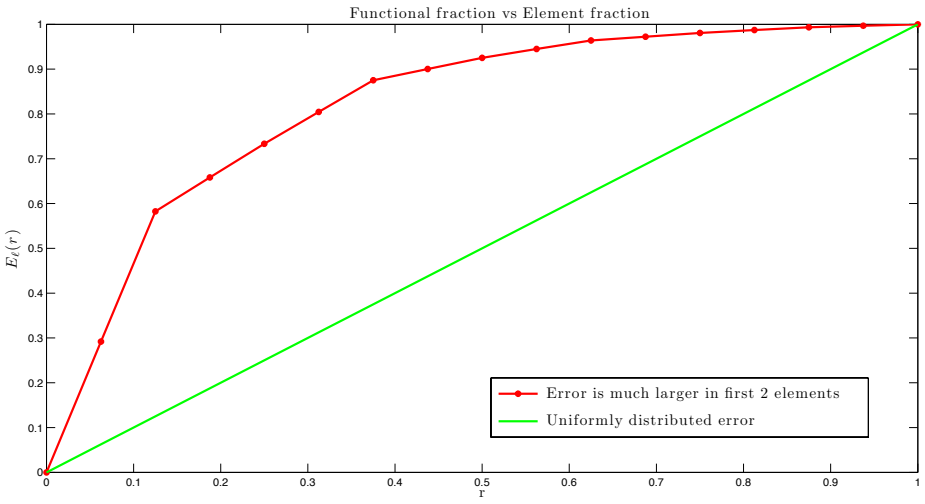


Figure 3.2: Fraction of functional versus the fraction of elements to be refined.

The algorithm allows an element to be refined more than once at each level. Let $r_i \in [0, 1]$ be the fraction of elements to be refined i times at level ℓ. Let m be the maximum refinements allowed on a single level. Writing $\mathbf{r} = (r_1, r_2, ..., r_m)$ and combining the functional distribution (3.8) and the functional reduction heuristic (3.5), we estimate the functional reduction as a function of $\mathbf{r}$:

$$\gamma_\ell(\mathbf{r}) = 1 - E_\ell(r_1) + \sum_{k=1}^{m-1} \frac{1}{2^{2kp}} \left[E_\ell(r_{k+1}) - E_\ell(r_k)\right] + \frac{1}{2^{2mp}} E_\ell(r_m). \tag{3.9}$$

For the work required to achieve this reduction, the main concern is the increase in the DOF. Assuming simple bisection of the elements, the anticipated increase in DOF is easily computed. We have

$$N_{\ell+1} \simeq \eta_\ell(\mathbf{r}) N_\ell, \tag{3.10}$$

where

$$\eta_\ell(\mathbf{r}) = 1 - r_1 + \sum_{k=1}^{m-1} 2^{kn}(r_{k+1} - r_k) + 2^{md} r_m. \tag{3.11}$$

Assume that the AMG convergence factor is bounded by $0 < \rho < 1$ uniformly in mesh-size. This value can be determined dynamically during computation. We further assume that, at each level, AMG V-cycles are applied until the discretization error, $\mathcal{M}_\ell(\mathbf{u}^h; \mathbf{f})$, is resolved. Then, the anticipated number of V-cycles, $\kappa_{\ell+1}(\mathbf{r})$, is determined by

$$\rho^{\kappa_{\ell+1}} = \frac{\mathcal{M}_{\ell+1}}{\mathcal{M}_\ell} = \sqrt{\gamma_\ell(\mathbf{r}))}. \tag{3.12}$$

Solving for $\kappa_{\ell+1}$ gives

$$\kappa_{\ell+1}(\mathbf{r}) = \left\lceil \frac{1}{2} \frac{\log \gamma_\ell(\mathbf{r})}{\log \rho} \right\rceil. \tag{3.13}$$

In a real simulation, a certain number of V-cycles are needed to extrapolate the discretization error. Denoting this number by $ncyc_{\min}$, we have

$$\kappa_{\ell+1}(\mathbf{r}) = \max\left(\left\lceil \frac{1}{2} \frac{\log \gamma_\ell(\mathbf{r})}{\log \rho} \right\rceil, ncyc_{\min} \right). \tag{3.14}$$

Now, the work on the next level, $\ell + 1$, is given by

$$\mathcal{W}_{\ell+1}(\mathbf{r}) = [c_s + c_v \kappa_{\ell+1}(\mathbf{r})] \times N_{\ell+1} = [c_s + c_v \kappa_{\ell+1}(\mathbf{r})] \times \eta_\ell(\mathbf{r}) \times N_\ell, \tag{3.15}$$

where c_s and c_v represent the respective set-up cost and cost factor for a V-cycle.

Remark. The nested iteration strategies proposed in Chapter 2. stop the linear iteration once the algebraic error is within a certain range, see the quantity described in (2.24), of the discretization error. That leads to a variation of equation (3.12)

$$\rho^{\kappa_{\ell+1}} \geq \frac{\sqrt{\epsilon_s^{\ell+1}} \mathcal{M}_{\ell+1}}{\sqrt{\epsilon_0^{\ell+1}} \mathcal{M}_{\ell+1}} = \frac{\sqrt{\epsilon_s^{\ell+1}}}{\sqrt{\epsilon_0^{\ell+1}}}, \tag{3.16}$$

where $\epsilon_s^{\ell+1}$ is computed using (2.24), i.e.,

$$\epsilon_s^{\ell+1} = \frac{\left(\left(\rho^{1/\eta_\ell(\mathbf{r})} \right)^2 - 1 \right) (1 - \gamma_\ell(\mathbf{r}))}{\rho^2 - \left(\rho^{1/\eta_\ell(\mathbf{r})} \right)^2}, \tag{3.17}$$

and $\epsilon_0^{\ell+1}$ is computed using (2.23)

$$\epsilon_0^{\ell+1} = \frac{\epsilon_s^\ell + (1 - \gamma_\ell(\mathbf{r}))}{\gamma_\ell(\mathbf{r})}. \tag{3.18}$$

3.1.3 Efficiency-based Marking Strategies

Let $\mathcal{E}_T = \frac{\mathcal{G}_L}{\mathcal{G}_0}$ be the desired total factor of reduction from the initial functional. We wish to find an overall refinement strategy that minimizes the total work required to achieve a reduction of the functional by the factor $\mathcal{E}_T$. That is, we want to find a sequence, $\{\mathbf{r}_\ell\}_{...}$, to minimize the total work:

$$\mathcal{W}_T = \sum_{\ell=1}^{L} \mathcal{W}_\ell \qquad \text{with } \prod_{\ell=0}^{L-1} \gamma_\ell = \mathcal{E}_T. \tag{3.19}$$

Here, $\mathcal{W}_\ell$ is the work at level ℓ and γ_ℓ is the functional reduction between level ℓ and $\ell+1$. Several difficulties arise in the solution of such a minimization problem. The choice of $\mathbf{r}_\ell$ depends on the functional distribution, E_ℓ, which is unavailable before refinement is performed at level ℓ. We can estimate the distributions at finer levels based on heuristics (3.5) and the functional distribution at coarser levels. Such estimates are often not accurate enough, especially when the grid is not fine enough to resolve the solution.

Our first approach to (3.19) is based on optimization of the accuracy-per-computational-cost in the move from grid ℓ to grid $\ell+1$. Define the effective functional reduction measure as follows:

$$\gamma_\ell(\mathbf{r})^{\frac{1}{\mathcal{W}_{\ell+1}(\mathbf{r})}}. \tag{3.20}$$

The ACE algorithm, first developed in [22,34], marks elements for refinement based on minimizing the anticipated effective functional reduction.

ALGORITHM 4 (ACE)**.** *At level ℓ, order the elements so that*

$$\epsilon_1^2 \geq \epsilon_2^2 \geq ... \geq \epsilon_{N_\ell}^2.$$

Allow m-multiple refinements, e.g., $m = 1, 2, 3$. Let $\mathbf{r} = (r_1, r_2, ..., r_m)$ with $0 \leq r_m \leq ... \leq r_1 \leq 1$. Find

$$\mathbf{r}_{opt} = \arg\min_{\mathbf{r}} \gamma_\ell(\mathbf{r})^{\frac{1}{\mathcal{W}_{\ell+1}(\mathbf{r})}}, \tag{3.21}$$

or

$$\mathbf{r}_{opt} = \arg\min_{\mathbf{r}} \frac{\log \gamma_\ell(\mathbf{r})}{\mathcal{W}_{\ell+1}(\mathbf{r})}. \tag{3.22}$$

Then, refine the first $\lceil r_i N_\ell \rceil$ *elements* i *times,* $i = 1, 2, ..., m$.

In some instances, however, the ACE algorithm only refines a few elements. This may be optimal for the move from grid ℓ to grid level $\ell + 1$, but, if this happens at all levels, then the total work (3.19) will be unnecessarily large. Although c_s and ρ are factored into the algorithm, the above behavior may occur if ρ is very close to 1.0 and c_s is not large relative to the cost of one V-cycle. Furthermore, if elements are allowed to be refined more than twice, finding $\mathbf{r}_{opt}$ can be expensive at finer levels. One modification that reduces this expense is the use of bins, which we discuss in Chapter 4.

Below, we propose three variations of the ACE algorithm. The first two, ACE-DOF and ACE-Reduc, enforce fixed increase in the DOF and a fixed reduction of the functional, respectively. The third algorithm, NACE, attempts to optimize (3.19). Numerical results in Chapter 3.2 indicate that all the ACE algorithms used with NI-FOSLS-AMG are able to approximate the solutions to the same accuracy with much less computational cost than the threshold-based refinement and uniform refinement.

As indicated above, the algorithm ACE-DOF has the goal of forcing ACE to refine a certain number of elements such that the number of elements at the next level is a prescribed factor of the number that would result from performing a single refinement globally.

ALGORITHM 5 (ACE-DOF)**.** *On level* ℓ*, order the elements so that the local functional is decreasing. Assume* $m \geq 2$*. Given a parameter* $1 < \theta_{DOF} \leq (2^d)^m$*. Find*

$$\mathbf{r}_{opt} = \arg\min_{\mathbf{r}} \frac{\log \gamma_\ell(\mathbf{r})}{\mathcal{W}_{\ell+1}(\mathbf{r})} \qquad \textit{with} \qquad \eta_\ell(\mathbf{r}) \simeq \theta_{DOF}. \tag{3.23}$$

In particular, one can choose $\theta_{DOF} = 2^d$ such that the number of elements at next level is the same as the number that would result from a single global refinement.

The second variation finds the optimal fraction, $\mathbf{r}_{opt}$, by fixing the anticipated functional reduction such that it is a prescribed factor of the anticipated functional reduction that would result

from a single global refinement.

ALGORITHM 6 (ACE-Reduc)**.** *At level ℓ, order the elements so the local functional is decreasing. Assume $m \geq 2$. Given a parameter $(\frac{1}{2^{2p}})^m \leq \theta_{Reduc} < 1$. Find*

$$\mathbf{r}_{opt} = \arg\min_{\mathbf{r}} \frac{\log \gamma_\ell(\mathbf{r})}{\mathcal{W}_{\ell+1}(\mathbf{r})} \qquad \textit{with } \gamma_\ell(\mathbf{r}) \simeq \theta_{Reduc}, \tag{3.24}$$

or

$$\mathbf{r}_{opt} = \arg\min_{\mathbf{r}} \mathcal{W}_{\ell+1}(\mathbf{r}) \qquad \textit{with } \gamma_\ell(\mathbf{r}) \simeq \theta_{Reduc}. \tag{3.25}$$

All of the above algorithms are developed based on optimization between two consecutive levels, which we refer to as local optimization in this context. Of course, this does not guarantee global optimization; that is, optimization of the entire solution process. We also devise a marking algorithm that minimizes the 'anticipated-**overall**-computational-cost' efficiency, as defined in (3.19), (which we call NACE). Let

$$\epsilon_{T,\ell} = \frac{\mathcal{G}_L}{\mathcal{G}_\ell} \tag{3.26}$$

be the overall functional reduction needed from the current functional to the desired tolerance. Let

$$K_\ell(\mathbf{r}) = \lceil \frac{\log(\epsilon_{T,\ell})}{\log(\gamma_\ell(\mathbf{r}))} \rceil. \tag{3.27}$$

In order to obtain $\mathcal{G}_L$, assume we repeat $\gamma_\ell(\mathbf{r})$ reduction $K_\ell(\mathbf{r})$ times. The anticipated total work to accomplish this is

$$\begin{aligned} \mathcal{W}_{T,\ell}(\mathbf{r}) &= [c_s + c_v \kappa_{\ell+1}(\mathbf{r})] \left(\eta_\ell + \eta_\ell^2 + ... \eta_\ell^{K_\ell(\mathbf{r})-1} \right) N_\ell \\ &= [c_s + c_v \kappa_{\ell+1}(\mathbf{r})] \frac{\left(\eta_\ell(\mathbf{r})^{K_\ell(\mathbf{r})} - 1\right)}{\eta_\ell(\mathbf{r}) - 1} N_\ell, \end{aligned} \tag{3.28}$$

where $\kappa_{\ell+1}(\mathbf{r})$, the anticipated number of V-cycles associated with reduction $\gamma_\ell(\mathbf{r})$, is defined in (3.14). Now, the NACE algorithm is described.

ALGORITHM 7 (NACE)**.** *At level ℓ, order the elements so that the local functional is decreasing. The refinement decision is made by finding $\boldsymbol{r}_{opt}$ to minimize the estimated remaining total work given by (3.28). This is equivalent to finding*

$$\mathbf{r}_{opt} = \arg\min_{\mathbf{r}} \log\left((c_s + c_v \kappa_{\ell+1}(\mathbf{r})) \, \frac{\eta_\ell(\boldsymbol{r})^{K_\ell(\boldsymbol{r})} - 1}{\eta_\ell(\boldsymbol{r}) - 1} \right), \tag{3.29}$$

where $\kappa_{\ell+1}$, η_ℓ, and K_ℓ are given by (3.14), (3.11), and (3.27), respectively.

While this algorithm cannot guarantee optimal work as defined in (3.19), it attempts to take into consideration the total work required on all remaining levels.

3.2 Numerical Results

This section presents numerical results for ACE-like algorithms in the context of a serial computer architecture. The first test is a Poisson equation with steep gradients and flats. The second test involves a system of time dependent MHD equations.

3.2.1 Poisson Equation

Consider the Poisson problem on the unit square, $\Omega = (0,1) \times (0,1)$,

$$\begin{cases} -\Delta p = f(x,y) & \text{in } \Omega, \\ p = g & \text{on } \partial\Omega. \end{cases} \tag{3.30}$$

The equivalent first-order system we study here is

$$\begin{cases} -\nabla \cdot U = f \quad \text{in} \quad \Omega, \\ U = \nabla p \\ \nabla \times U = 0 \\ p = g \quad \text{on} \quad \partial\Omega, \\ \tau \cdot U = \dfrac{\partial g}{\partial \tau}, \end{cases} \tag{3.31}$$

where U is a vector of auxiliary unknowns and τ is the unit vector tangent to $\partial\Omega$. H^1-ellipticity of the corresponding FOSLS functional is shown in [17].

3.2.1.1 Test Problem: Steep Gradients and Flats

Define the function

$$p_1(r,\theta) = \begin{cases} 1 & r \le 0.7, \\ h_1(r) & 0.7 \le r \le 0.8, \\ 0 & r \ge 0.8, \end{cases} \tag{3.32}$$

where (r, θ) is the polar coordinate centered at the origin and h_1 is a unique degree 7 polynomial such that $p_1 \in H^4(\Omega)$. Similarly, define the function

$$p_2(r', \theta') = \begin{cases} 2 & r' \leq 0.7, \\ h_2(r') & 0.7 \leq r' \leq 0.8, \\ 0 & r' \geq 0.8, \end{cases} \tag{3.33}$$

where (r', θ') is the polar coordinate centered at $(1, 0)$ and h_2 is a unique degree 7 polynomial such that $p_2 \in H^4(\Omega)$.

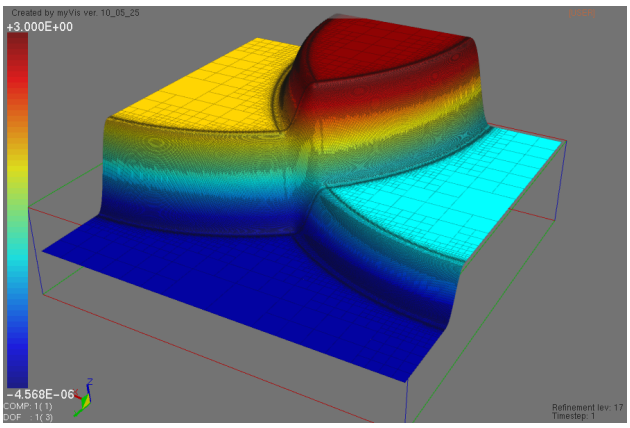


Figure 3.3: Exact solution

The right-hand side, f, and boundary data, g, are chosen such that the exact solution is given by

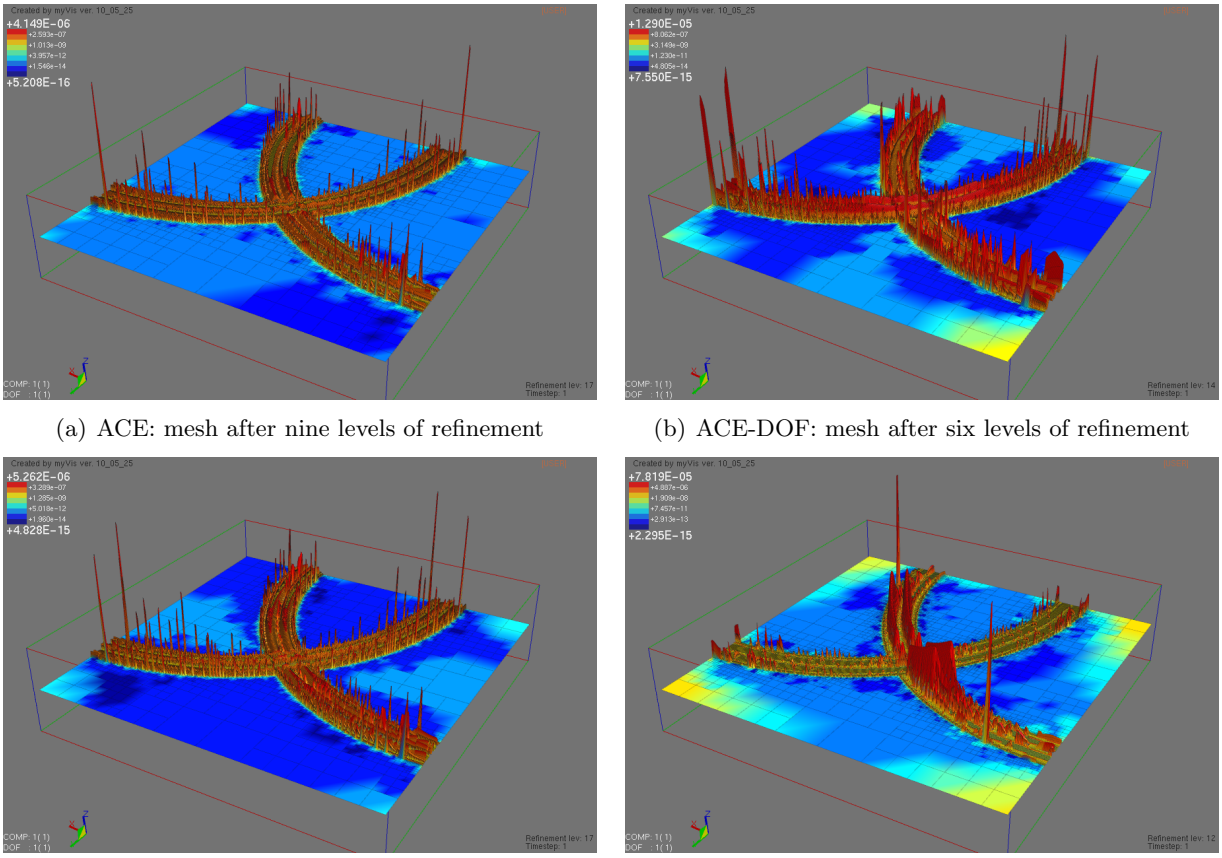
$$p(x, y) = p_1(x, y) + p_2(x, y). \tag{3.34}$$

The 3D plot of p displayed in Fig. 3.3 shows a large gradient within two thin strips with constants elsewhere. For a given mesh size and approximation order, the error should be relatively large in the thin strips. To get an accurate approximation, the refinement algorithm needs to concentrate elements there to effectively resolve these gradients.

3.2.1.2 Results

All ACE algorithms are applied to test problem (3.30) with bi-quadratic elements. Refinement stops when the functional is reduced by a factor of 10^{-7}. Elements are allowed to be refined at

most twice at each level. The finest grids and functional distribution are depicted in Fig. 3.4. They are consistent with the anticipated mesh because the finest resolution encompasses the strips containing the large gradient. Furthermore, we see that all schemes result in equal distribution of error on the finest grids. In Fig. 3.4, we assign colors to each element according to the size of local functional in such a way that the first color represents local functionals in the range $[\epsilon^2_{\max}, \frac{1}{8}\epsilon^2_{\max}]$, the second color represents the range $[\frac{1}{8}\epsilon^2_{\max}, \frac{1}{8^2}\epsilon^2_{\max}]$ and so forth. If we only consider the functional distribution within the steep gradients, since the solution is flat elsewhere, then it is observed that all ACE schemes result in only three colors within the two thin strips. In other words, local functionals only differ by a factor of $\frac{1}{512}$. Since the error is the square root of the functional, local errors are only different by approximately a factor of $\frac{1}{22}$.



(a) ACE: mesh after nine levels of refinement

(b) ACE-DOF: mesh after six levels of refinement

(c) ACE-Reduc: mesh after eight levels of refinement

(d) NACE: mesh after five levels of refinement

Figure 3.4: Locally-refined meshes and functional distribution

To investigate the behavior of each scheme, we tabulate various relevant values with respect to each refinement level. These results are given in Tables 3.1, 3.2, 3.3, and 3.4. All schemes work as expected. A large fraction of elements are refined at the initial levels when grids are too coarse to resolve features of the solution. For instance, Table 3.1 shows that more than 50% of the elements containing nearly 96% of the error are refined at the first 5 refinement levels. Then, at the intermediate levels, once local features of the solution are exposed, a small fraction of elements that contain large local error are refined, e.g., in Table 3.1, only 34% of the elements are refined at level 6 and 7. In particular, at refinement level 6, 0.63% of the elements containing nearly 28% of the error are refined twice, which speeds up the process of equal-distribution of the error. Later, at finer levels, since error is fairly equally distributed, a large fraction of elements are refined once again. For example, in Table 3.1, 82% of the elements are refined at refinement level 9. To show that ACE eventually results in nearly global refinement, more refinement levels are required; however, this exceeds the memory limit of our machine. We give such an example in the parallel section. Furthermore, the last column in each table shows that the anticipated functional reduction, γ_{est}, provides an accurate estimate to the actual reduction at finer levels. This verifies the FOSLS approximation heuristics derived in Chapter 3.1.1.

Next, to demonstrate the efficiency of each scheme, we compute the total computational cost in terms of a work unit (WU) on the finest grid, defined as the amount of computation required to perform one matrix vector multiplication on the finest grid. The total computational cost is, then, given in terms of the total work units (TWU):

$$TWU = \frac{\sum_{\ell=1}^{L}(C_s + ncyc_\ell) \times (\nu_1 + \nu_2) \times \sigma_\ell \times nnz_\ell}{nnz_L}. \tag{3.35}$$

Here, ν_1 and ν_2 are the number of pre- and post-relaxations, respectively, σ_ℓ is the operator complexity of the AMG solver at level ℓ, nnz_ℓ is the number of nonzeroes at level ℓ, $ncyc_\ell$ is the number of AMG cycles performed at level ℓ, and C_s is defined as the set up cost in terms of the cost of a single V-cycle on level ℓ. For this test problem, $V(1,1)$-cycles are employed and the set up cost is proportional to 30.0 V-cycles. Results show that the total work to solve the linear systems

ℓ	$\mathcal{G}_\ell$	N_ℓ	nnz_ℓ	r_1	r_2	$E(r_1)$	$E(r_2)$	η_ℓ	$ncyc$	$\gamma_{est}(\gamma_{act})$
1	1.37e+5	16	9,801	63%	0.00%	96%	00%	2.87	4	0.10(0.49)
2	6.70e+4	46	29,709	52%	0.00%	98%	00%	2.56	4	0.08(0.42)
3	2.85e+4	130	84,277	52%	0.00%	99%	00%	2.57	4	0.07(0.19)
4	5.49e+3	364	234,237	56%	0.55%	98%	16%	2.74	4	0.07(0.11)
5	5.92e+2	1,114	664,061	53%	2.33%	98%	46%	2.88	4	0.05(0.09)
6	5.48e+1	3,505	2,093,261	34%	0.63%	92%	28%	2.09	4	0.12(0.13)
7	7.33e+0	7,756	4,595,173	34%	0.04%	91%	02%	2.02	4	0.15(0.15)
8	1.14e+0	16,213	9,531,203	69%	0.12%	98%	06%	3.09	4	0.08(0.08)
9	1.01e-1	51,157	29,884,277	82%	0.27%	99%	09%	3.50	4	0.07(0.07)
10	7.62e-3	181,633	105,595,645							

Table 3.1: ACE, $\sigma_L = 1.899$, set up 171.804 WU, solve 22.907 WU.

ℓ	$\mathcal{G}_\ell$	N_ℓ	nnz_ℓ	r_1	r_2	$E(r_1)$	$E(r_2)$	η_ℓ	$ncyc$	$\gamma_{est}(\gamma_{act})$
1	1.37e+5	16	9,801	81%	6.25%	98%	29%	4.19	4	0.06(0.42)
2	5.77e+4	67	41,873	60%	10.45%	99%	59%	4.04	4	0.03(0.25)
3	1.44e+4	298	184,313	59%	10.40%	99%	82%	4.01	4	0.02(0.05)
4	6.99e+2	1,333	809,373	64%	9.00%	99%	76%	4.00	4	0.02(0.02)
5	1.60e+1	5,920	3,573,901	70%	7.62%	99%	49%	4.00	4	0.04(0.05)
6	7.84e-1	25,153	15,128,019	80%	5.01%	99%	47%	4.00	4	0.04(0.04)
7	3.12e-2	103,444	61,036,269							

Table 3.2: ACE-DOF, $\sigma_L = 1.983$, set up 160.695 WU, solve 21.426 WU.

ℓ	$\mathcal{G}_\ell$	N_ℓ	nnz_ℓ	r_1	r_2	$E(r_1)$	$E(r_2)$	η_ℓ	$ncyc$	$\gamma_{est}(\gamma_{act})$
1	1.37e+5	16	9,801	100%	0.00%	100%	00%	4.00	4	0.065(0.48)
2	6.60e+4	64	38,025	47%	1.56%	99%	13%	2.59	4	0.065(0.39)
3	2.59e+4	169	102,677	46%	1.18%	99%	14%	2.52	4	0.065(0.13)
4	3.24e+3	472	284,901	57%	1.27%	99%	11%	2.86	4	0.065(0.07)
5	2.39e+2	1,492	888,377	58%	1.14%	99%	17%	2.86	4	0.065(0.09)
6	2.16e+1	4,627	2,741,061	60%	0.63%	98%	22%	2.87	4	0.065(0.09)
7	1.95e+0	13,849	8,169,677	70%	0.86%	98%	27%	3.20	4	0.065(0.08)
8	1.52e-1	45,448	26,632,607	81%	0.51%	99%	17%	3.49	4	0.065(0.07)
9	1.06e-2	160,897	93,704,707							

Table 3.3: ACE-Reduc, $\sigma_L = 1.994$, set up 173.298 WU, solve 23.106 WU.

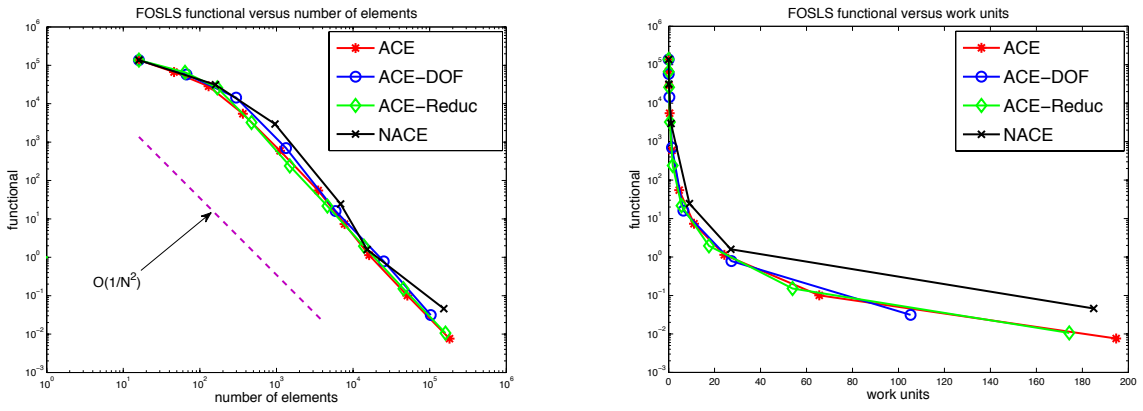
ℓ	$\mathcal{G}_\ell$	N_ℓ	nnz_ℓ	r_1	r_2	$E(r_1)$	$E(r_2)$	η_ℓ	$ncyc$	$\gamma_{est}(\gamma_{act})$
1	1.37e+5	16	9,801	100%	50%	100%	93%	10.00	4	0.010(0.228)
2	3.11e+4	160	104,967	46%	23%	99%	86%	5.16	4	0.014(0.095)
3	2.96e+3	958	653,599	72%	29%	99%	97%	6.63	4	0.006(0.008)
4	2.45e+1	6,868	5,511,917	36%	01%	97%	50%	2.18	4	0.061(0.065)
5	1.59e+0	15,277	12,768,435	56%	55%	95%	95%	9.18	4	0.057(0.029)
6	4.58e-2	153,064	108,988,185							

Table 3.4: NACE, $\sigma_L = 2.220$, set up = 158.112 WU, solve 21.082 WU.

throughout all levels is about 22 work units, and the total set up cost is between 158 and 173 work units. To illustrate what these numbers mean, we take the finest grid resulting from the original ACE, set up the FOSLS discrete problem, and solve it using AMG with a zero initial guess. The results in Table. 3.5 show that the NI-FOSLS-AMG-ACE method requires only about 137% of the work of solving the problem directly on the same finest grid. That is, the process of discovering the optimal grid requires an overhead of 37% of the cost of solving on that grid.

Method	Setup Cost	n_{cyc}	Solve Cost	Total Work
NI-FOSLS-AMG-ACE	171.80	4	22.91	194.71
FOSLS-AMG	113.94	11	37.98	141.92

Table 3.5: Comparison of NI-FOSLS-AMG-ACE and applying FOSLS-AMG directly to the finest-grid, n_{cyc} is the number of V-cycles used on the finest grid.



(a) Functional vs number of elements

(b) Functional vs work units

Figure 3.5: Comparison of all ACE schemes, where a work unit is defined as the cost of one matrix vector multiplication on the finest grid of ACE.

Next, we see from Table. 3.4 that the NACE scheme takes the least levels of refinement to reach the error tolerance due to a lot more double refinements. This may lead to possible over-refinement and less accurate grids, which is indeed the case; see Fig. 3.5(a), where the functional-versus-number-of-elements curve and work units are depicted. The convergence rates of ACE, ACE-DOF, and ACE-Reduc approach the optimal rate of quadratic elements, while the convergence rate of the NACE scheme is slightly slower. Double refinements also have the potential of introducing

more nonzeroes in the resulting matrices, e.g., the NACE scheme results in more nonzeroes in the finest-grid matrix than ACE and ACE-Reduc, but fewer elements. To compare the computational work required to reach a certain functional value, we compute the work units in terms of relaxation on the finest grid of ACE, which contains 105,595,645 nonzeroes. The ACE scheme (and its two variations) results in smaller functional values compared with the NACE scheme. It appears that, for this test, when the work units equal 180, the functional resulting from NACE is almost an order of magnitude larger than the functional using the ACE scheme, as seen in Fig. 3.5(b).

Lastly, we compare four schemes by computing the total work ratio

$$WR = \frac{\sum_{\ell=1}^{L_i}(C_0 + ncyc_{\ell,i}) \times (\nu_1 + \nu_2) \times \sigma_{\ell,i} \times nnz_{\ell,i}}{\sum_{\ell=1}^{L_{nace}}(C_0 + ncyc_{\ell,nace}) \times (\nu_1 + \nu_2) \times \sigma_{\ell,nace} \times nnz_{\ell,nace}}$$

and total functional reduction ratio

$$FRR = \frac{\mathcal{G}_{L_i}}{\mathcal{G}_{L_{nace}}}$$

relative to NACE, and the effective functional reduction ratio

$$WR^{1/FFR}$$

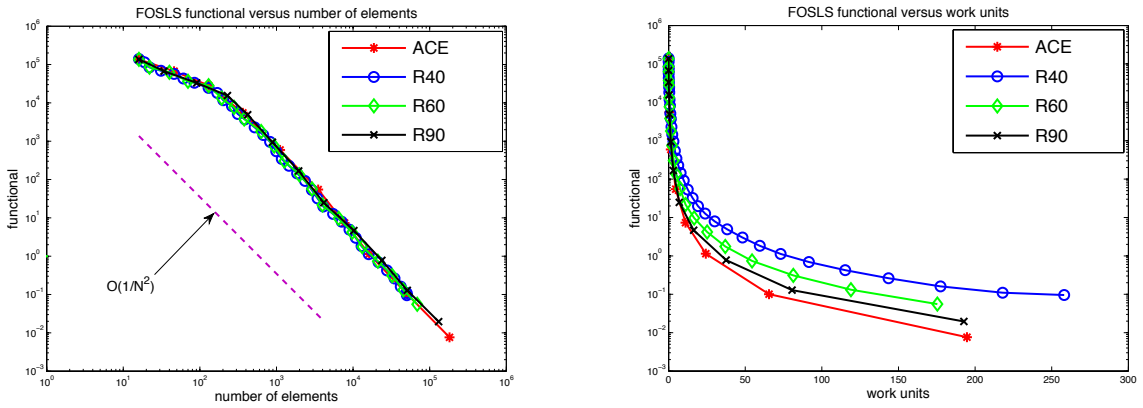
in Table. 3.6. It appears that the original ACE and ACE-Reduc have the best efficiency. However, this is the ideal case. In real simulation, many levels of refinement often result in a lot of overhead, e.g., in parallel, the cost of communication and load-balancing must be considered, methods such NACE employing more aggressive marking might be a better choice.

Ratio to NACE	ACE	ACE-DOF	ACE-Reduc	NACE
Work Ratio	1.0528	0.5692	0.9424	1.000
Functional Reduction Ratio	0.1663	0.6819	0.2318	1.000
Effective Functional Reduction Ratio	0.1820	0.5104	0.2120	1.000

Table 3.6: Comparison of effective functional reduction of all ACE schemes.

Comparison to Threshold-based Marking Scheme

We conclude our analysis of the Poisson equation by comparing the original ACE algorithm with the threshold-based marking scheme (1.4). Three threshold-based algorithms that refine 40,



(a) Functional vs number of elements

(b) Functional vs work units

Figure 3.6: Comparison of threshold-based schemes with refinement of 40, 60, and 90 percent of the functional at each level, and the ACE scheme.

60, and 90 percent of the functional at each refinement level are considered. Work units for Fig. 3.6(b) are defined as one relaxation on the finest grid of ACE. It is shown in Fig. 3.6(a) that the convergence rate of ACE is the same as the best convergence rate of the three threshold-based algorithm. For the same amount of work, the ACE scheme results in the smallest functional value compared with the threshold-based schemes. For example, when work units = 200, ACE results in a functional one order of magnitude less than threshold-based refinement schemes. This is expected since the ACE algorithm is based on optimizing computational efficiency.

3.2.2 Magnetohydrodynamics

In this section, an incompressible, resistive magnetohydrodynamics (MHD) test problem is investigated. The results in [4,5] show that methods such as nested iteration and first-order system least-squares are capable of solving the nonlinear MHD systems in a minimal amount of work units. Here, the various forms of adaptive mesh refinement described above are applied to a tokamak test problem [18, 19, 36, 40]. A reduced set of MHD equations is obtained that models a "large-aspect-ratio" tokamak, with non-circular cross-sections. The magnetic B-field along the z-direction, or the toroidal direction, is very large and mostly constant. In this context, we are able to look at plasma behavior in the poloidal cross-section. The 2D reduced model is described by the following

equations:

$$\frac{1}{\sqrt{R_e}}\nabla \times \mathbf{u} - \sqrt{R_e}\omega = 0, \tag{3.36}$$

$$\frac{1}{\sqrt{R_e}}\nabla \cdot \mathbf{u} = 0, \tag{3.37}$$

$$\frac{1}{\sqrt{R_e}}\frac{\partial \mathbf{u}}{\partial t} - \mathbf{u} \times \boldsymbol{\omega} - \mathbf{j} \times \mathbf{B} - \sqrt{R_e}\nabla p + \frac{1}{\sqrt{R_e}}\nabla^{\perp}\omega = \mathbf{f}, \tag{3.38}$$

$$\frac{1}{\sqrt{S_L}}\nabla \times \mathbf{B} - \sqrt{S_L}j = 0, \tag{3.39}$$

$$\frac{1}{\sqrt{S_L}}\nabla \cdot \mathbf{B} = 0, \tag{3.40}$$

$$\frac{1}{\sqrt{S_L}}\frac{\partial \mathbf{B}}{\partial t} + \frac{1}{\sqrt{R_e S_L}}(\mathbf{u} \cdot \nabla \mathbf{B} - \mathbf{B} \cdot \nabla \mathbf{u}) + \frac{1}{\sqrt{S_L}}\nabla^{\perp} j = \mathbf{g}. \tag{3.41}$$

The x-direction denotes the periodic poloidal direction in the tokamak, whereas the y dimension represents a thin annulus in the poloidal cross section. In this 2D setting, vorticity, ω, and current density, j, are both scalar variables. The remaining variables are the fluid velocity, $\mathbf{u}$, the fluid pressure, p, and the magnetic field, $\mathbf{B}$. The equations have been scaled using the Reynolds number, R_e, which is the ratio of fluid speed to viscosity, and the Lundquist number, S_L, which is the ratio of fluid speed to magnetic resistivity. This scaling produces a first-order system that is amenable to algebraic multigrid methods in the FOSLS context, as shown in [4].

One important application of MHD physics is the study of instabilities that can occur in tokamak fusion reactors. One such instability, the island coalescence problem, is described below. The various ACE schemes are applied to see which one most efficiently captures the magnetic reconnection that results from this instability.

3.2.2.1 Test Problem: Island Coalescence

This test problem simulates an island coalescence in the current density arising from perturbations in an initial current density sheet. A current density sheet in the toroidal direction of the tokamak is perturbed, resulting in an instability that causes a reconnection in the magnetic field lines and the merging of two islands in the current density field. This produces a sharp peak in

current density where the magnetic field lines reconnect. This region is known as the reconnection zone, and the point at which the magnetic field lines break is known as the $\mathcal{X}$ point. See [7,28,36] for more detail. We choose a low enough resistivity (i.e., Lundquist number above $50,000$) in order to observe the interesting physics. For the following simulations, we define

$$\begin{aligned} \Omega &= [-1,1]\times[-1,1], \\ R_e &= S_L = 50,001. \end{aligned}$$

The initial conditions at equilibrium are

$$\mathbf{B}_0(x,y) = \frac{1}{\cosh(2\pi y)+k\cos(2\pi x)}\begin{pmatrix}\sinh(2\pi y)\\ k\sin(2\pi x)\end{pmatrix}, \tag{3.42}$$

$$\mathbf{u}_0(x,y) = \mathbf{0}, \tag{3.43}$$

$$\omega_0(x,y) = 0, \tag{3.44}$$

$$j_0(x,y) = \nabla\times\mathbf{B}_0 = \frac{2\pi(k^2-1)}{(\cosh(2\pi y)+0.2\cos(2\pi x))^2}, \tag{3.45}$$

$$p_0(x,y) = \frac{(1-k^2)}{2}\left(1+\frac{1}{(\cosh(2\pi y)+0.2\cos(2\pi x))^2}\right), \tag{3.46}$$

where $k=0.2$. These initial conditions are perturbed away from equilibrium as follows:

$$\delta\mathbf{B}_0(x,y) = \begin{pmatrix}-\epsilon\frac{1}{\pi}\cos(\pi x)\sin(\pi\frac{y}{2})\\ \frac{1}{2}\epsilon\frac{1}{\pi}\cos(\pi\frac{y}{2})\sin(\pi x)\end{pmatrix}, \tag{3.47}$$

$$\delta j_0(x,y) = \epsilon\cos(\pi\frac{y}{2})\cos(\pi x), \tag{3.48}$$

where $\epsilon=-0.01$. The boundary conditions are periodic in x and Dirichlet for the current density and vorticity on the top and bottom of the domain. We also have $\mathbf{n}\cdot\mathbf{u}$ and $\mathbf{n}\cdot\mathbf{B}$ known on the top and bottom. Again, the FOSLS formulation, (3.36)-(3.41), is H^1 elliptic.

3.2.2.2 Results

The problem was run to time $15\tau_A$ with a timestep of $0.1\tau_A$, using a BDF-2 implicit time-stepping scheme. Here, τ_A is the time in Alfvén units. It is the time needed for an Alfvén wave to travel across the domain $[7, 40]$. By this time, the islands have coalesced and the large peak in current density has occurred at the reconnection point. Using both uniform refinement and the ACE schemes, the instability was captured. With ACE employed, the grids evolve over time to refine in areas with steeper gradients. In this problem, as time progresses, a steep gradient occurs at the reconnection point. This is seen in Fig. 3.7. We expect, then, that most of the refinement occurs in this region, which is indeed the case. Next, a comparison of the 4 ACE schemes is done

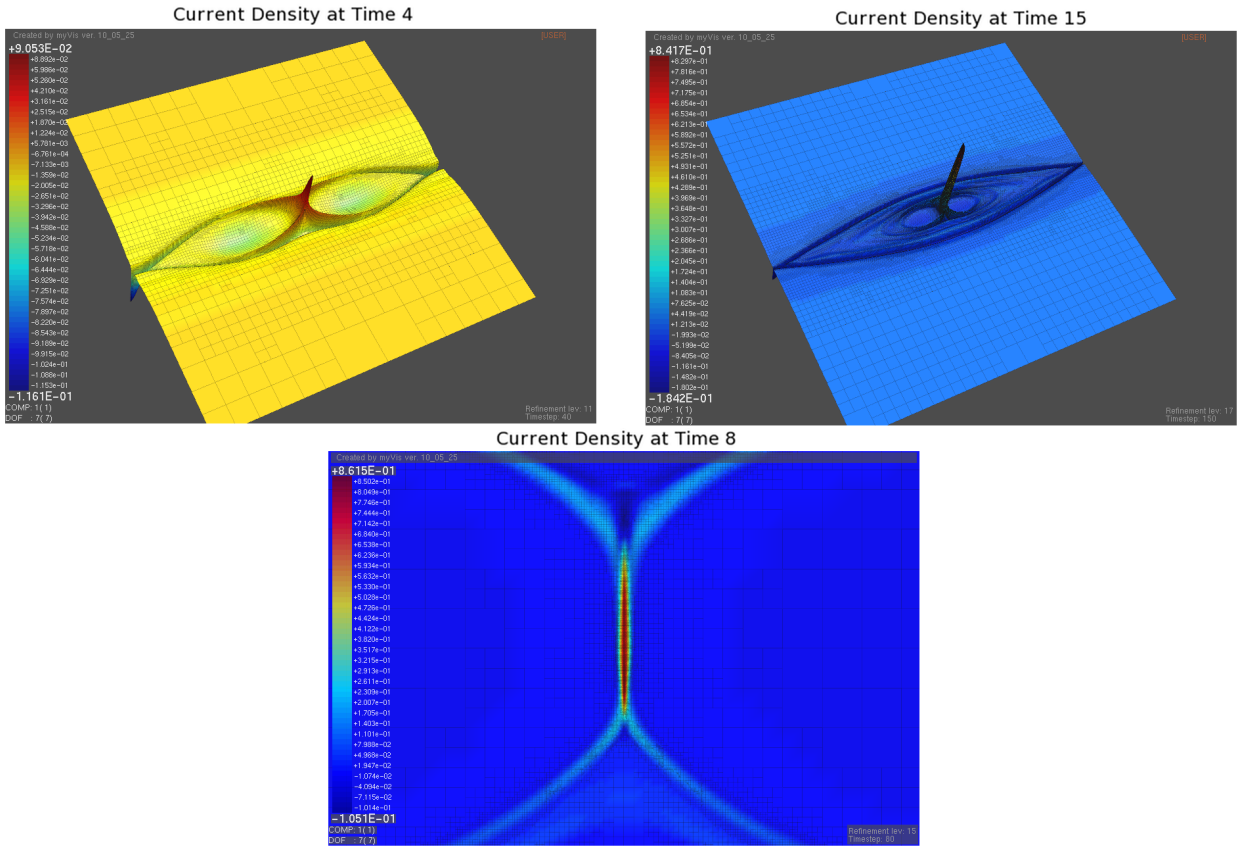


Figure 3.7: Numerical solution using adaptive refinement. $S_L = R_e = 50,001$. Top Left: Current Density at Time $4\tau_A$. Top Right: Current Density at Time $15\tau_A$. Bottom: Zoomed in plot of current density peak at Time $8\tau_A$.

relative to uniform refinement. The functional is reduced to the same order of magnitude in all

cases. The work at one time step is calculated by first determining the work of all the V-cycles on a given refinement level for that particular scheme. These values, times the number of matrix nonzeroes for the level, are then summed over all grids and divided by the number of nonzeroes on the finest refinement level for the given problem. In Table 3.7, the work unit values given are with respect to the finest level of the given refinement scheme. They are an average over all time steps. To compare two schemes, the average work unit value is multiplied by the fine-grid nonzeroes for that scheme and then the ratio is taken. This ratio is defined as the Work Ratio in Table 3.7. Similarly, the Element Ratio column is the ratio of elements on the finest grid of the adaptive scheme compared to the number of elements on the finest grid of the uniform scheme.

Uniform		Ratio to Uniform	
Work Units	Avg Elements		
80.473	13,380		
ACE			
Work Units	Avg Elements	Work Ratio	Element Ratio
9.789	1,779.9	0.12	0.13
ACE-DOF			
Work Units	Avg Elements	Work Ratio	Element Ratio
29.610	3,040.7	0.37	0.23
ACE-Reduc			
Work Units	Avg Elements	Work Ratio	Element Ratio
23.513	3,083.0	0.29	0.23
NACE			
Work Units	Avg Elements	Work Ratio	Element Ratio
22.907	2,895.2	0.28	0.22

Table 3.7: Average number of work units per timestep using uniform refinement versus various ACE refinement. All values are relative to finest grid of uniform refinement. A total of 45 time steps were performed to compute the averages.

The results show that using adaptive refinement greatly reduces the amount of work needed, compared to that of using uniform refinement. ACE requires 12% of the work that uniform refinement requires. The physics is more localized in this problem, especially by the time the reconnection begins to develop and, thus, the refinement is more localized. It appears that, for this problem, ACE gives the best efficiency. The functional is reduced to the same order of magnitude in all cases, but original ACE needs fewer elements. The ACE-DOF and ACE-Reduc schemes appear

to add unnecessary elements just to get a certain total number or to reduce the functional more than is needed. The NACE scheme also appears to be less efficient. In this case, although the NACE scheme puts the elements in almost the same places as the ACE scheme, and reduces the functional to the same level. It uses more work to get there. Because in some steps, it either over refines or under refines. For instance, it tends to refine more elements at coarser levels since large overall functional reduction is expected. At finer levels, the NACE scheme tends to refine less number of elements just in order to reach a small amount of overall functional reduction since the functional at the current level is close to the final functional tolerance. However, it is still on par with the ACE-DOF and ACE-Reduc schemes. Qualitatively, all 4 ACE schemes appear to capture the coalescence of the two islands; see Fig. 3.8.

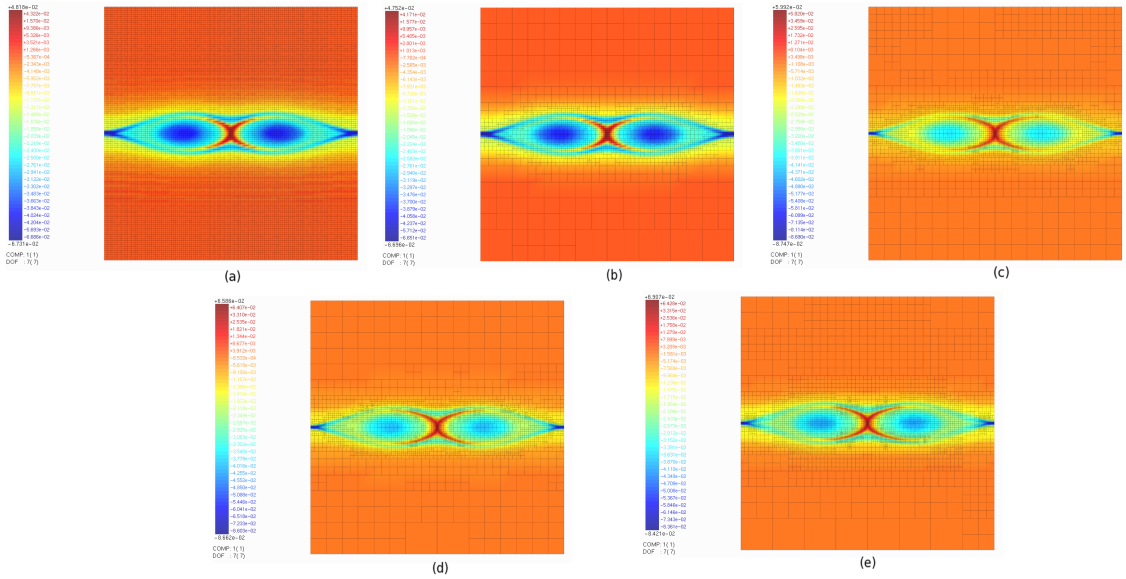
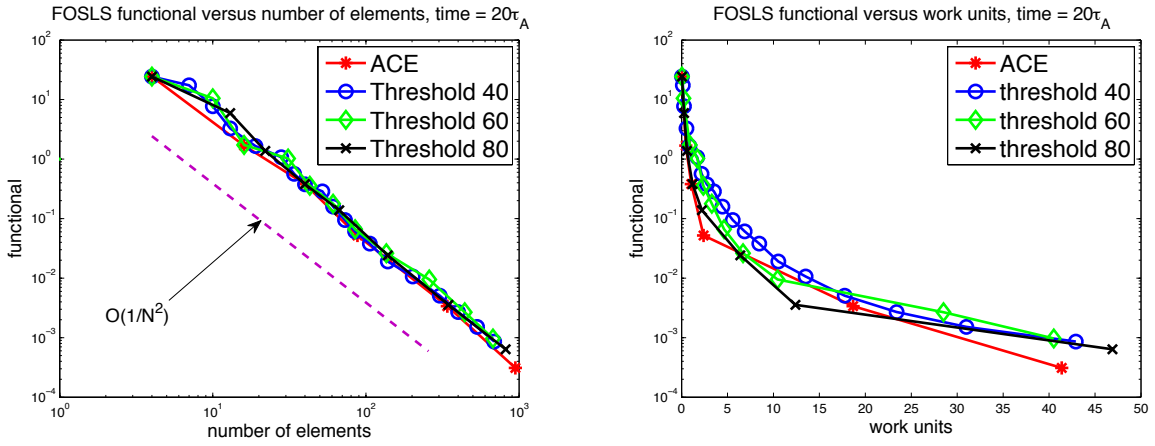


Figure 3.8: Current density at time = $40\tau_A$. (a) Uniform Refinement. (b) Original ACE. (c) ACE-DOF. (d) ACE-Reduc. (e) NACE.

Comparison to Threshold-based Schemes

As a comparison to the threshold-based schemes described above, the island problem was also run using these schemes with values of 40, 60, and 80%, for the number of elements to refine. Comparisons were made at various time steps throughout the run. While all different schemes captured the qualitative behavior of the island coalescence problem, the threshold schemes often required

more elements and more work units to resolve the problem to the same functional values. Figures 3.9 and 3.10 give a comparison of the schemes for time step 20 ($t = 2\tau_A$) and time step 80 ($t = 8\tau_A$), respectively. These figures show the relationship between number of elements on the finest grid versus functional and the relationship between the number of work units and functional. At time step 20, the solution is still rather smooth and ACE appears to get the optimal grid, requiring fewer elements and less work units to get to the same functional value as the threshold-based schemes. At time step 80, the reconnection has taken place and steep gradients have developed. At this time, all schemes appear to require more work and elements to resolve the physics. However, ACE is no worse than any of the threshold-based schemes. While ACE picked the optimal refinement pattern from the efficiency measures while running, the best threshold method required knowing the correct percentage ahead of time.



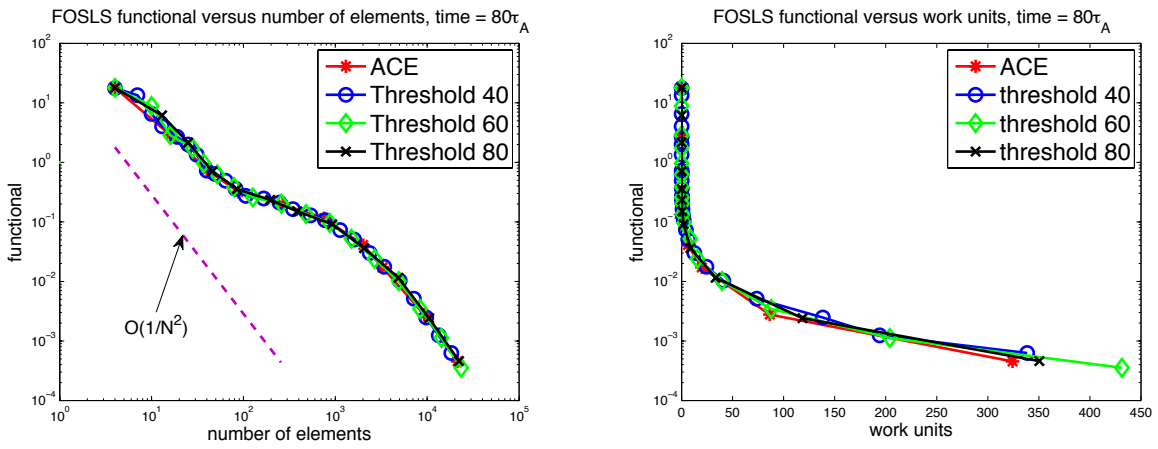
(a) Functional vs number of elements

(b) Functional vs work units

Figure 3.9: Comparison between ACE and threshold-based schemes at time step $t = 2\tau_A$.

3.3 Conclusions

In this chapter, efficiency-based refinement algorithms for the FOSLS finite element method with algebraic multigrid solvers in the context of nested iteration (NI-FOSLS-AMG) are developed. The algorithms choose which elements to refine based on optimizing computational efficiency, taking into account both error reduction and computational cost. Two efficiency measures are



(a) Functional vs number of elements

(b) Functional vs work units

Figure 3.10: Comparison between ACE and threshold-based schemes at time step $t = 8\tau_A$.

considered: predicted 'accuracy-per-computational-cost' (ACE) and the new 'anticipated-overall-computational-cost' (NACE). The use of the FOSLS local functional as a sharp a posteriori error estimate along with NI-AMG methods allows parameters to be computed that are used to estimate the current measures. In addition, several "flavors" of these efficiency-based schemes are tested to determine whether adding certain constraints to the efficiency measure, such as the total number of elements to add or the total amount of error to be reduced, would make it easier to obtain a near optimal grid. Numerical tests show that all of the efficiency-based algorithms effectively and efficiently capture local features of the solution. For the linear test problem, all schemes perform equally well, suggesting that the standard ACE scheme is sufficient without any extra constraints. For the more complicated nonlinear time-dependent MHD problem, this also is the case. In fact, the constrained schemes appear to at times perform unnecessary work, making them less optimal. However, all schemes greatly reduce the amount of computational cost for solving these problems to a specified accuracy compared to the cost of uniform refinement. In addition, in comparing the ACE scheme to threshold-based schemes, ACE either outperformed the threshold-schemes or was no worse than the best threshold-based method at any given time step. As the optimal refinement strategy varies over time steps, choosing a scheme such as ACE, which can adaptively choose the optimal refinement strategy is preferable in the case in which many time steps are needed and the

physics can change dramatically.

Several aspects still need to be studied. In this work, a generic AMG solver was used. Deterioration in the AMG convergence for increased timestep size as well as Reynolds and Lundquist numbers are observed in the MHD test. Even a slight improvement in the AMG algorithm would greatly reduce the total work units required to achieve a specified accuracy. AMG algorithms specifically designed for systems of PDEs are a topic of current research. This might involve the use of newly developed adaptive multigrid algorithms described more in [11, 12]. In addition, the hierarchy of the grids resulting from adaptive refinement might be used to reduce or eliminate the set up phase of AMG at each level. A new multigrid solver might be developed for problems arising from adaptive refinement procedures. This would involve including more of the geometry or structure of the grids into the multilevel solver. Since the problems that would use such schemes, such as MHD, which is used in a variety of applications, including fusion energy physics and space weather, are gaining increased interest, it is to reasonable to tune the numerics for such specific problems.

Many aspects of the adaptive refinement algorithms can be improved. The FOSLS approximation heuristics introduced in Chapter. 3.1.1 require certain smoothness assumptions. When the solution contains singularities, for instance, one might want to adaptively determine the strength of the singularity and appropriately apply graded refinement techniques rather than splitting elements into subelements with equal size in each direction. This will be explored in future research.

Finally, it is observed that more computational resources are required at certain time steps in order to fully resolve the local physics near the reconnection zone during the MHD simulation. This makes the parallelization of the NI-Newton-FOSLS-AMG-ACE approach necessary. To efficiently construct a sequence of grids that converge to a near optimal grid, communication cost must be taken into account. Modifications of the ACE algorithm to yield efficient parallel implementation involve grouping elements based on local functional values. Load balancing issues are addressed by redistribution of elements and nodes based on a space filling curve (SFC). The use of SFC preserves locality properties of local grids and thus reduces communication cost. Communication patterns

and interactions between local grids are managed by parallel tree (or forest) structures. Details of the parallelization and evidences of its efficiency are discuessed in the next chapter.

Chapter 4

Parallel Implementation

To accommodate the continual need for greater computing power, it is imperative to implement NI-Newton-FOSLS-AMG and the efficiency-based ALR algorithms in parallel for two- and three- dimensional problems. In this chapter, the parallel implementation is detailed. Clearly, a global sort of the local functional values is not efficient, especially in a massively parallel environment. A binning strategy is developed that groups elements according to local functional values. Then, marking decisions are made for bins based on the most accuracy-per-compuational-cost. Options for binning strategies and the parallel efficiency-based ALR algorithms are discussed in Chapter 4.1.

Scaling adaptive local refinement in parallel on thousands of processors is considered a big challenge. Efficient parallel mesh structures and algorithms are crucial to overcome difficulties in communications, load balancing, and grid interactions. Chapter 4.2 describes details of the parallel tree (or forest) based mesh structure, load balancing algorithms based on a space filling curve, and the grid partitioning strategies.

Next, in chapter 4.3, we describe the software package that implements NI-FOSLS-AMG-ACE-like algorithms in parallel. Numerical tests including Poisson equation with steep gradients and flats, backward facing step Stokes problem, and backward facing step Navier-Stokes problem are presented in chapter 4.4. We study both the numerical performance and the parallel scalability in various of situations. Results show that the NI-FOSLS-AMG approach with the parallel efficiency-based ALR algorithm is capable of reaching the same accuracy with significantly less computational

work as well as CPU time with uniform refinement. Parallel scalability up to $O(10^3)$ processors are also demonstrated.

4.1 Parallel Efficiency-based Marking Strategies

4.1.1 Parallel Binning Strategies

The main difficulty with parallelization of ACE-like algorithm is the global sort of local functional values. The usual way to deal with this problem in parallel is binning (or coloring). The heuristics developed in chapter 3 indicate that refining elements that contains similar local functionals would result in similar error reduction per cost. Therefore, it is reasonable to group elements according to local functionals into bins, and treat each bin as an abstract element in the ACE-like marking schemes. The heuristics on estimate of error reduction and work, the ideas of multiple refinements, and marking decisions based on optimization of efficiencies are easily extended. The question now becomes

- What binning strategy should be used?

To answer this question, two major aspects are considered:

- Elements are grouped together such that the marking schemes produce results similar to those without using bins.
- The binning strategy should be capable of greatly reducing communication cost in parallel.

Recall how the efficiency-based marking algorithm works in serial. The marking decision is made by finding the optimum of a given efficiency measure function that depends on how local functional are distributed, i.e., the functional distribution function, $E(r)$, given by (3.8). A binning strategy can be seen to be equivalent to applying a piecewise linear interpolation, $I_h E(r)$, to $E(r)$, (see Fig 4.1). Since the efficiency function (either ACE or NACE) is a smooth function depending on $E(r)$, decisions based on optimizing the efficiency using $I_h E(r)$ should produce similar results if

$I_h E(r)$ is close to $E(r)$. Now, the choice of a binning strategy becomes the question of how best to interpolate $E(r)$ with as few nodes as possible.

Toward that end, let $\epsilon^2_{\max}$ and $\epsilon^2_{\min}$ be the maximum and minimum **nonzero** local functional value among all elements, respectively. Here, elements that contain zero local functional values always land to the last bin. Let $nBin$ be the number of bins to be created. Consider the following binning strategies:

(1) Geometric binning: choose $0 < q < 1$, create bins such that the first bin has elements with local functional values in the range of $[\epsilon^2_{\max}, q\epsilon^2_{\max}]$, the second bin has elements with local functional values in $[q\epsilon^2_{\max}, q^2\epsilon^2_{\max}]$, etc.

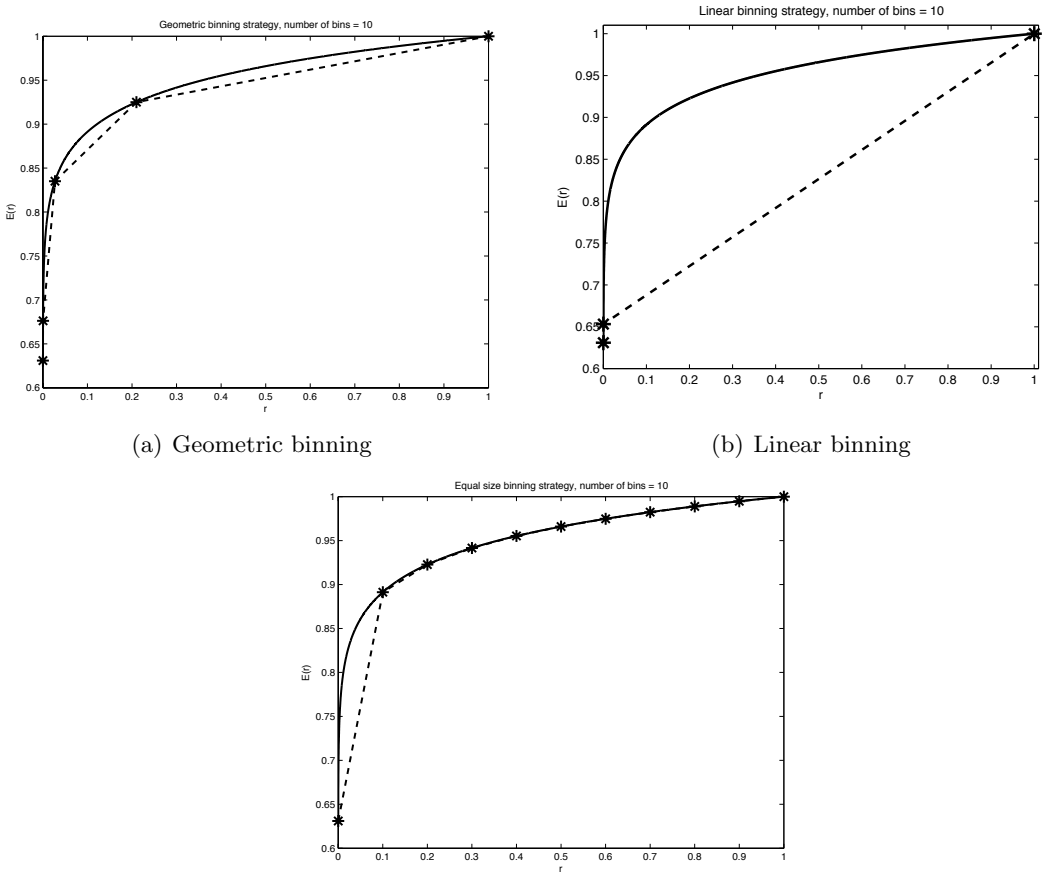
(2) Linear binning: create bins such that the i^{th} bin has elements with local functional values in the range of $[\epsilon^2_{\max} - (i-1) * \delta, \epsilon^2_{\max} - i * \delta]$.

(3) Equal size binning: create bins such that the number of elements in each bin is roughly a constant and such that elements in the i^{th} bin contain local functional values no less than elements in the $(i+1)^{\text{st}}$ bin.

When error distribution is close to uniform among elements, all binning strategies would results in the same decision: nearly uniform refinement. We care about the case when large error concentrates in a small fraction of elements. In this situation, good piecewise linear interpolation to such function requires more resolution near zero, where the derivative of the function to be interpolated is large. It is easy to see that geometric binning makes such an interpolation; see Fig 4.1. Linear binning does well to approximate the curve close to 0, but tends to group all of the rest elements into only the last bin. Equal size binning, on the contrary, tends to group elements with small local errors into different bins, which is not necessary.

The next question is the choice of the proper q for the geometric binning strategy. For a given number of bins, say $nBin$, one can compute q such that

$$q^{nBin}\epsilon^2_{\max} = \epsilon^2_{\min} \qquad \Rightarrow q = e^{\left(\log \frac{\epsilon^2_{\min}}{\epsilon^2_{\max}}\right)/nBin}. \tag{4.1}$$



(a) Geometric binning

(b) Linear binning

(c) Equal size binning

Figure 4.1: Binning strategies applied to functional distribution: local error concentrates in the first a few elements.

Geometric Binning Strategy

The discussion below shows that q can be chosen based on the smoothness of the solution and the order of finite elements, which can dynamically determine the number of bins needed. Consider element τ_i of polynomial degree p. Assume τ_i has local functional ϵ_i^2. A single h-refinement of τ results in 2^d children, namely $\tau_{i,j}$, $j = 1, 2, ..., 2^d$. FOSLS approximation heuristics in chapter (3.1.1) show that each child element is expected to have local functional

$$\epsilon_{i,j}^2 \approx \frac{1}{2^{2p+d}} \epsilon_i^2, \qquad \text{for } j = 1, 2, ..., 2^d. \tag{4.2}$$

We argue that one can choose $q = \frac{1}{2^{2p+d}}$ so that if an element in the i^{th} bin is refined, and the $(i+1)^{\text{st}}$ bin is not refined, then children elements from refining the i^{th} bin will land in the $(i+1)^{\text{st}}$

bin. Doing this hopefully results in only a small number of bins on finer levels, which leads to equal distribution of error and nearly uniform refinement, which can reduce or eliminate load balancing issues on finer grids. To finish the discussion of binning strategies, we present the algorithm with $p = 2$ and $d = 2$. It is straightforward to give the algorithm for other combinations of p and d.

ALGORITHM 8. *Parallel Geometric Binning Algorithm*

(1) MPI_Allreduce to get maximum local functional $\epsilon_{\max}$.

(2) Set up bins such that the i^{th} *bin consists of local functional range* $[(\frac{1}{64})^i \epsilon_{\max}, (\frac{1}{64})^{i-1} \epsilon_{\max}]$.

(3) Each processor counts its local contribution to each bin: number of owned elements and local functional values.

(4) MPI_Allreduce again to get the global bin information: number of elements and functionals in each bin.

Communications only occur at step (1) and step (4) and only consist of a few integers and double precision numbers. Although the communications are *Allreduce*-type, since the message sizes are quite small, they can be done quickly through $Butterfly$-like algorithms.

4.1.2 Parallel Efficiency-based Marking Strategies

Parallel ACE-like marking algorithms are described:

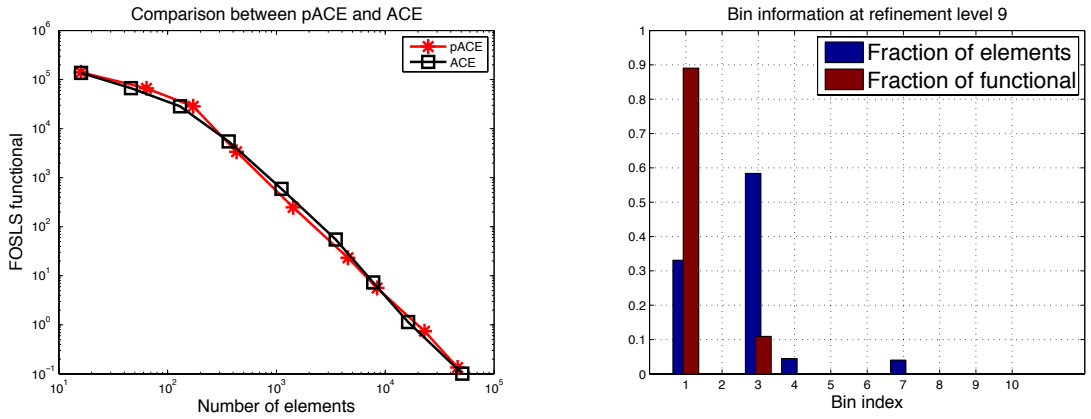
ALGORITHM 9. *Parallel ACE (pACE). At each refinement level*

(1) Create geometric bins using algorithm (8).

(2) Each processor treats bins as abstract elements, mark bins for refinement based on the most accuracy-per-computational-cost.

(3) Each processor finds its local elements in the marked bins, and mark them for refinement.

Apparently, communication only happens in step (1), which is almost negligible. The parallel NACE algorithm is similar, the only difference is that the NACE efficiency measure is used in step (2). For the other two variations of ACE, ACE with fixed DOF and ACE with fixed Error reduction, the geometric binning strategy might not work well. Instead, equal size binning is more suitable for ACE-DOF. The binning strategy in which each bin contains almost the same amount of local functional value is better for ACE-Reduc. Since the serial tests show that all ACE-like algorithms work similarly, this thesis focuses on the performance of the pACE algorithm. Research on the different types of binning algorithms and variations of the pACE algorithm remains for future study.



(a) Functional vs number of elements

(b) Bin information

Figure 4.2: Comparison between ACE and pACE: FOSLS functional vs number of elements for Steep Gradients.

To finish this section, we compare the numerical results of pACE and the serial ACE algorithm applied to the Poisson problem with steep gradients. Starting with 4×4 biquadratic elements, the FOSLS functional versus the number of elements for ACE and pACE are plotted in Fig. 4.2(a). The pACE algorithm produces almost identical result as ACE in terms of error per DOF. One of the objectives of geometric binning is to reduce the number of bins at fine levels, which is confirmed in Fig. 4.2(b). It is also observed from Fig. 4.2(b) that most elements are grouped in the first three bins. This shows that the pACE algorithm, using the geometric binning strategy, tends to equally

distribute local errors at finer levels. More evidence will be given in the numerical test section.

4.2 Parallel Adaptive Mesh Refinement

In [23], Most popular parallel AMR methods fall into two categories, structured AMR (SAMR) and unstructured AMR (UAMR) (see [23]). SAMR methods represent the PDE solution on a composite of nested and structured grids (or patches as they are called in SAMRAI library [44]). Solution accuracy is maintained by careful interpolation between the nested grids, which generally results in difficulties for high order elements and irregular domains. Load balancing issues, communication patterns among patches, and grid interactions are challenges to scale dynamic SAMR in parallel. Examples of SAMR implementation include SAMRAI [44], Chombo [21], and PARAMESH [30]. SAMR methods have been shown to scale to $O(10^3)$ processors. Since pFOSPACK allows high-order discretization and irregular domains, SAMR is not a good choice.

Contrary to SAMR methods, UAMR methods typically employ a single (often conforming triangle- or tetrahedral-) mesh generated by locally refining and/or coarsening elements. High-order elements and irregular meshes are no longer difficulties. However, maintaining the conforming property in the refinement process is rather complicated and expensive in parallel. One example of UAMR is Pyramid [35]. There are a few reasons to not choose UAMR. First, pFOSPACK employs quadrilateral or hexahedra elements. Secondly, maintaining conforming meshes in the refinement process is not required, i.e., hanging nodes are allowed and the continuous solutions are obtained through constraints between slave and master nodes. Lastly, FOSLS is particularly amenable to nonconforming finite element spaces. Therefore, choosing UAMR methods results in extra overhead but does not improve the convergence of the FOSLS approximations.

In addition to the SAMR and UAMR methods mentioned above, parallel AMR techniques that use space filling curves and parallel tree (or forest) structures were implemented and analyzed in [42]. These methods are considered intermediate between SAMR and UAMR. Therefore, we call these methods semi-unstructured AMR (SUAMR) in this thesis. SUAMR methods usually employ quadrilateral or hexahedra elements. Hanging nodes are allowed during refinement and

are handled through master-slave constraints and the 2-1 balance refinement. When SUAMR were initially introduced, they usually required the domain to be embedded into a unit square(or cube), and the coarse mesh to be uniform such that each element can be associated with a leave of a complete quadtree (or octree in 3D). Adaptively refining the mesh is then equivalent to refining the associated tree. Pre-order traversal of all leafs results in a Lebesgue space filling curve that connects all active elements, which can then be used for load balancing and grid repartitioning. An example of SUAMR implementation is the Octor library [42]. To extend SUAMR methods for irregular domains the whole domain is first broken to pieces. Then each piece is embedded in or transformed to a regular domain. Separate tree structures and space filling curves are generated for each piece. Next, trees are grouped to form a forest by connecting segments of space filling curves. Adaptive refinement of the mesh is equivalent to refining the forest. An example of SUAMR implementation for irregular domains is ALPS [15], which is built upon Octor. Results in [15] show that SUAMR methods are scalable to $O(10^4)$ processors. SUAMR methods greatly simplify the implementation compared to UAMR methods. On the other hand, SUAMR methods are capable of dealing with high-order elements and irregular domains in the contrary of SAMR methods. Since the purpose of pFOSPACK is to provide a parallel scalable software for high-order elements and irregular domains, SUAMR methods are used in the implementation.

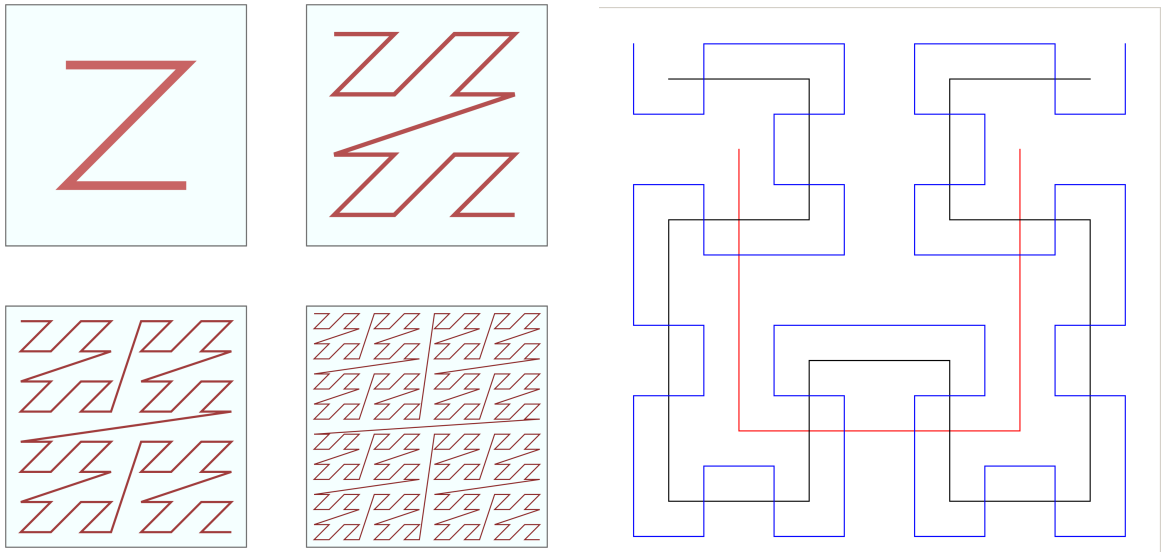
4.2.1 Space Filling Curves, Parallel Tree Structure and Load Balancing

Details of SUAMR methods and implementation in pFOSPACK are described in this section. They are organized as follows:

- Space filling curves, parallel tree structure, grid partitioning and load balancing,
- 2-to-1 balance refinement.

Algorithms are illustrated through a simple example in the unit square, but, as mentioned above, it can be generalize for irregular domains.

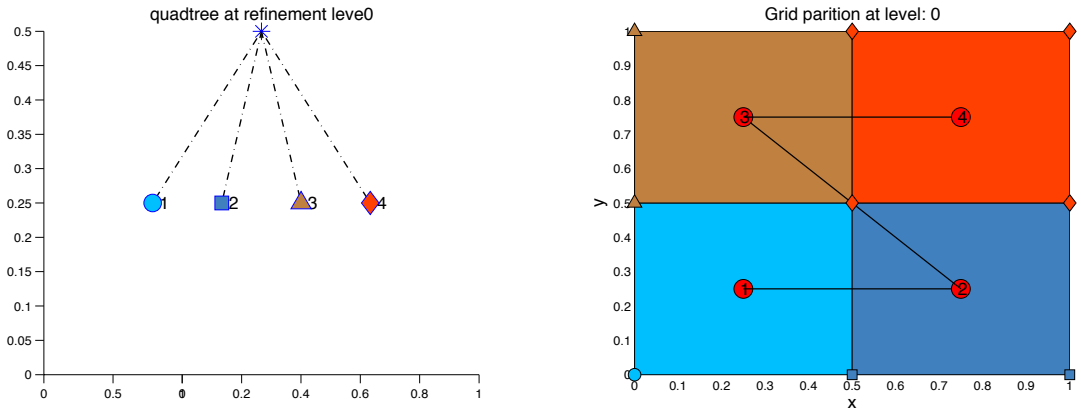
A space filling curve (SFC) is a curve whose range contains the entire 2-dimensional unit square (or more generally an N-dimensional hyprecube). Due to its good locality-preserving behavior, it is often used in for grid partitioning in parallel to reduce communication cost. Popular SFCs include Lebesgue curve (or Z-curve or Morton-order), and the Hilbert curve (see Fig. 4.3).



(a) Lebesgue curve (b) Hilbert curve

Figure 4.3: Space filling curves

The Lebesgue curve is particularly interesting because of its correspondence to the quad tree structure. For example, starting with one square element that covers the unit square, a single h-refinement results in 4 children. The parent-children correspondence can be represented by a quad-tree, see Fig. 4.4(a). Suppose that leaves associated with each child element is ordered according to lexicographic order. A pre-order traversal of the quad-tree gives exactly a Lebesgue space filling curve that connects the four children elements. If the grid is partitioned to 4 processors, one can partition the curve into four equal segment, and assign the i^{th} segment to the i-processor (see Fig. 4.4). Now, suppose element #1 is refined, which gives four children elements, #5, #6, #7, #8. The corresponding tree is expanded (or refined), performing a pre-order traversal of the active



(a) Quadtree at the coarsest level

(b) Initial grid partition

Figure 4.4: Initial grid partition based on space filling curve.

leaves gives the following order,

$$\#5 \rightarrow \#6 \rightarrow \#7 \rightarrow \#8 \rightarrow \#2 \rightarrow \#3 \rightarrow \#4,$$

which gives a Lebesgue curve that connects active elements in the refined mesh. Equal partition of the curve yields the new partition of elements

$$(\#5, \#6) \rightarrow \text{ proc } \#0$$

$$(\#7, \#8) \rightarrow \text{ proc } \#1$$

$$(\#2, \#3) \rightarrow \text{ proc } \#2$$

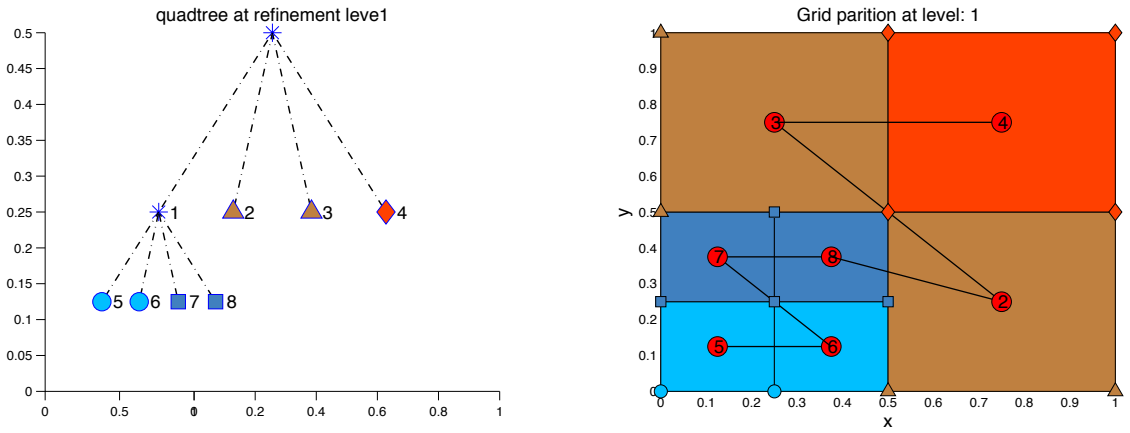
$$(\#4) \rightarrow \text{ proc } \#3$$

The next step is to decide owners of nodes, this is done based on the current partition of elements. The following rules are used

ALGORITHM 10. *Partition of nodes*

(1) Bottom-left node of each element is owned by the processor who owns the element.

(2) Nodes who have not been assigned owners in step 1 are owned by the processor who owns the majority of the adjacent elements. If more than one such processors exist, then the processor with higher rank becomes the owner. For example, node $(0.5, 1.0)$ *in Fig 4.5(b) is owned by processor* #3 *(red) instead of processor* #2 *(chocolate).*



(a) Quadtree after local refinement

(b) Grid partition after local refinement

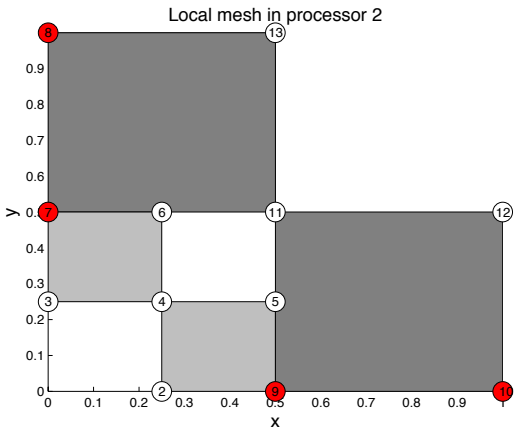
Figure 4.5: Grid partition after refinement. Owners of elements and nodes are assigned based on SFC.

Now that we know owners of elements and nodes, communications are performed according to the new partition. After that, global numbers are assigned to each node such that nodes owned by processor i always come after nodes owned by processor $i-1$. For example, in Fig. 4.6, processor #0 owns nodes #1, #2, processor #1 owns nodes #3, ..., #6, and so forth. Our implementation assumes that each processor is responsible for assembling the rows associated with unknowns at owned nodes. This leads to extra ghost nodes and elements might need be gathered from other processors. Ghost nodes and elements are defined as not owned by this processor, but are connected to any owned node. They are illustrated in Fig. 4.6 as nodes colored in white and elements colored in light grey. The very last step is to remove redundant nodes and elements, such elements and nodes are defined as not connected to any owned node. This step is to save memory usage. To summarize, the grid partitioning and load balancing methods are described as follows:

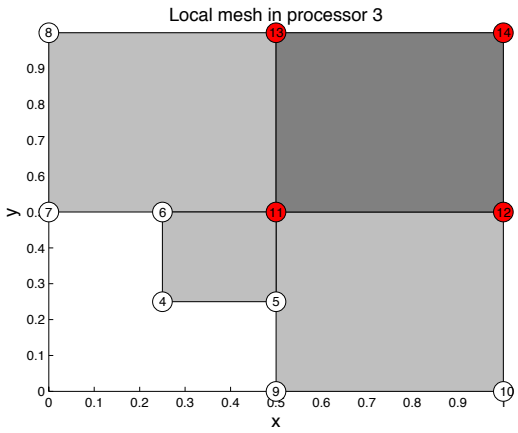
ALGORITHM 11. *Grid partition and load balancing*

After each refinement level, do

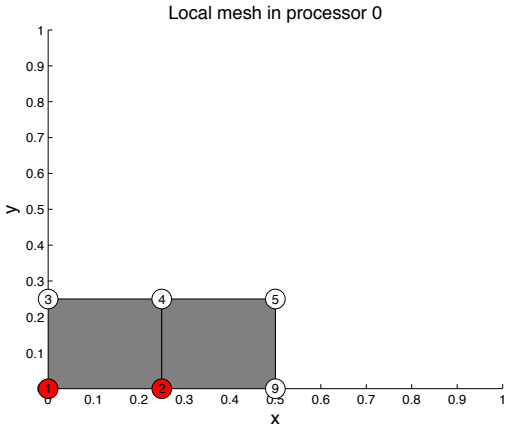
(1) Refine the parallel tree according to refinement: if an element is refined, then the corresponding leaf becomes a parent tree node, children leaves are added to the tree. This step is performed within each processor, no need for communication.



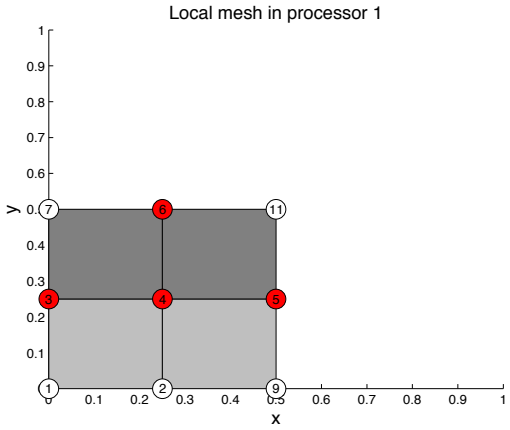
(a) Local mesh in processor 2



(b) Local mesh in processor 3



(c) Local mesh in processor 0



(d) Local mesh in processor 1

Figure 4.6: Local mesh in each processor after load-balancing: owned elements are colored in dark grey, ghost elements are colored in light grey, owned nodes are colored in red, and ghost nodes are colored in white.

(2) Each processor performs a pre-order traversal of the locally refined new tree. This gives a local space filling curve within each processor.

(3) Now, each processor knows the size of its local SFC. Perform an MPI_Allreduce to get the local SFC size from every processor.

(4) Create a new partition of elements by dividing the SFC into roughly equal segments. Use the new partition of elements to decide a new partition of nodes, i.e., assign an owner to each node using algorithm 10.

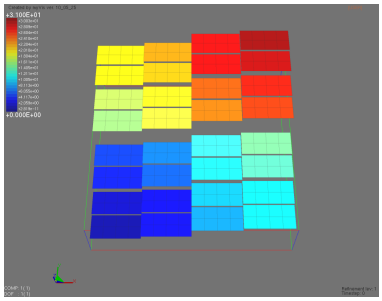
(5) Send and receive nodes and elements based on the new partition.

(6) Assign a global number to each node: each processor assigns global numbers to owned nodes first such that processor #i owns nodes $k, k+1, ..., k+\ell$ and processor #$i+1$ owns nodes $k+\ell+1, k+\ell+2, ..., k+\ell+m$.

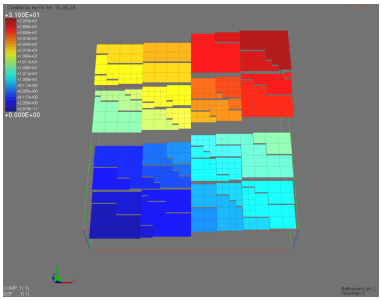
(7) Each processor builds local node-node connections and node-element connections and uses them to gather ghost nodes and elements.

(8) Each processor performs a cleanup step to remove nodes and elements that are not connected to any owned node.

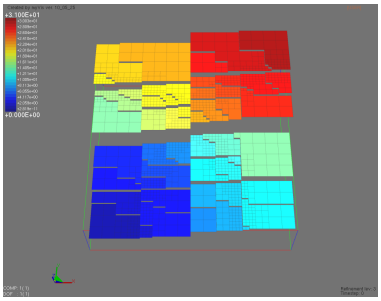
Most communications occur at step (5); however, we argue that since we start load balancing at coarser grids and pACE results in near uniform refinement at finer levels, communication cost would be small at finer levels. We finish this section by giving the grid partitions at different refinement levels of the Poisson problem with steep gradients running on 32 processors; see Fig 4.7. Although the refinement at level 7 is not close to uniform refinement, it is observed that the partition at level 7 and level 8 does not change dramatically, therefore the data movement among processors is small. Also, it is noticed that grid partition based on the Lebesgue space filling curve preserves the locality quite well.



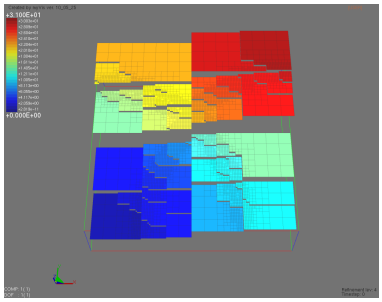
(a) Grid partition at level 1



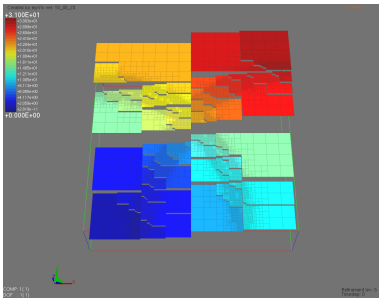
(b) Grid partition at level 2



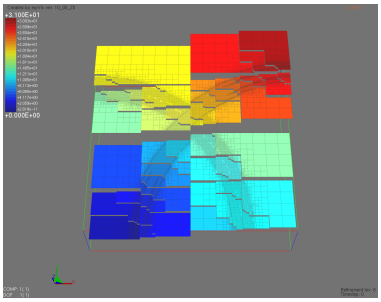
(c) Grid partition at level 3



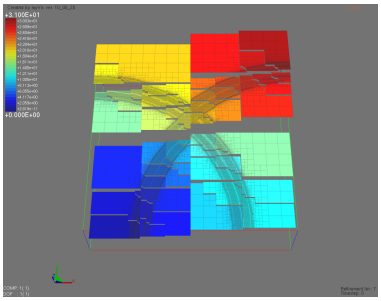
(d) Grid partition at level 4



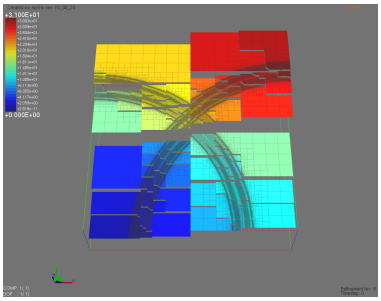
(e) Grid partition at level 5



(f) Grid partition at level 6



(g) Grid partition at level 7



(h) Grid partition at level 8

Figure 4.7: Grid partitions for Steep Gradients at different refinement levels based on SFC.

4.2.2 2-to-1 Balance Refinement in Parallel

The 2-to-1 balance refinement described in chapter enforces an extra marking step, (we call it clean-up), after the applying the standard pACE marking scheme. The main purpose is to improve AMG convergence rate as it is shown in Table. 4.1.

level	1	2	3	4	5	6	7	8	9	10
With clean-up	0.15	0.13	0.14	0.16	0.18	0.19	0.18	0.18	0.20	0.21
Withour clean-up	0.15	0.13	0.37	0.48	0.58	0.59	0.70	0.71	0.72	0.77

Table 4.1: Average Boomer AMG convergence for Poisson equation with steep gradients.

Difficulties arise from two aspects:

- It requires element-element correspondence: each element needs to know its neighbor elements that share edges. The introduction of hanging nodes complicates the algorithm,
- Ripple propagation effects, i.e., marking one element for refinement in one processor might require refinement of its neighbors, which may greatly increase the communication cost if not handled properly.

The first difficulty is handled easily using the parallel tree structure. Instead of sending and receiving edge and node information, one can find neighbor elements through parent and sibling nodes in the corresponding tree. In fact, element-element correspondence needs not be built from the scratch at each refinement level. It can be updated at the same time as the tree is refined. Only small communications are required. One can update the element-element correspondence within each processor, then cross processor correspondence can be figured out by a MPI_Gather-like algorithm followed by a $MPI\_Scatter$-like algorithm. For example, consider an element, τ, which is shared by more than one processors. Without loss of generality, we can assume that element τ, is owned by processor W, and is shared (as ghostelements) by processor S, E, and N. Here, we use W, S, E, N to label the processors such that processor W only contains the west neighbors, processor S only contains the south neighbors, and so forth. After local refinement within each

processor, each processor should update the neighbor elements on the west, south, east, and north side, respectively. In order to update the cross processor neighbor elements, one can first gather all updated neighbor information from processors S, E, N to the owner processor W. Now, processor W should have the updated neighbor information of τ in each side. Next, processor W scatters the updated neighbor information to processors S, E, N. Notice that communications only occur for elements that are shared among processors, i.e., elements that lie on processor boundaries. Notice that on finer levels, the number of elements that lie on processor boundaries is a lot smaller than the total number of elements in each processor. The communication cost for updating element-element correspondence is small. For example, in $\mathbb{R}^2$, let Ne be the number of elements and np be the number of processors. This gives roughly $\frac{Ne}{np}$ elements per processor. One can estimate the number of processor boundary elements in each processor as roughly $\sqrt{\frac{Ne}{np}}$. Noting that each quadrilateral element has four neighbors, the total communication cost is

$$O\left(2 \times 4 \times np\sqrt{\frac{Ne}{np}}\right). \tag{4.3}$$

Thi is, on average,

$$O\left(2 \times 4 \times \sqrt{\frac{Ne}{np}}\right). \tag{4.4}$$

per processor.

To overcome the second difficulty, the method described in [41] is employed, which also utilizes the parallel tree structure. The algorithm uses a two-stage balancing schemes. Local balancing on each processor is first performed, followed by balancing across the interprocessor boundaries. This process repeats until no more balancing is needed. The *prioritized ripple propagation* algorithm proposed in [42] is used for balancing interprocessor boundaries.

4.3 Parallel FOSPACK

One of the main contributions of this thesis is the implementation of the NI-Newton-FOSLS-AMG-pACE algorithm in parallel FOSPACK (*pFOSPACK*). pFOSPACK is the software package developed by the computational math group at CU Boulder. The package is written in Fortran

and C++ based on MPI. It uses the FOSLS methodology for PDE discretization and employs either the AMG solver BoomerAMG from HYPRE [25] or Smoothed Aggregation Multigrid solver *parSAMIS*, developed by the CU computational math group, to solve the discretized equations. The design goal is to develop a scalable, efficient, and easy-to-use PDE solver to support numerical simulations for terascale/petascale applications.

4.4 Performance Study

This section provides various numerical tests on Frost, a four-rack IBM Blue Gene/L system with 4096 computing nodes. Each node has two cores. Normally one core works for computing and the other one takes care of communication. Therefore, overlapping computation and communication is available for improving parallel efficiency.

We analyze both numerical performance and parallel scalabity of the NI-Newton-FOSLS-AMG-pACE algorithm together with SFC-based SUAMR. For the numerical performance, we focus on whether the results verify the following pACE heuristics.

- Refinements yields near equal distribution of error on finer levels, which leads to near uniform refinement.
- Starting load-balancing on very coarser levels helps to ameliorate load balancing issues at finer levels.

Meanwhile, results to verify the efficiency and effectiveness of the algorithms are also provided. They include error versus number of elements, figures of grid alignments at different refinement levels, comparisons to uniform refinement, etc.

For the study of parallel scalability, results of strong scaling and weak scaling are presented. Since each computing node of Frost only has 512MB memory, which makes testing strong scaling quite difficult since even the largest problem that fits in a small number of processors soon becomes less computationally dominant on a larger number of processors. To overcome this difficult, we break the strong scaling tests into a set of tests, which consist of a small, medium, and large

problem scaled on different numbers of processors. Weak scaling tests normally require a fixed problem size per processor as the number of processors increases. However, we can only keep the problem size per processor roughly a constant for parallel AMR. Proper scalings are performed to compute the weak efficiency. Details are described below.

4.4.1 Preliminaries of Parallel Scalability

Definitions of strong scaling, weak scaling, speedup, and parallel efficiency are reviewed very briefly here. Strong scaling usually refers to fixing a problem size, testing how the overall run time behaves with respect to increasing number of processors. Weak scaling means to increase the problem size when the number of processors are increased; most of time, problem size per processor is roughly fixed. Speedup is defined as

$$S_{np} = \frac{T_1}{T_{np}}, \tag{4.5}$$

where T_1 is the execution time of running the problem on a single processor and T_{np} is the execution time of running on np processors. In the ideal case, if no communication happens, if the problem is perfectly equally distributed to each processor, and ignoring all hardware bottlenecks, then $S_{np} = np$. This is called *ideal speedup*. For most problems, increasing the number of processors usually leads to more communication and less computation per processor. Speedup is expected to decay as the number of processors, p, increases. When one wants to test speedup, S_{np}, for large number of processors, np, usually a large test problem is used. Otherwise small problem leads to many idle processors that makes the test less meaningful. However, it is unlikely that the large problem can fit into one processor. T_1 becomes unavailable. In this situation, usually the problem is tested on the smallest possible number of processors that can hold the problem, denoted by $np_{\min}$. Then speedup is usually computed by

$$S_{np} = \frac{np_{\min} T_{np_{\min}}}{T_{np}}. \tag{4.6}$$

Formula (4.6) is mostly used in this thesis due to the memory limitations on Frost. Parallel efficiency, E_{np} is defined as

$$E_{np} = \frac{S_{np}}{np}. \tag{4.7}$$

In reality, parallel efficiency is mostly used for weak scaling. Assume the problem size per processor is fixed. Let $T_{w,1}$ and $T_{w,np}$ represent the overall runtime on a single processor and np processors, respectively. Parallel efficiency can be computed by

$$E_{np} = \frac{T_{w,1}}{T_{w,np}}. \tag{4.8}$$

$T_{w,np}$ is expected to increase due to communications, whereas E_{np} usually decreases. For our AMR tests, problem size per processor can only be roughly controlled; formula (4.8) must be modified. Let W_1 and W_{np} be the global problem size on 1 and np processors, respectively. Define the weak scalability efficiency as

$$E_{np} = \frac{W_{np}/(np \times T_{np,w})}{W_1/T_{1,w}}, \tag{4.9}$$

meaning work per processor per run time executed on np processors normalized by work per run time executed on a single processor.

4.4.2 Poisson Equation

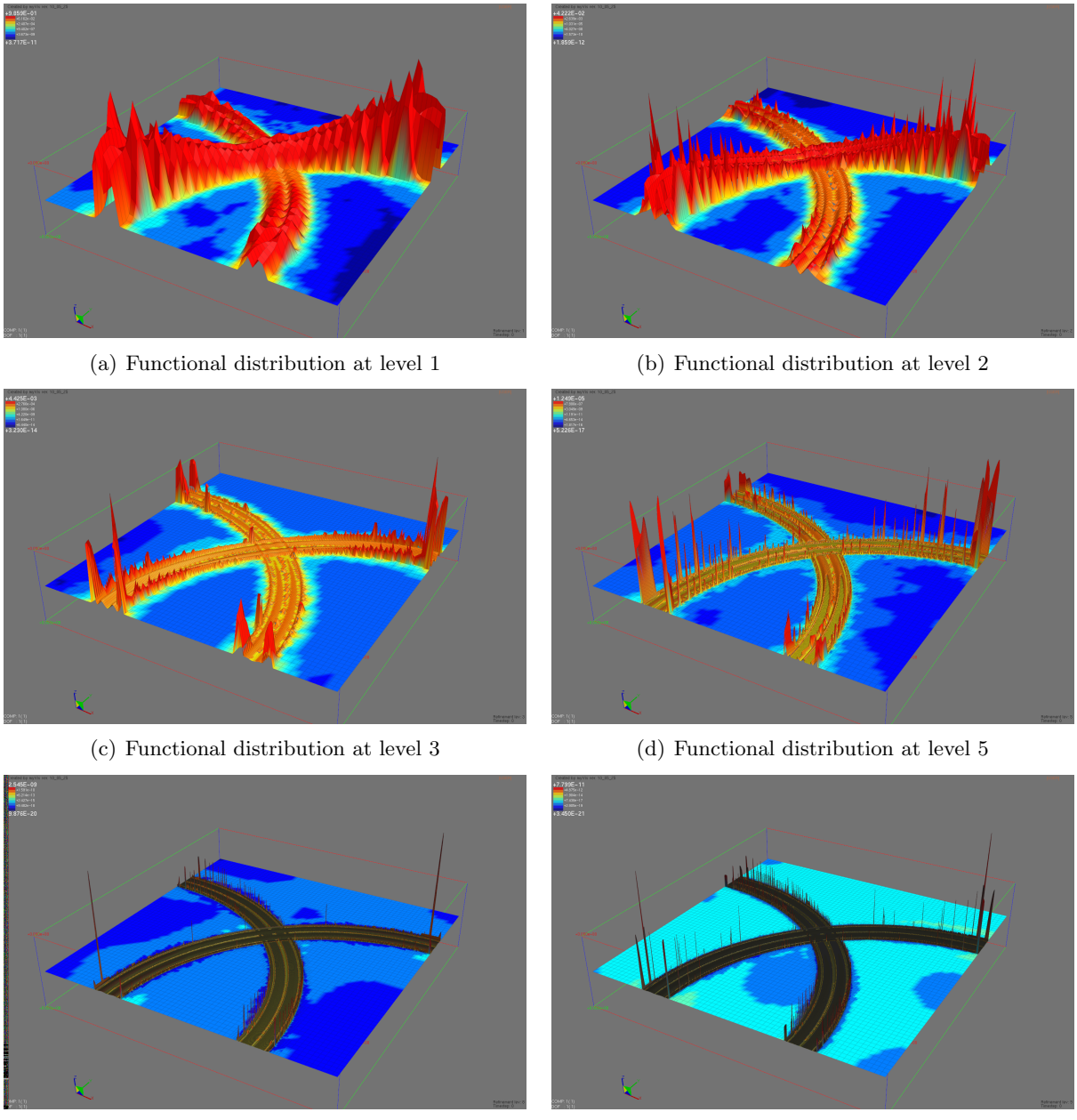
The first test problem is the Poisson equation with steep gradients and flats, defined in chapter 3.2. Test results are analyzed in two categories:

- Numerical performance is analyzed based on error convergence and work units.
- Parallel performance is discussed in terms of scalability and efficiency. Both weak scaling and strong scaling are tested.

4.4.2.1 Numerical Performance

Numerical results presented in this section come from a run on $1,024$ processors with an average of 4 elements on the coarsest grid. Refinement stops when the global FOSLS functional

reaches $O(10^{-6})$. Local functional distribution and grid alignment at different refinement levels are plotted in Fig. 4.8. Various relevant values with respect to each refinement level are tabulated in Table. 4.2.



(a) Functional distribution at level 1

(b) Functional distribution at level 2

(c) Functional distribution at level 3

(d) Functional distribution at level 5

(e) Functional distribution at level 8

(f) Functional distribution at level 9

Figure 4.8: Poisson with steep gradients: locally-refined mesh and functional distribution.

We make a few observations. First, the pACE algorithm behaves similar to the ACE algo-

ℓ	r_1	$E(r_1)$	N_ℓ	$\mathcal{G}_\ell$	$ncyc$	σ	ρ
1	1.00	1.00	4,096	2.43e+02	4	1.42	0.21
2	0.24	1.00	7,036	1.71e+01	4	1.46	0.21
3	0.50	1.00	17,944	1.33e+00	4	1.51	0.23
4	0.53	0.96	47,245	1.72e-01	4	1.52	0.23
5	0.90	1.00	176,224	1.18e-02	4	1.48	0.22
6	0.13	0.72	248,068	3.85e-03	4	1.49	0.23
7	0.93	1.00	944,353	2.50e-04	4	1.46	0.20
8	0.98	1.00	3,735,118	1.61e-05	4	1.44	0.19
9	0.99	1.00	14,814,997	1.03e-06	4	1.44	0.19

Table 4.2: NI-FOSLS-AMG-pACE for steep gradients: relative setup cost $C_s \approx 19.80$, setup 77.79 WU, solve 15.59WU, overall runtime 74.59 sec.

rithm in serial. It refines a large fraction of elements at coarser levels when grids are too coarse to resolve the local features of the solution. Later, at the intermediate levels, refinements are concentrated in a small fraction of the elements where large errors are detected; see refinement from level 5 to level 6 in Table. 4.2. After that, refinement becomes nearly uniform since the error is fairly equal-distributed; see level 8 and level 9. Efficiency of the algorithm can be reflected by either work units or CPU time. It takes just a little more than 1 minute to construct and solve the problem on a nearly optimal grid with roughly 15 million biquadratic elements, 60 million nodes with 180 million unknowns and order $O(10^9)$ nonzeroes in the finest grid matrix. Another observation is that refinement on the finest grid is almost uniform refinement, which requires zero load balancing. In Fig. 4.9, percentages of ALR functions, mark, refine, and balance, of overall runtime at each refinement level are given. On the finest level where near uniform refinement is performed, the load balancing cost is zero. In addition, because the load balancing starts on very coarse level, load balancing cost is controlled at finer levels. Time consumed by load-balancing at each refinement level remains below 6% of the overall runtime.

The comparison of the NI-FOSLS-AMG-pACE solution process to applying FOSLS-AMG directly to the constructed finest grid is given in Table. 4.3. The overhead is only 4.6%, which is better than the serial results. This is because refinement is close to uniform refinement at finer levels and that work increases by a factor of 4.0, which implies that the work on coarser levels is

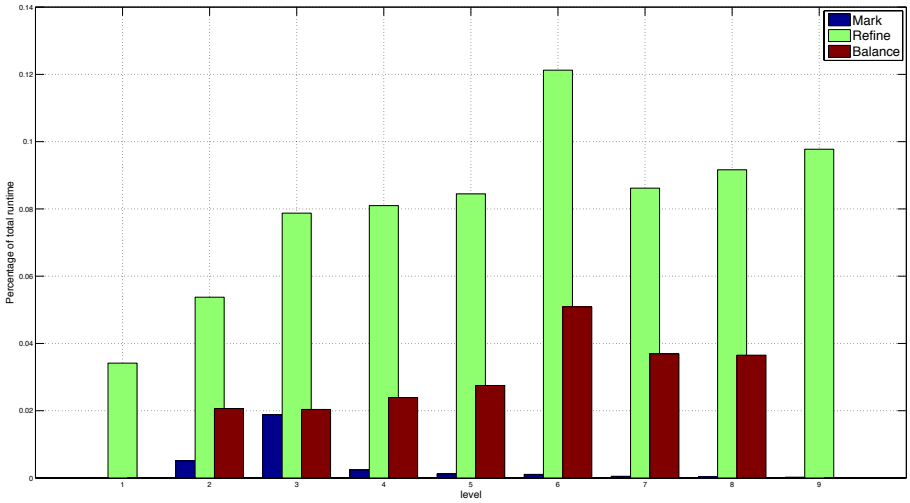


Figure 4.9: Percentage of ALR functions of total runtime per refinement level.

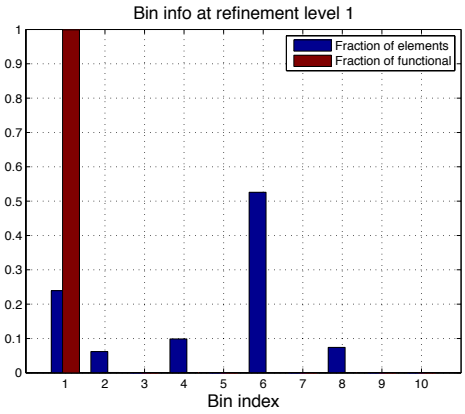
Method	Setup Cost	*ncyc*	Solve Cost	Total Work
NI-FOSLS-AMG-pACE	77.79	4	15.59	92.79
FOSLS-AMG	57.02	11	31.68	88.70

Table 4.3: Steep gradients: comparison of NI-FOSLS-AMG-pACE and applying FOSLS-AMG with random initial guess to the finest grid.

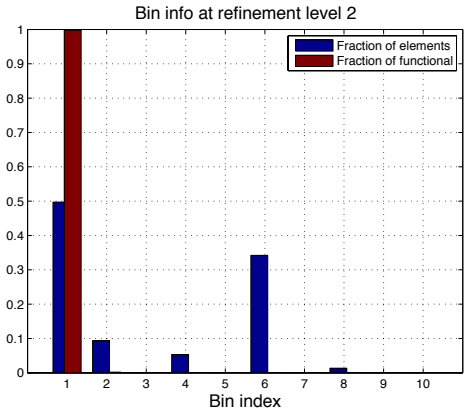
relatively less expensive compared to the work on the finest level.

To verify that pACE algorithm based on the geometric binning strategy results in near equal-distribution of error on finer levels, we give the binning results on each refinement level; see Fig 4.10 and Fig. 4.11. Refinement starts with more elements landing in lower bins at coarser levels (see Fig. 4.10(a) and Fig. 4.10(b)). Then since the first few bins are refined and error is more equally distributed, more and more elements land in higher bins; see Fig. 4.10(c) to Fig. 4.11(b). At the very fine levels, almost all elements land in the first one or two bins; see Fig. 4.11(c) and Fig. 4.11(d). This shows the nearly equal-distribution of error. The ratio between the largest local functional value and the smallest local functional value is bounded by $\frac{1}{64^2}$. Note that local error is the square root of the local functional. This gives that the minimum local error is about a factor $\frac{1}{64}$ of the maximum local error. A geometric binning strategy that creates bins less aggressively might give

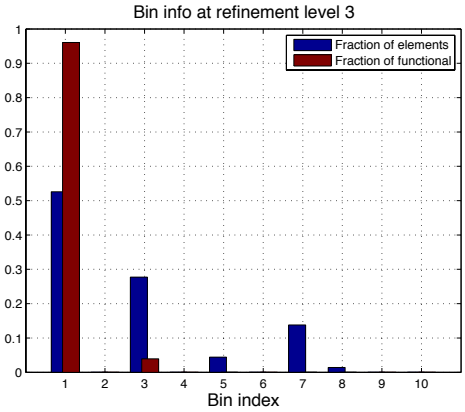
better results in terms of equal distribution of error. This remains for future study.



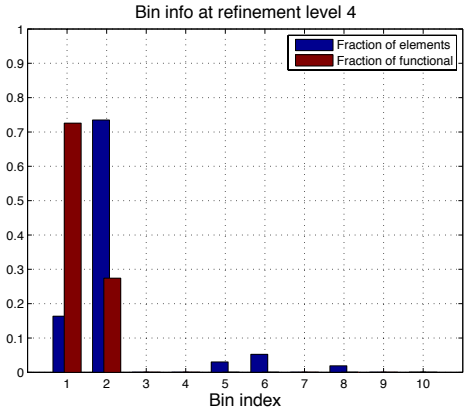
(a) Binning results at level 1



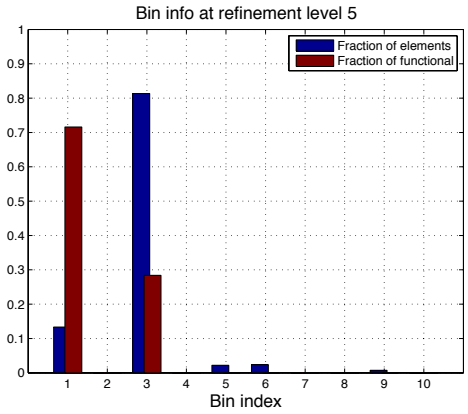
(b) Binning results at level 2



(c) Binning results at level 3



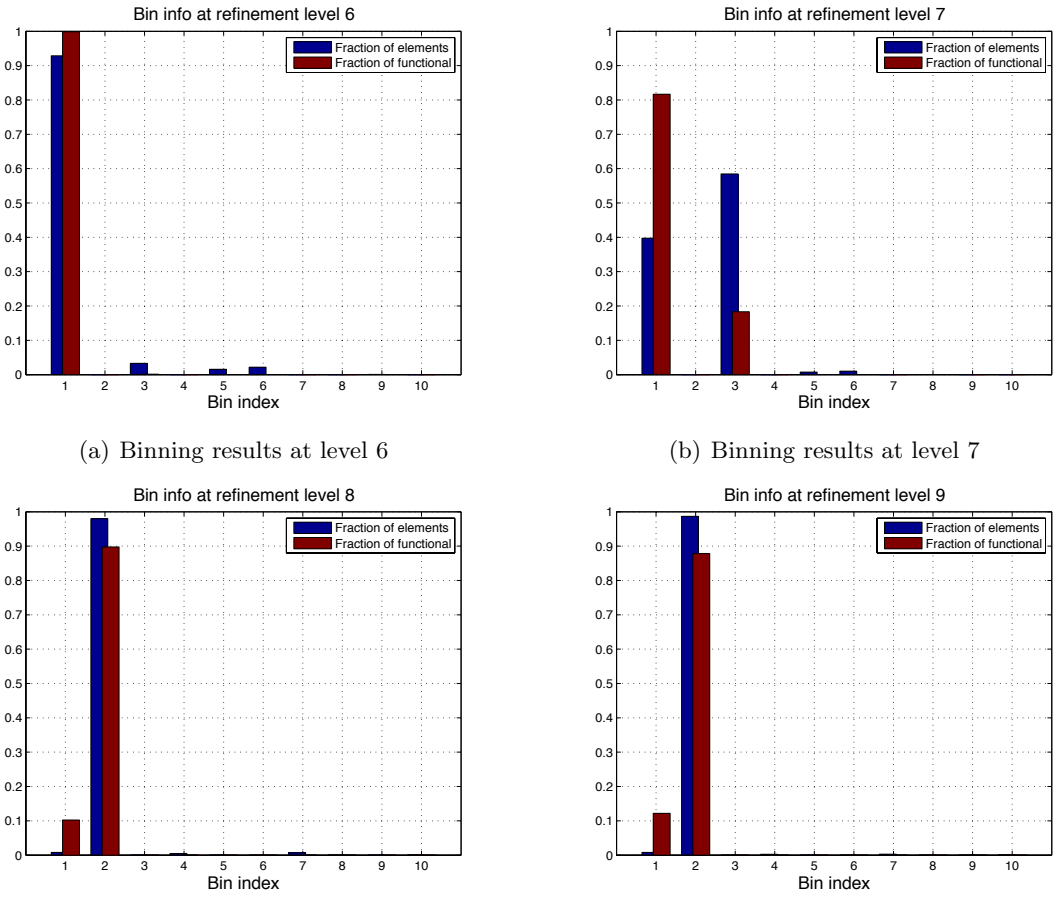
(d) Binning results at level 4



(e) Binning results at level 5

Figure 4.10: Steep gradients: geometric binning results from refinement level 1 to refinement level 5.

We conclude our discussion of the numerical performance by comparing pACE and uniform



(a) Binning results at level 6

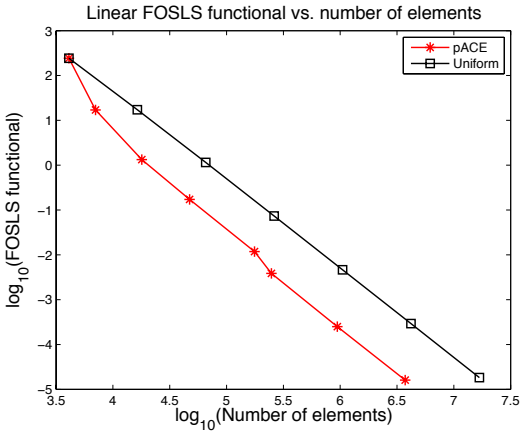
(b) Binning results at level 7

(c) Binning results at level 8

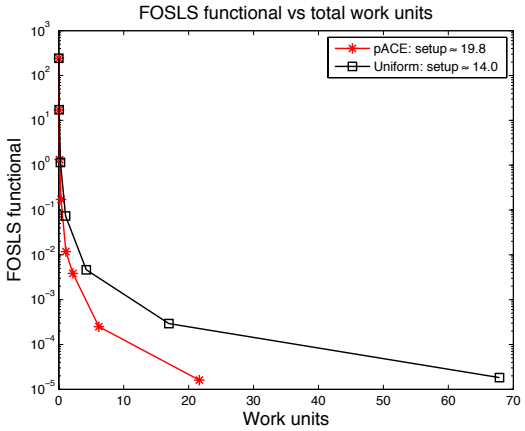
(d) Binning results at level 9

Figure 4.11: Steep gradients: geometric binning results from refinement level 6 to refinement level 9.

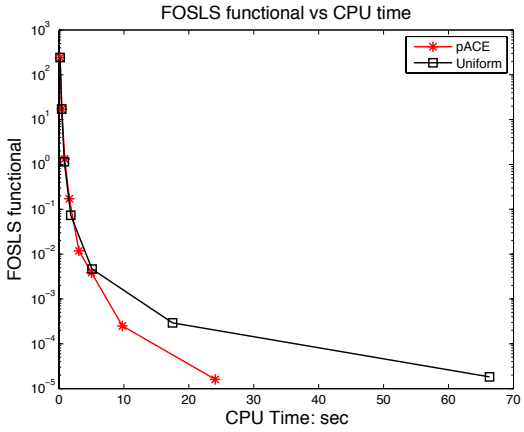
refinement; see Fig 4.12. First of all, pACE is able to reach the same accuracy with only 20% of the elements that were required by uniform refinement. Secondly, pACE only takes slightly more than 20 work units (including all costs), which is roughly $\frac{22}{68} \approx 35\%$ of uniform refinement. Lastly, measured in CPU time, the overall run time of pACE is only 30% of uniform refinement, considering that uniform refinement does not need to do any load balancing, the results show the NI-FOSLS-AMG-pACE approach is promising on $O(10^3)$ processors.



(a) Functional vs. Number of elements



(b) Functional vs Work units



(c) Functional vs CPU time

Figure 4.12: Steep gradients: comparison between ALR and uniform refinement running in 1,024 processors. Work unit is equivalent to one matrix-vector multiplication on the finest grid of uniform refinement.

4.4.2.2 Parallel Performance Study

Parallel scalability of NI-FOSLS-AMG-pACE together with SFC-based AMR is discussed in this section.

Strong Scaling

Test results of strong scaling for steep gradients are first discussed in this section. Speedups are given in Fig 4.13 for four different test problem. The smallest problem has 0.27 million elements, two intermediate problems have 1.06 million and 3.74 million elements, and the largest problem has 10 million elements. Results show that strong scaling speedups are close to optimal. For

instance the small problem with 0.27 million elements has speedup 100 at 128 processors, which is about a $100/16 = 6.25$ speedup over the runtime on 16 processors, (the optimal speedup is 8). For the largest problem of 10.52 million elements, the overall run time in $4,096$ processors results in speedup 3.0 over the runtime on $1,024$ processors, which is 75% of the ideal case.

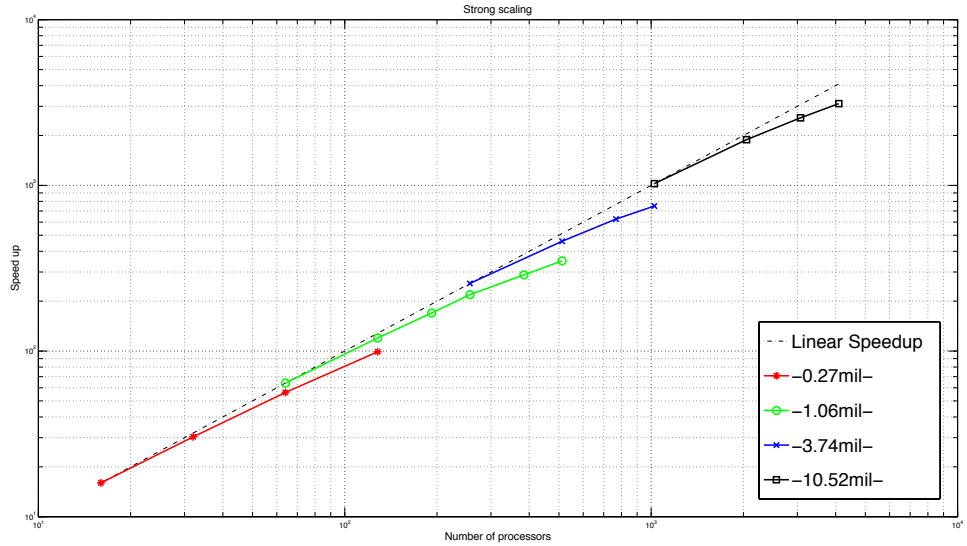


Figure 4.13: Strong scalability for steep gradients: speedups based on total runtime versus the number of processors for four different problem sizes. The largest problem has 10.52 million biquadratic elements, which is roughly $10,000$ elements per processor.

Weak Scaling

Figure 4.14 and 4.15 provide evidence of the weak scalability of pACE on the steep-gradient problem using pFOSPACK. Figure 4.14 plots the breakdown of the overall runtime into two major categories: numerical PDE solves (error estimate, matrix assembling, linear solver setup, and AMG-CG solve) and ALR functions (marking, refining, and load balancing). Problem size is roughly a constant per processor: roughly 15,000 biquadratic elements per processor at the finest level. The cost of all ALR functions are all controlled within 20% of the overall runtime from 2 to $4,096$ processors. Despite $np = 4,096$, runtime of all ALR functions is almost the same as the AMG-CG solves, which is nearly optimal for Poisson-type equations (only 4 iterations per refinement level). In other words, cost of all ALR functions is comparable to 4-AMG-CG iterations per refinement. For $np = 1,024$,

that is equivalent to only 15.59 work units.

Fig. 4.15 is the parallel efficiency for weak scaling from 1 to 4,096 processors. Here, parallel efficiency is defined as the total work units per processor per total run time for a given number of processors, normalized by the total work units per total run time for single processor. Work unit from the finest grid of $np = 4,096$ is used for all calculation. A parallel efficiency of 1.0 indicates perfect weak scaling. As we discuss before, it is hard to have exact weak scaling in ALR, we have to normalize the total work units per processor. Given that ALR functions remains roughly 4 AMG-CG iterations per refinement, it is not surprising to see the weak scaling parallel efficiency remains above 50% from $np = 1, ..., 4,096$.

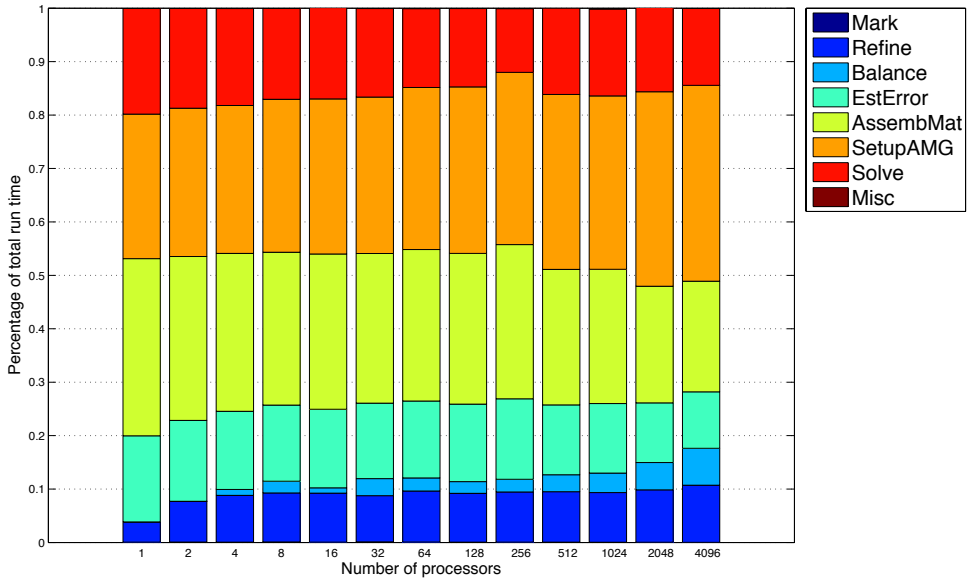


Figure 4.14: Weak scalability for steep gradients: breakdown of total run time into different components related to numerical PDE functions (green, yellow, orange, and red) and ALR functions (light and dark blue). Problem size increases at roughly 15,000 biquadratic elements per processor (at finest refinement level). The most expensive ALR operation is RefineMesh, which takes up to 10% of the runtime. Overall, ALR takes less than 20% of the overall run time.

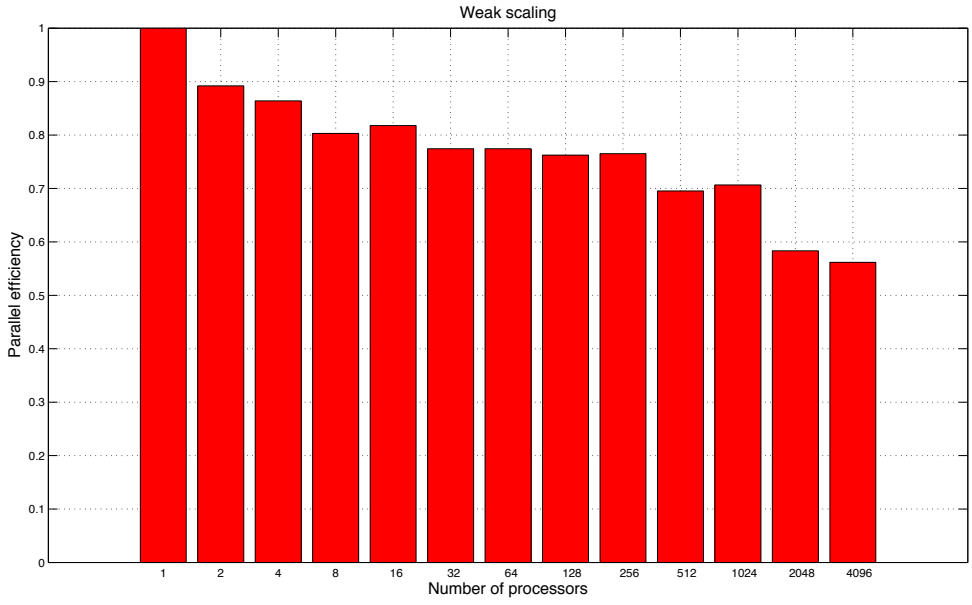


Figure 4.15: Weak scalability for steep gradients: parallel efficiency measured in total work units per processor per total run time, normalized by the total work units per total run time for a single processor, with number of processors from 1 to 4,096. Parallel efficiency remains above 50%.

4.4.3 Stokes Equation on the Backward Facing Step

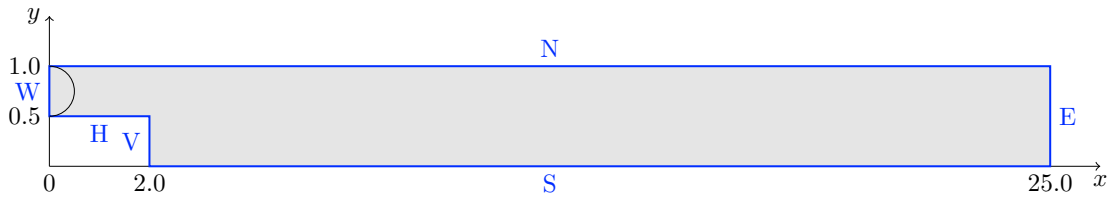


Figure 4.16: Domain for backward facing step.

Compressible Stokes equations in a backward facing step domain, see figure 4.16, is the second problem on which we test pACE implementation in pFOSPACK. As a system of PDEs, it is more challenging for discretization and linear solvers. Particularly, the stokes flow in the backward facing step domain is singular at the corner, which implies that the local error reduction near the singularity does not follow the FOSLS approximation heuristics. Possible remedies include applying a weighting function to the equations where singularities appear [29], or/and employing graded mesh refined towards the corner [22, 26]. Here, we use a weighting function to un-weight the corner in order to test the following:

- If the FOSLS approximation heuristics did not hold, would the NI-Newton-FOSLS-AMG-pACE algorithm still be efficient and effective?
- Does pACE result in near uniform refinement on finer levels? If not, how would that affect the parallel performance?

Let $\mathbf{u} = (u_1, u_2)^T$, where u and v are the velocities in the x- and y-direction and let p be the pressure. The Stokes equation is given by

$$\begin{aligned} -\Delta \mathbf{u} + \nabla p &= 0 \qquad \text{in } \Omega, \\ \nabla \cdot \mathbf{u} &= 0 \qquad \text{in } \Omega, \end{aligned} \tag{4.10}$$

with boundary conditions

$$\begin{aligned} u_1 &= 38.4(1-y)(y-0.5) && \text{on W,} \\ u_1 &= 0 && \text{on N,S,H,V,} \\ u_2 &= 0 && \text{on W,N,E,S,V,H,} \\ p &= 0 && \text{on E.} \end{aligned} \tag{4.11}$$

Define $\mathbf{U} = (U_{i,j})_{i,j=1,2} = (\nabla u_1, \nabla u_2)$, the gradient of the velocity. The velocity-velocity-gradient formulation of Stokes equations is given by

$$\begin{aligned} \mathbf{U} - \nabla \mathbf{u}^t &= 0 && \text{in } \Omega, \\ -(\nabla \cdot \mathbf{U})^t + \nabla p &= 0 && \text{in } \Omega, \\ \nabla \cdot \mathbf{u} &= 0 && \text{in } \Omega, \\ \nabla \times \mathbf{U} &= 0 && \text{in } \Omega. \end{aligned} \tag{4.12}$$

Write $U_{11} = \partial_x u_1$, $U_{12} = \partial_y u_1$, $U_{21} = \partial_x u_2$, and $U_{22} = \partial_y u_2$. Notice that $\nabla \cdot u = 0$ implies that $U_{11} + U_{22} = 0$, thus U_{22} can be eliminated in (4.12) to get

$$\mathbf{U} = \begin{pmatrix} U_{11} & U_{12} \\ U_{21} & -U_{11} \end{pmatrix}. \tag{4.13}$$

The corner induces singularity in the analytic solutions. We have $(\mathbf{U}, \mathbf{u}, p) \in (H^\sigma)^3 \times (H^{1+\sigma})^2 \times H^\sigma$, where $\sigma \approx 0.52$. This leads to the derivatives of p and $U_{i,j}$ not in L^2, and the L^2 FOSLS functional does not converge. A weighted FOSLS formulation is used here; see [29] for details. We have

$$\begin{aligned} \mathcal{G}_w(\mathbf{U}, \mathbf{u}, p; f) = ||\mathbf{U} - \nabla \mathbf{u}^t||_0^2 + ||w(-(\nabla \cdot \mathbf{U})^t + \nabla p)||_0^2 + ... \\ ||\nabla \cdot \mathbf{u}||_0^2 + ||w \nabla \times \mathbf{U}||_0^2, \end{aligned} \tag{4.14}$$

where w is the weighting function. Let r be the distance from the corner, choose

$$w = \begin{cases} \left(\frac{r}{.125}\right)^\alpha, & r < 0.125, \\ 1 & \text{otherwise.} \end{cases} \tag{4.15}$$

We use $\alpha = \frac{5}{2}$ for our test; hopefully, that can recover the quadratic convergence for biquadratic finite element space. The Heuristics is that wp and $wU_{i,j}$ lies in the weighted-Sobolev space H^{σ}_{α}. That would give $O(h^{\alpha+\sigma-1}) \approx O(h^{2.2})$ order convergence rate for the weighted-FOSLS functional.

Remark. The way in which we construct the weighting function follows [29]. However, theory there can be applied in a straight forward manner to the first-stage, weighted FOSLS functional

$$\mathcal{G}^1_w(\mathbf{U}, p; f) = ||w(-(\nabla \cdot \mathbf{U})^t + \nabla p)||^2_0 + ||w\nabla \times \mathbf{U}||^2_0. \tag{4.16}$$

Since we don't weight the other two terms, it is likely the convergence rate will be dominated by the smoothness of $\mathbf{u}$, which would yield $O(h^{2\sigma}) = O(h^{1.04})$. Numerical tests in the next section show the weighted-FOSLS functional converges like $O(h^{2.8})$, which is not the optimal convergence for quadratic elements, but obviously better than $O(h^{1.04})$. In addition, when the ALR grid is sufficiently fine, slow convergence rate, close to $O(h)$, of FOSLS functional is observed, which confirms the theoretic lower bounds. Figuring out the correct weights can significantly improve the accuracy, but that is beyond this thesis.

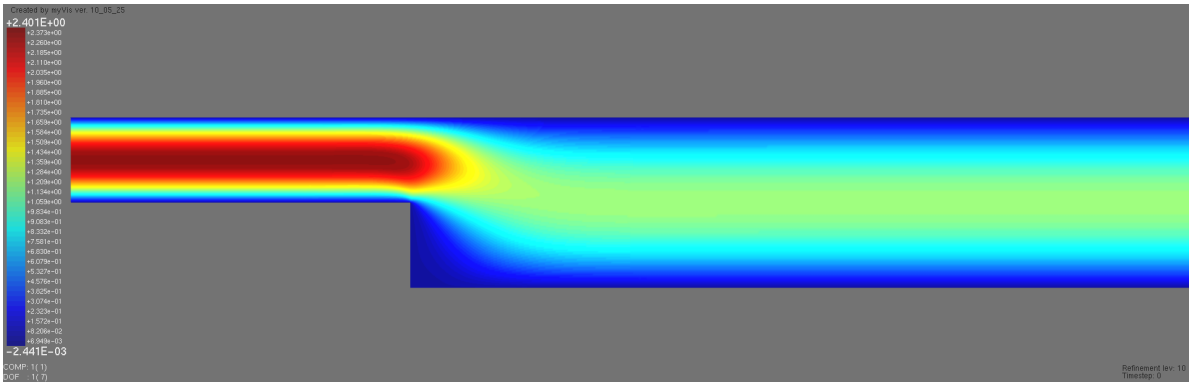
Remark. Computed velocity $\mathbf{u}$ and streamlines, ϕ, of the fluid velocity field are given in figure 4.17. It is computed by an auxiliary solve:

$$\nabla^{\perp}\phi = \mathbf{u}^h. \tag{4.17}$$

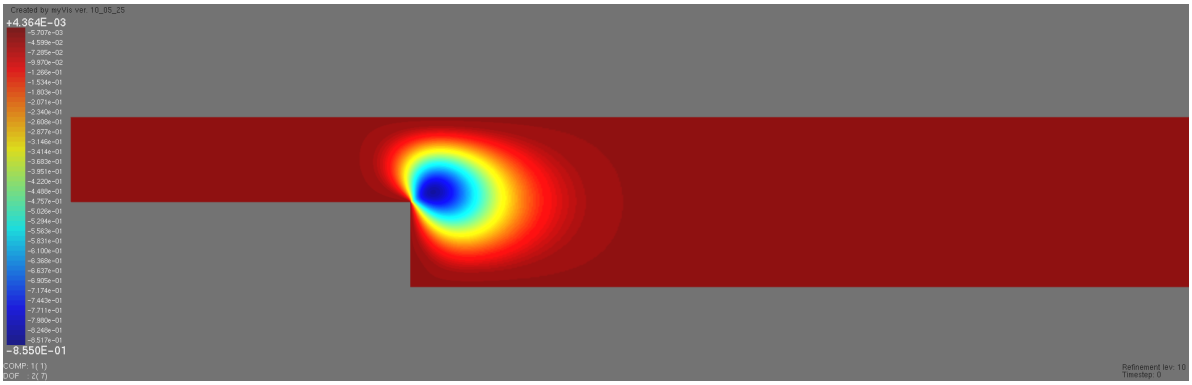
Boundary conditions of ϕ are computed by taking an integral along the inflow boundary, W.

4.4.3.1 Numerical Performance

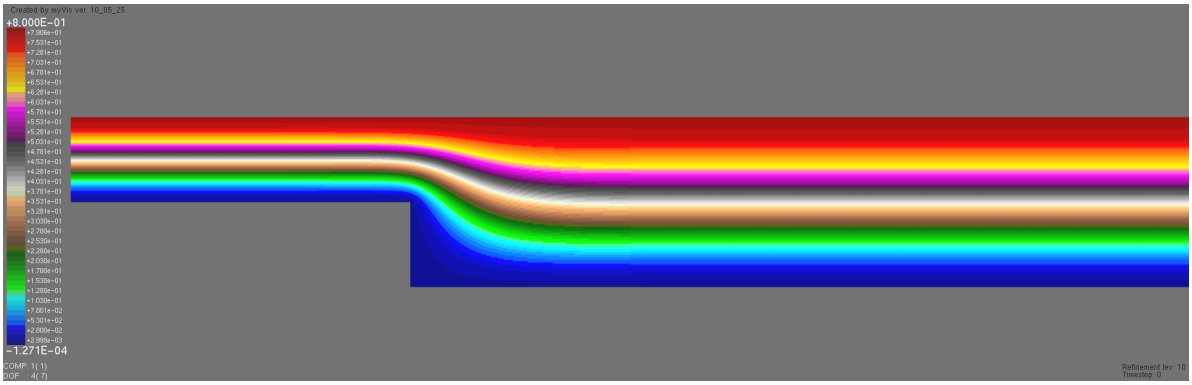
Local functional distribution and grid alignments are given in figure 4.18. The region near the corner is highly refined. Interestingly, another corner at the intersection of vertical boundary V and south boundary S gets refined several times. Another observation is that large numerical error occurs on the vertical boundary and the horizontal boundary, where the first derivative of the weighting function is discontinuous. This might result in large numerical error at fine grids. All of these aspects lead to modifications to the weighting function, and lead to violations to the



(a) X-velocity



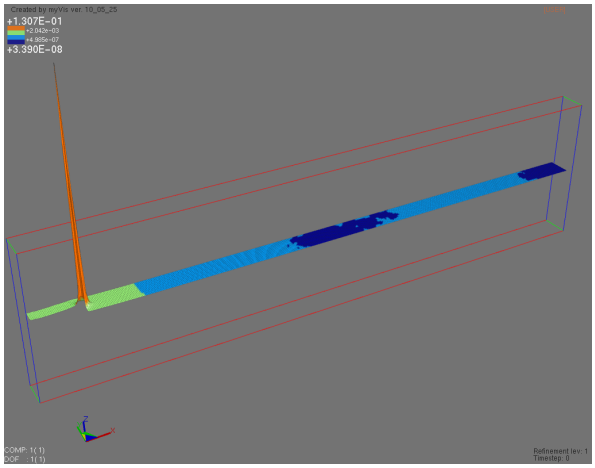
(b) Y-velocity



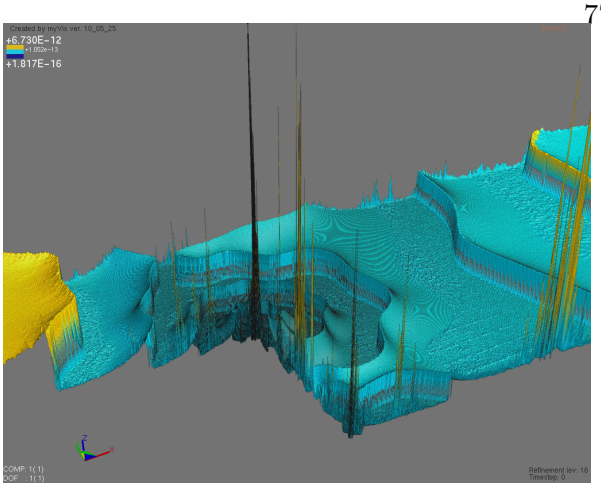
(c) Streamline

Figure 4.17: Computed solution of backward facing step Stokes.

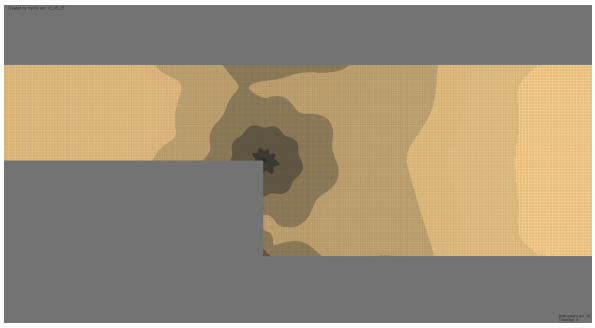
FOSLS approximation heuristics. That explains, in Table 4.4, why refinements behave as we expect before level 10: it refines everywhere when grid is too coarse, then concentrates on elements that contain large error. Once local error is equally distributed, refinement tends to uniform refinement. However, after refinement level 10, where there are 0.26 million biquadratic elements, additional



(a) Functional distribution at level 1



(b) Functional distribution at level 16



(c) Locally refined mesh at the finest level

(d) Grid alignment and partition near corner singularity

Figure 4.18: Stokes: locally refined mesh and functional distribution

features, due to flaws in the discretization, are found. Refinement becomes concentred on the features again and the slower convergence of the functional after level 10 also indicates heuristics for pACE do not hold. pACE might not be optimal at that point. However, results in Table 4.4 show that the NI-FOSLS-AMG-pACE approach is quite robust. It does not tend to over solve at each grid from level 12 to level 16. To verify this, we tried another test with fixed 350 AMG-CG iterations from level 13 and 16. Numerical error at each grid is just slightly improved. The finest level contains 2.66 million elements with FOSLS functional 3.71×10^{-7}. However, it costs more than 6 times work to reduce the finest grid error by roughly 2%.

Lastly, we compare the pACE algorithm to uniform refinement in Fig. 4.19. Because of the existence of corner singularity, it is not surprising that pACE is able use nearly 1% number of elements to reach the same accuracy as uniform refinement. Functional versus work units and CPU

ℓ	r_1	r_2	$E(r_1)$	$E(r_2)$	N_ℓ	$\mathcal{G}_\ell$	$ncyc$	σ	ρ
1	1.00	0.000	1.00	0.00	6,144	9.99e-01	51	1.55	0.94
2	0.03	0.000	0.79	0.00	6,633	2.03e-01	41	1.55	0.96
3	0.01	0.000	0.88	0.00	6,849	3.86e-02	42	1.55	0.94
4	0.03	0.000	0.86	0.00	7,434	9.75e-03	14	1.55	0.81
6	0.30	0.034	1.00	0.75	17,190	2.96e-04	33	1.56	0.90
8	0.92	0.006	1.00	0.42	65,991	1.52e-05	81	1.56	0.96
10	1.00	0.005	1.00	0.31	266,841	1.95e-06	41	1.57	0.95
12	0.01	0.005	0.63	0.63	287,433	1.77e-06	20	1.55	0.98
13	0.18	0.000	0.79	0.00	439,965	9.61e-07	51	1.54	0.98
14	0.33	0.000	0.88	0.00	867,555	5.69e-07	51	1.55	0.98
15	0.19	0.000	0.91	0.00	1,370,745	4.34e-07	51	1.52	0.98
16	0.34	0.000	0.94	0.00	2,752,761	3.78e-07	53	1.50	0.98

Table 4.4: NI-FOSLS-AMG-pACE for Stokes, relative setup cost $C_s \approx 31.7$, setup 213.76 WU, solve 338.22WU, overall runtime 271.36 sec. Here σ is the AMG grid complexity.

Method	Setup Cost	$ncyc$	Solve Cost	Total Work
NI-FOSLS-AMG-pACE	214	53	338	552
FOSLS-AMG	95	193	579	674

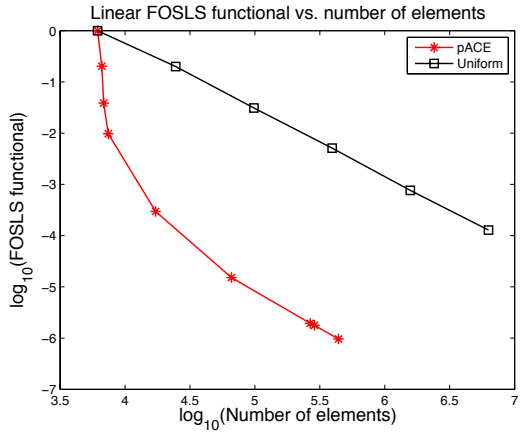
Table 4.5: Stokes: comparison of NI-FOSLS-AMG-pACE to applying FOSLS-AMG with random initial guess to the finest grid.

time also demonstrate the great improvement of using pACE over uniform refinement.

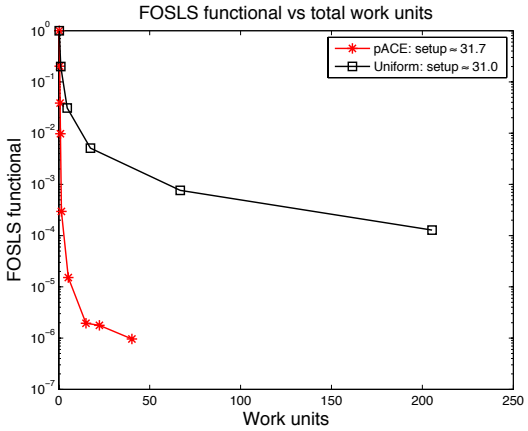
4.4.3.2 Parallel Performance Study

Three problems of different sizes are tested for strong scaling speedup. Results are similar to steep gradients tests in section 4.4.2.2. The smallest problem has speedup 90 in 128 processors, which is $90/16 \approx 5.65$ speedup over the runtime in 16 processors. Compared to the idea speedup, $128/16 = 8$, this represent about 70% of the ideal speedup. The speedup for the largest problem size, 2.75 million elements, is close to $3000/512 \approx 6.0$, compared to the ideal speed up 8, which is about 75% of the ideal speedup. These results are excellent when we consider the fact that the presence of the singularity require more load balancing on the finer grids.

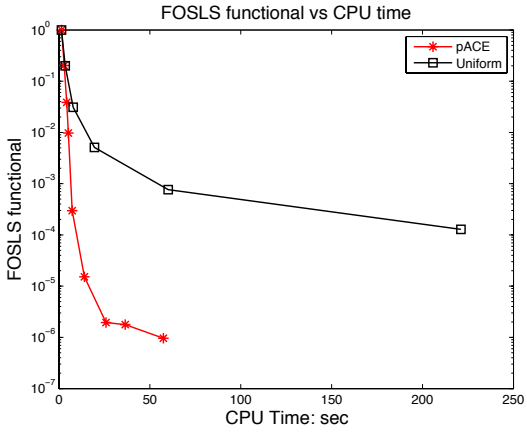
Figure 4.21 and 4.22 provide data with which to evaluate weak scaling. Like the test in section 4.4.2.2, figure 4.21 breaks overall run time into two categories: numerical PDE solve (error estimate, matrix assembling, linear solver setup, and AMG-CG solve) and ALR functions (marking, refining



(a) Functional vs. Number of elements



(b) Functional vs Work units



(c) Functional vs CPU time

Figure 4.19: Stokes: comparison between ALR and uniform refinement running on 1,024 processors. Work unit is equivalent to one matrix-vector multiplication on the finest grid of uniform refinement.

and load balancing). It is important to point out that, for the Stokes equations, the majority of overall runtime is numerical PDE functions. ALR operations take less than 10% of the overall run time. This results in weak scaling parallel efficiency that is above 70% for $np = 1, 2, ..., 1,024$, 60% for $np = 2,048$, and above 50% for $np = 4,096$.

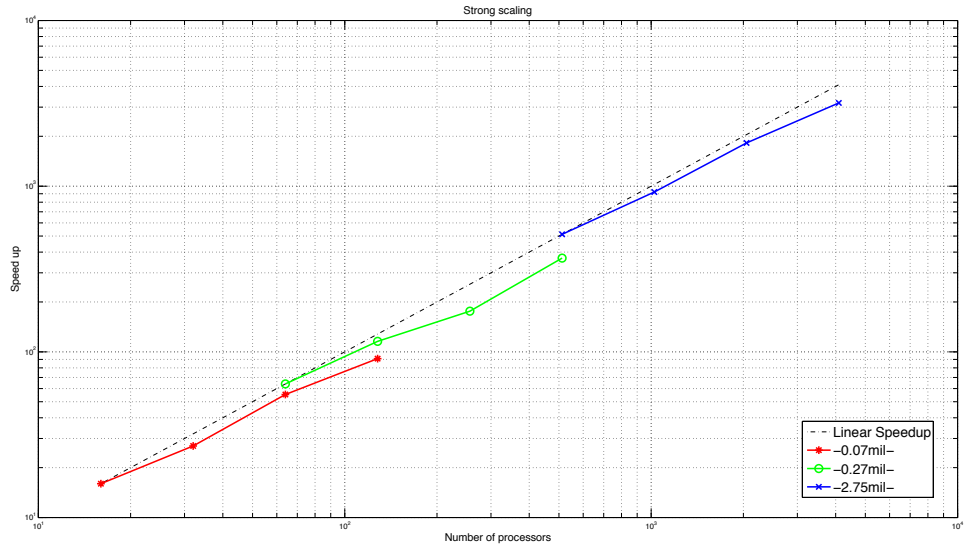


Figure 4.20: Strong scalability for Stokes: speedups based on total runtime versus the number of processors for three different problem sizes. The largest problem has 2.75 million biquadratic elements, which is roughly $5,400$ elements per processor.

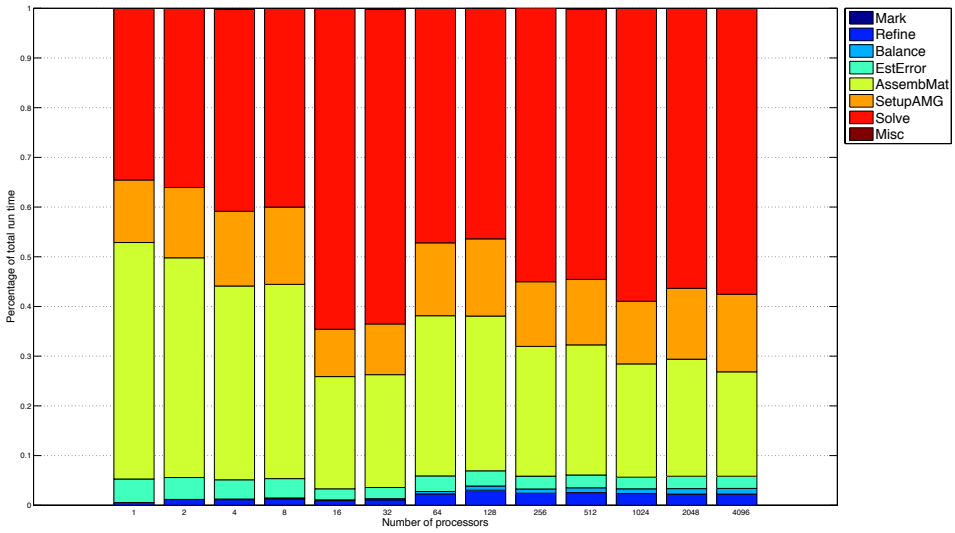


Figure 4.21: Weak scalability for Stokes: breakdown of total run time into different components related to numerical PDE functions (green, yellow, orange, and red) and ALR functions (light and dark blue). Problem size increases at roughly 6,000 biquadratic elements per processor (at finest refinement level). ALR functions takes less than 10% of overall time.

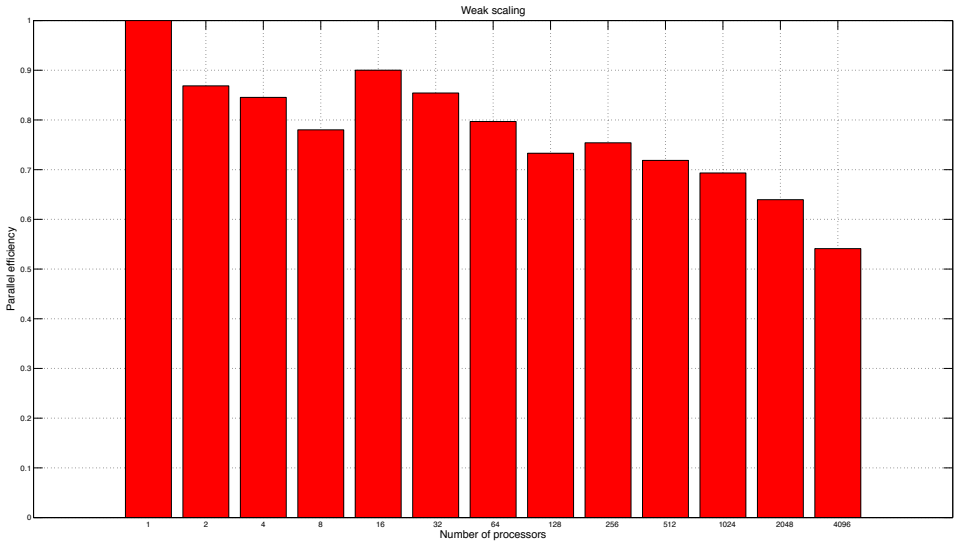
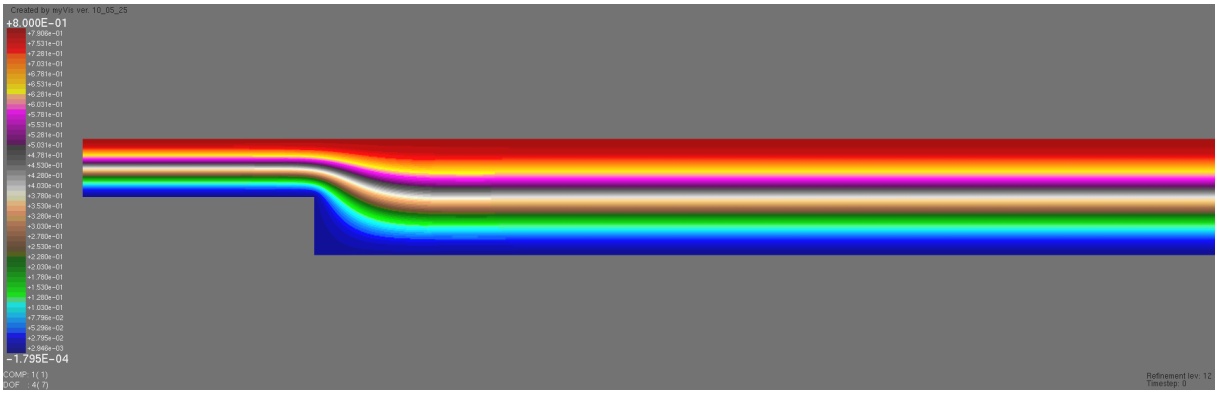
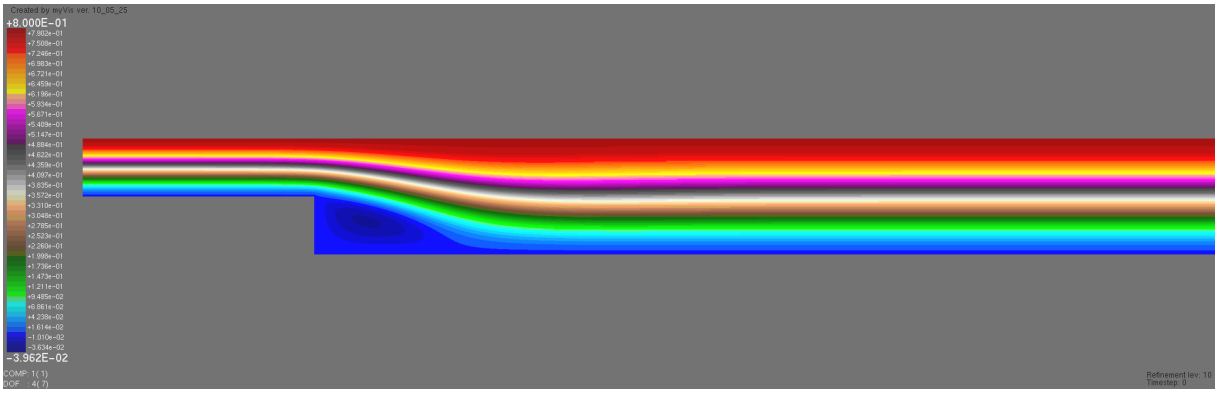


Figure 4.22: Weak scalability for Stokes: parallel efficiency measured in total work units per processor per total run time, normalized by the total work units per total run time for a single processor. With increasing number of processors from 1 to 4,096, parallel efficiency remains above 50%.

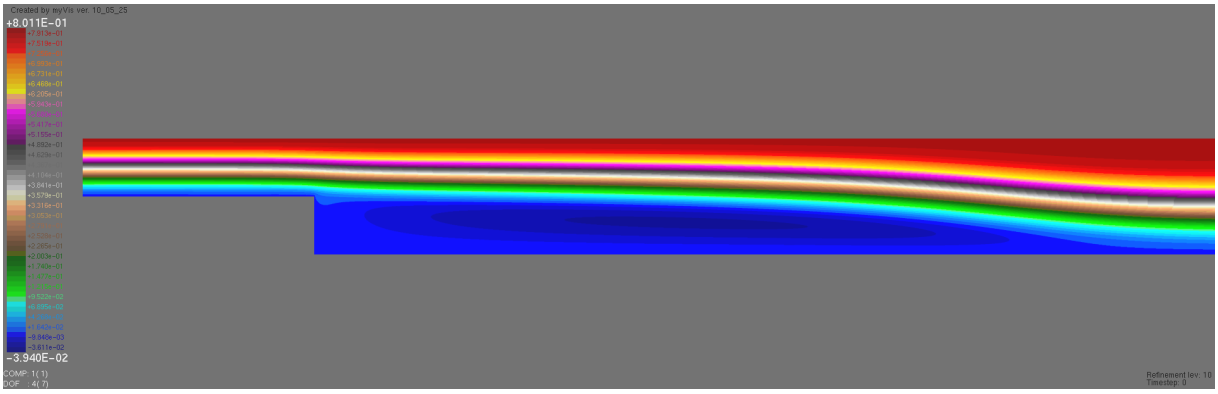
4.4.4 Navier Stokes Equation on the Backward Facing Step



(a) Streamline, $Re = 1.0$



(b) Streamline, $Re = 50.0$



(c) Streamline, $Re = 500.0$

Figure 4.23: Recirculation zones with respect to different Reynolds numbers.

This section discusses the numerical approximation and parallel performance of pACE applied

to the steady-state incompressible Navier-Stokes equations:

$$\begin{aligned} -\frac{1}{Re}\Delta\mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u} + \nabla p &= \mathbf{f} \qquad \text{in } \Omega, \\ \nabla\cdot\mathbf{u} &= 0 \qquad \text{in } \Omega, \end{aligned} \tag{4.18}$$

where Ω is the backward facing step domain described in the previous section. Here Re denotes the Reynolds number. To simplify our test, we choose $\mathbf{f} = \mathbf{0}$.

Similar to the Stokes equations, define the velocity-gradient tensor, $\mathbf{U} = (U_{i,j})_{2\times 2} = \nabla\mathbf{u}^t$ to get the velocity velocity-gradient pressure formulation of the Navier-Stokes equations:

$$\begin{aligned} \mathbf{U} - \nabla\mathbf{u}^t &= 0 \qquad \text{in } \Omega, \\ -\frac{1}{Re}(\nabla\cdot\mathbf{U})^t + \mathbf{U}^t\mathbf{u} + \nabla p &= \mathbf{0} \qquad \text{in } \Omega, \\ \nabla\cdot\mathbf{u} &= 0 \qquad \text{in } \Omega, \\ \frac{2}{Re}\nabla\times\mathbf{U} &= 0 \qquad \text{in } \Omega, \\ \nabla tr(\mathbf{U}) &= 0 \qquad \text{in } \Omega. \end{aligned} \tag{4.19}$$

The factor $\frac{2}{Re}$ appears here to improve AMG performance. We satisfy the trace free term exactly by eliminating U_{22}, i.e., let

$$\mathbf{U} = \begin{pmatrix} U_{11} & U_{12} \\ U_{21} & -U_{11} \end{pmatrix}. \tag{4.20}$$

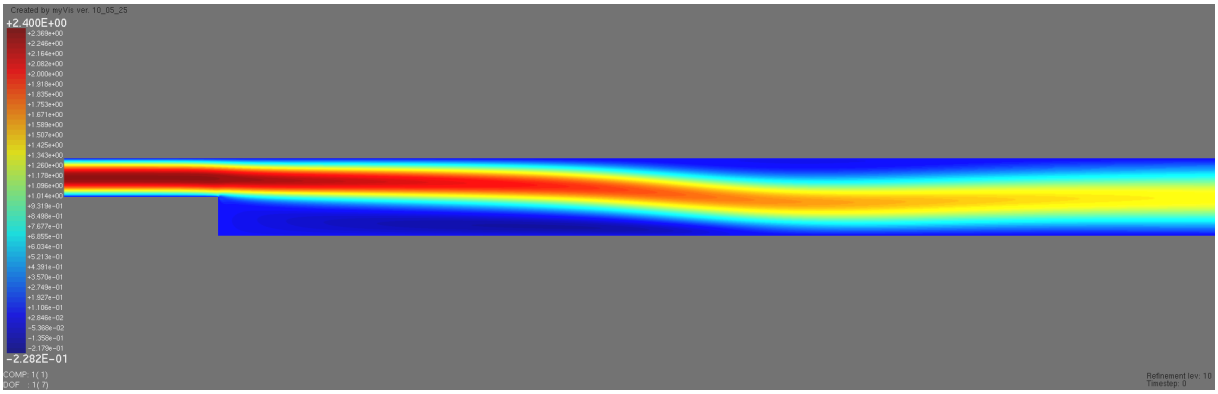
Then, the trace-free equation can be ignored. Applying the Newton-FOSLS approach to (4.19) at current iterate $(\mathbf{U}_n, \mathbf{u}_n, p_n)$ yields the linearized system

$$\begin{aligned} \delta\mathbf{U}_n - \nabla\delta\mathbf{u}_n^t &= -\mathbf{U}_n + \nabla\mathbf{u}_n^t && \text{in } \Omega, \\ -\frac{1}{Re}(\nabla\cdot\delta\mathbf{U}_n)^t + \delta\mathbf{U}_n^t\mathbf{u}_n + \mathbf{U}_n^t\delta\mathbf{u}_n + \nabla\delta p_n &= \frac{1}{Re}(\nabla\cdot\mathbf{U})_n^t - \mathbf{U}_n^t\mathbf{u}_n - \nabla p_n && \text{in } \Omega, \\ \nabla\cdot\delta\mathbf{u}_n &= -\nabla\cdot\mathbf{u}_n && \text{in } \Omega, \\ \frac{2}{Re}\nabla\times\delta\mathbf{U}_n &= -\frac{2}{Re}\nabla\times\mathbf{U}_n && \text{in } \Omega. \end{aligned} \tag{4.21}$$

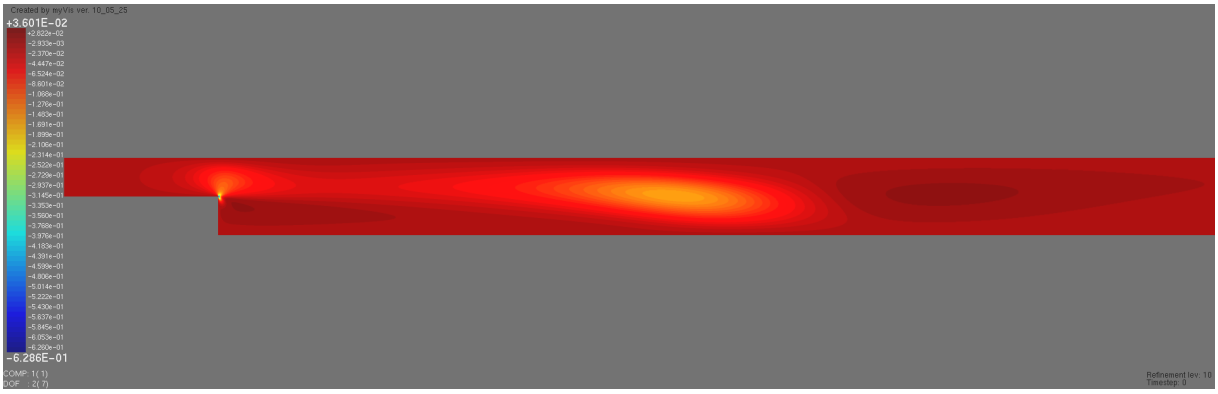
The same weighting function, w, used for solving the Stokes equations is applied to equations (2) and (4) in (4.21). The Navier-Stokes equations are more difficult to solve than the Stokes equations because of the nonlinearity. The nonlinear term becomes more dominant as the Reynolds

number increases, which brings more challenges to discretizations and linear solvers. For example, recirculating flow is formed by a backward facing step. The recirculation zone spreads out with increasing Reynolds number in a certain range, as it is illustrated in figure 4.23, where streamlines of the fluid velocity field are given for Reynolds number $Re = 1, 50, 500$, respectively. Numerical results will show not only the great power of super computers, but also, actually more importantly, good mathematics, which are required for accurately resolving features of the flow.

4.4.4.1 Numerical Performance



(a) X-velocity

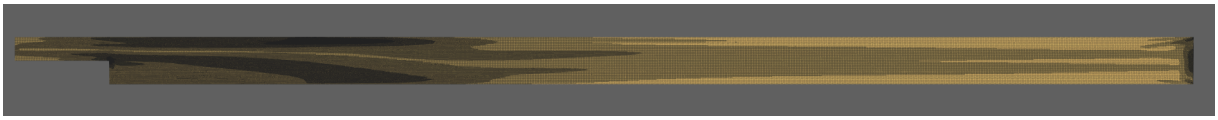


(b) Y-velocity

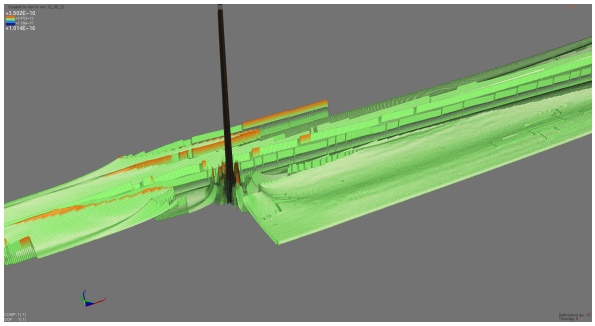
Figure 4.24: Computed solution of backward facing step Navier-Stokes, $Re = 500.0$.

We take Reynolds number $Re = 500$ for most of the tests in this section. The computed velocity $\mathbf{u}^h = (u_1^h, u_2^h)$ are depicted in figure 4.24. Roughly 1.8 million biquadratic elements exist on the finest grid, and are distributed nearly equally to 1,024 processors. Refinement patterns

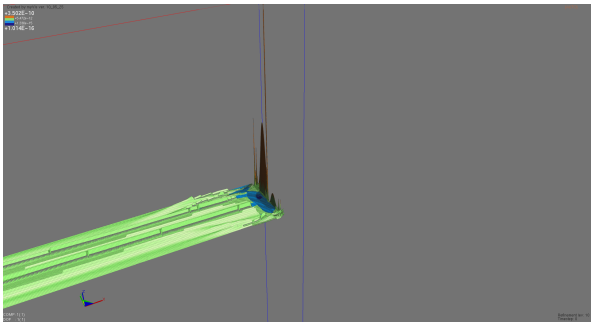
in figure 4.25 show that most refinements are made near the corner, where the singularity is located. Refinements also follow the flow downstream and occurs near outflow boundary. Careful examination of the refinement process reveals that elements close to outflow boundary only appear when grid is sufficiently fine. In our parallel weak scaling tests, such refinements never happena on ALR grid with number of elements less 0.6 million, (or with FOSLS functional greater than 10^{-5}). We suspect this is due to inaccurate boundary conditions along the outflow boundary. To be more specific, boundary conditions $v = 0.0$ at $x = 25.0$ is only an approximation. Despite the length of the tube, with Reynolds number $Re = 500$, the flow is not yet fully developed, while the boundary conditions at the outflow assume fully developed flow. This causes some inaccuracy near the outflow, and hence, some additional refinement when the grid is very fine.



(a) Locally refined grid at the finest level



(b) Functional distribution near corner singularity



(c) Functional distribution near outflow boundary

(d) Grid alignment near corner singularity

(e) Grid alignment near outflow boundary

Figure 4.25: Navier-Stokes: locally refined mesh and functional distribution

ℓ	r_1	r_2	$E(r_1)$	$E(r_2)$	N_ℓ	$\mathcal{G}_\ell$	$Newton$	$ncyc$	σ	ρ
1	1.00	0.00	1.00	0.00	6,144	1.07e-02	5	438	1.17	0.96
2	0.04	0.00	0.71	0.00	6,918	2.94e-03	2	236	1.17	0.97
3	0.16	0.00	0.72	0.00	10,305	8.22e-04	3	341	1.18	0.98
4	0.29	0.00	0.79	0.00	19,314	2.78e-04	1	71	1.22	0.96
5	0.78	0.00	0.95	0.00	64,434	1.00e-04	2	205	1.37	0.98
6	0.57	0.00	0.89	0.00	175,437	3.43e-05	3	371	1.44	0.98
7	0.03	0.00	0.41	0.00	191,424	2.78e-05	1	14	1.45	0.88
8	1.00	0.00	1.00	0.00	765,696	7.08e-06	3	340	1.51	0.98
9	0.03	0.00	0.45	0.00	832,839	6.45e-06	1	13	1.51	0.92
10	0.42	0.00	0.82	0.00	1,877,244	5.92e-06	1	36	1.52	0.95

Table 4.6: NI-Newton-FOSLS-AMG-pACE for Navier-Stokes, relative setup cost $C_s \approx 33.7$, setup 320.41 WU, solve 682.69WU, overall runtime 263.54 sec.

Method	Setup Cost	$Newton$	$ncyc$	Solve Cost	Total Work
NI-Newton-FOSLS-AMG-pACE	320	1	36	682	1002
Newton-FOSLS-AMG	153	6	503	1509	1662

Table 4.7: Navier-Stokes: comparison of NI-Newton-FOSLS-AMG-pACE to applying Newton-FOSLS-AMG with zero initial guess to the finest grid. Setup cost for Newton-FOSLS-AMG takes into account matrix assembling and AMG setup at each Newton step.

Various values at each refinement level are tabulated in Table 4.6. The number of Newton steps performed verifies the theory developed in [20] that only one Newton step is needed at finest level since solutions from previous grid provide "a very good initial" guess. In fact, it is so good that only one Newton step is required to bring the error to within discretization error on that grid. The efficiency and effectiveness of NI-Newton-FOSLS-AMG-pACE can be best demonstrated by comparing the total cost of the NI-Newton-AMG-pACE to the cost of assuming the optimal grid is known and applying Newton-FOSLS-AMG to the final grid. The choice of initial guess for the Newton-FOSLS-AMG method applied directly to the finest grid is important. Our tests show that with random initial guess normally, the Newton iteration normally fails to converge because the random initial guess lies outside of the attraction basin of the Newton-FOSLS method. Alternatively, a zero initial guess leads to a first Newton step that is equivalent to solving Stokes equations in the same domain. This provides a better initial guess. However, as shown in Table 4.7, zero initial guess results in 6 Newton steps to converge. The entire solve requires 503 AMG-CG

iterations and 6 times of matrix assembling and AMG setup. Compared to that, the NI-Newton-FOSLS-AMG-pACE method only takes one Newton step on the finest grid (with 36 AMG-CG iterations) to converge to the desired solution. Although thet NI-Newton-FOSLS-AMG-pACE method takes more Newton steps on the coarser levels, but solve on the coarser levels is relatively cheaper than on the finer levels. As a result, the overall cost of the NI-Newton-FOSLS-AMG-pACE method compared to the cost of applying Newton-FOSLS-AMG directly to the finest grid is about a factor of 60%. That is 40% saving.

To finish our discussion of the NI-Newton-FOSLS-pACE algorithm, we compare pACE to uniform refinement. Table 4.8 and Table 4.6 indicate that pACE is able to reach a functional value of 3.43×10^{-5} with 0.17 million elements. Compared to that, uniform refinement takes 1.57 million to have the same accuracy, which is about 9 times as large as with pACE. When considering CPU time, pACE is also the winer. It takes half of the CPU time of uniform refinement, but can drop the functional one order of magnitude lower.

ℓ	N_ℓ	$\mathcal{G}_\ell$	$Newton$	$ncyc$	σ	ρ
1	6144	1.07e-02	5	438	1.17	0.96
2	24,576	1.40e-03	4	578	1.18	0.98
3	98,304	2.49e-04	2	189	1.40	0.97
4	393,216	8.60e-05	1	135	1.47	0.98
5	1,572,864	3.80e-05	2	473	1.52	0.98

Table 4.8: NI-Newton-FOSLS-Uniform, overall runtime 525.54 sec.

4.4.4.2 Parallel Performance Study

Strong scalability results are presented for three different test problems. Speedups are excellent, in particular for the largest problem (see figure 4.26). For weak scaling, figure 4.27 demonstrates that the cost of the ALR functions for the Navier-Stokes problem is almost negligible relative to numeric PDE solves. One important thingin this figure is the light blue listed as "EstError". That includes every time the linear, nonlinear, and difference functional is computed after each Newton step to decide whether more Newton steps are needed. Computing and storing local error

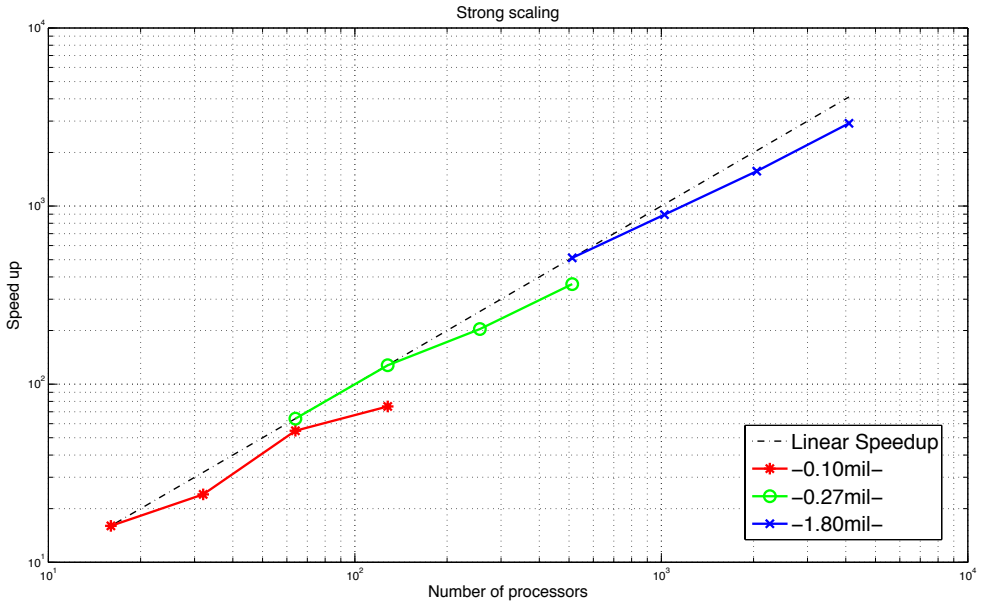


Figure 4.26: Strong scalability for Navier-Stokes: speedups based on total runtime versus the number of processors for three different problem sizes. The largest problem has 1.80 million biquadratic elements, which is roughly $3,600$ elements per processor.

indicators is done only once per level. Figure 4.28 demonstrates the excellent weak scaling parallel efficiency. Parallel efficiency remains above 60% up to $np = 4,096$. This is, again, not surprising, since load balancing is less significant than numerical PDE solves for Navier Stokes.

4.5 Conclusions

Parallel efficiency-based adaptive refinement algorithms are proposed and are applied to a 2D Poisson equation with steep gradients, 2D backward facing step Stokes equations, and nonlinear, incompressible, backward facing step Navier-Stokes equations. Numerical results show that if FOSLS approximation heuristics hold, then the NI-Newton-FOSLS-AMG-pACE approach results in equal distribution of local error and near uniform refinement on finer levels. That greatly reduces load balancing cost. Using Frost, the CU/NCAR BlueGene L, we demonstrate excellent strong and weak scalability up to $4,096$ processors of problem size 15 million biquadratic elements.

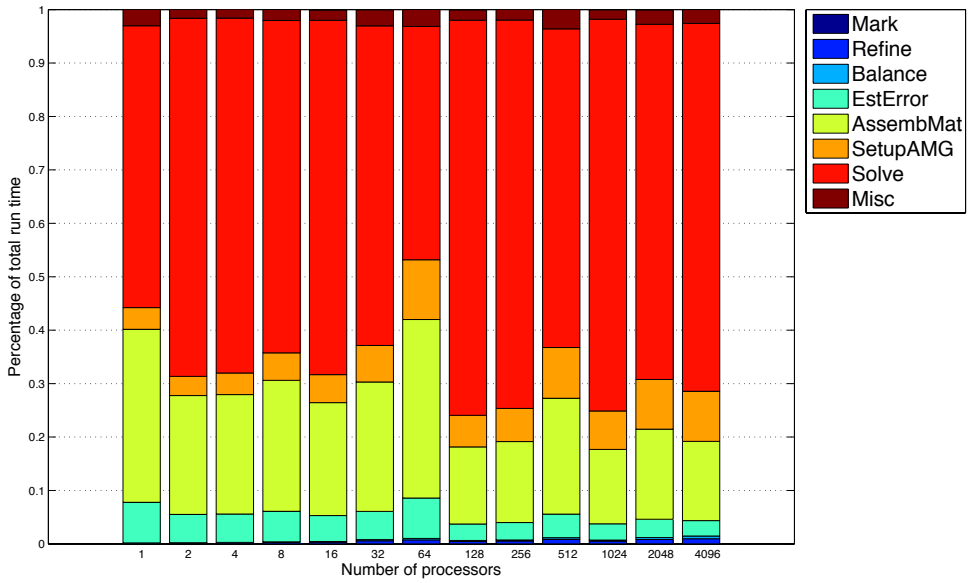


Figure 4.27: Weak scalability for Navier-Stokes: breakdown of total run time into different components related to numerical PDE functions (green, yellow, orange, and red) and ALR functions (light and dark blue). Problem size increases at roughly 6,000 biquadratic elements per processor (at finest refinement level). ALR functions takes less than 2% of overall time.

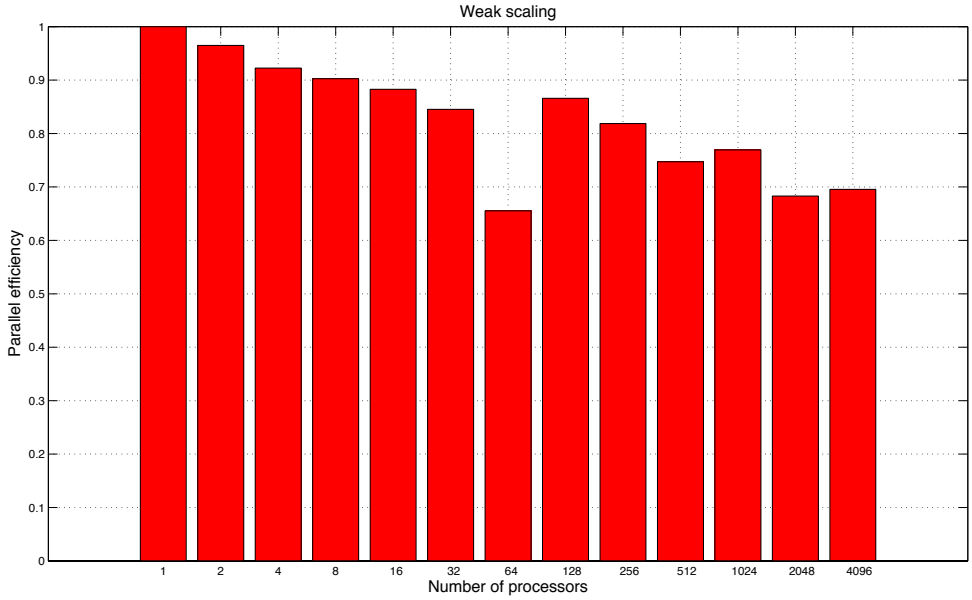


Figure 4.28: Weak scalability for Navier-Stokes: parallel efficiency measured in total work units per processor per total run time, normalized by the total work units per total run time for a single processor. With increasing number of processors from 1 to 4,096, parallel efficiency remains above 60%.

Chapter 5

Discussion

5.1 Concluding Remarks

In this thesis, efficiency-based refinement algorithms for the FOSLS finite element method with algebraic multigrid solvers in the context of nested iteration (NI-FOSLS-AMG) are developed. The algorithms choose which elements to refine based on optimizing computational efficiency, taking into account both error reduction and computational cost. Two efficiency measures are considered: predicted 'accuracy-per-computational-cost' (ACE) and the new 'anticipated-overall-computational-cost' (NACE). The use of the FOSLS local functional as a sharp a posteriori error estimate along with NI-AMG methods allows parameters to be computed that are used to estimate the current measures. In addition, several "flavors" of these efficiency-based schemes are tested to determine whether adding certain constraints to the efficiency measure, such as the total number of elements to add or the total amount of error to be reduced, would make it easier to obtain a near optimal grid. Numerical tests show that all of the efficiency-based algorithms effectively and efficiently capture local features of the solution. For the linear test problem, all schemes perform equally well, suggesting that the standard ACE scheme is sufficient without any extra constraints. For the more complicated nonlinear time-dependent MHD problem, this also is the case. In fact, the constrained schemes appear to at times perform unnecessary work, making them less optimal. However, all schemes greatly reduce the amount of computational cost for solving these problems to a specified accuracy compared to the cost of uniform refinement. In addition, in comparing the ACE scheme to threshold-based schemes, ACE either outperformed the threshold schemes or was

no worse than the best threshold-based method at any given time step. As the optimal refinement strategy varies over time steps, choosing a scheme such as ACE, which can adaptively choose the optimal refinement strategy, is preferable in the case in which many time steps are needed and the physics can change dramatically.

The modifications of the ACE-like ALR algorithms designed for parallel computers employ binning strategies that group elements into bins based on the local errors. Then, each bin is treated as an abstract element in the efficiency-measure formula. This is equivalent to using piecewise linear interpolation to approximate local functional distribution function. Refinement decisions are made according to bins by optimizing computational efficiency. Several binning strategies are developed. Equal size binning creates bins such that each bin has roughly the same number of elements, which, in particular, fits for parallel ACE with a constraint on the DOF. The geometric binning strategy groups elements according to the FOSLS approximation heuristics on local functional reduction, tends to provide a better approximation to the functional distribution, and, thus, usually produces better results. Tests demonstrate that the parallel ACE ALR algorithm based on geometric binning keeps the nice numerical properties of their serial counterparts and is capable of greatly reducing communication. Load balancing starts at very coarser grids. Elements and nodes are redistributed using a space filling curve and parallel tree structures at each refinement level. The SFC preserves locality of each grid partition and, thus, reduces communication cost. Numerical tests demonstrate that the NI-FOSLS-AMG-pACE approach tends to equally distribute local errors on finer levels, near uniform refinements are used. Together with load balancing performed on coarser levels, load balancing is no longer required. Tests on Frost demonstrate the numerical accuracy, effectiveness and efficiency of this approach. Weak and strong scaling tests show excellent parallel scalability and efficiency up to $4,096$ processors with problem size 25 million biquadratic elements.

5.2 Future Works

Several aspects still need to be studied. In this work, a generic AMG solver was used. Deterioration in the AMG convergence for increased timestep as well as Reynolds and Lundquist

numbers are observed in the MHD test and Navier-Stokes test. Even a slight improvement in the AMG algorithm would greatly reduce the total work units required to achieve a specified accuracy. AMG algorithms specifically designed for systems of PDEs are a topic of current research. This might involve the use of newly developed adaptive multigrid algorithms described more in [11, 12]. In addition, the hierarchy of the grids resulting from adaptive refinement might be used to reduce or eliminate the set up phase of AMG at each level, which normally takes a large fraction of the overall run time on each refinement level. A new multigrid solver might be developed for problems arising from adaptive refinement procedures. This would involve including more of the geometry or structure of the grids into the multilevel solver. The problems that would use such schemes, such as Stokes, Navier-Stokes, and MHD, are used in a variety of important applications, including fusion energy physics and space weather. It is to reasonable to tune the numerics for such specific problems.

Many aspects of the adaptive refinement algorithms can be improved. The FOSLS approximation heuristics introduced in chapter 3 require certain smoothness assumptions. When the solution contains singularities, for instance, one might want to adaptively determine the strength of the singularity and appropriately apply graded refinement techniques rather than splitting elements into subelements with equal size in each direction. The diffusion algorithm used to redistribute elements based on the SFC needs improvement. For example, load balancing is not necessary if load in only one or two out of thousands of processors is not balanced. Also pFOSPACK should allow for reading irregular meshes generated by third party software such as Cubit. This will allow pFOSPACK to test more realistic problems. Finally, time stepping should be added for parallel MHD tests. This will be explored in future research.

Bibliography

[1] R. A. Adams and J. J. F. Fournier. Sobolev Spaces. Academic Press, second edition, 2003.

[2] J. H. Adler. Nested Iteration and First-Order System Least-Squares for Incompressible Resistive Magnetohydrodynamics. PhD thesis, University of Colorado, Applied Mathematics Department, 2009.

[3] J. H. Adler, T. A. Manteuffel, S. F. McCormick, J. W. Nolting, J. W. Ruge, and L. Tang. Efficiency-based Adaptive Local Refinement for First-order System Least-squares Formulations. SIAM J. Sci. Comp. (SISC), to appear, 2010.

[4] J. H. Adler, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge. First-order system least squares for incompressible resistive Magnetohydrodynamics. SIAM J. Sci. Comp. (SISC), 32(1):229–248, 2010.

[5] J. H. Adler, T. A. Manteuffel, S. F. McCormick, J. W. Ruge, and G. D. Sanders. Nested Iteration and First-Order System Least Squares for Incompressible, Resistive Magnetohydrodynamics. SIAM J. Sci. Comp. (SISC), 32(3):1506–1526, 2010.

[6] R. E. Bank and M. J. Holst. A New Paradigm for Parallel Adaptive Meshing Algorithms. SIAM Review, 45:291–323, 2003.

[7] G. Bateman. MHD Instabilities. The MIT Press, 1978.

[8] M. Berndt, T. A. Manteuffel, and S. F. McCormick. Local Error Estimates and Adaptive Refinement for First-Order system Least Squares (FOSLS). E.T.N.A., 6:35–43, 1997.

[9] M. Berndt, T. A. Manteuffel, and S. F. McCormick. Analysis of First-order System Least Squares (FOSLS) for Elliptic Problems with Discontinuous Coefficients: Part II. SIAM J. Numer. Anal, 43:409–436, 2005.

[10] S. C. Brenner and L. R. Scott. The Mathematical Theory of Finite Element Methods. Springer, 2nd edition, 2002.

[11] M. Brezina, R. Falgout, S. Maclachlan, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge. Adaptive Smoothed Aggregation (aSA) Multigrid. SIAM Review (SIGEST), pages 317–346, 2005.

[12] M. Brezina, R. Falgout, S. Maclachlan, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge. Adaptive Algebraic Multigrid. SIAM J. Sci. Comp. (SISC), pages 1261–1286, 2006.

[13] F. Brezzi. On the Existentce, Uniqueness, and Approximation of Saddle-point Problems Arising from Lagrange Multipliters. RAIRO Anal. Numer., 8:129–151, 1974.

[14] W. L. Briggs, V. E. Henson, and S. F. McCormick. A Multigrid Tutorial. SIAM, Philadephia, PA, 2nd edition, 2000.

[15] C. Burstedde, O. Ghattas, M. Gurnis, G. Stadler, E. Tan, T. Tu, L. C. Wilcox, and S. Zhong. Scalable Adaptive Mantle Convection Simulation on Petascale Supercomputers. In Proceedings of ACM/IEEE SC '08, 2008.

[16] Z. Cai, R. Lazarov, T. A. Manteuffel, and S. F. McCormick. First-Order System Least Squares for Second-Order Partial Differential Equations. I. SIAM J. Numer. Anal., 31:1785–1799, 1994.

[17] Z. Cai, T. A. Manteulffel, and S. F. McCormick. First-Order System Least Squares for Second-Order Partial Differential Equations. II. SIAM J. Numer. Anal, 34:425–454, 1997.

[18] L. Chacon, D. A. Knoll, and J. M. Finn. An Implicit, Nonlinear Reduced Resistive MHD Solver. J. of Computational Physics, 178:15–36, 2002.

[19] L. Chacon, D. A. Knoll, and J. M. Finn. Nonlinear Study of the Curvature-Driven Parallel Velocity Shear-Tearing Instability. Physics of Plasmas, 9:1164–1176, 2002.

[20] A. Codd, T. A. Manteuffel, and S. F. McCormick. Multilevel First-Order System Least-Squares for Nonlinear Partial Differential Equations. SIAM J. Numer. Anal., 41:2197–2209, 2003.

[21] P. Colella, J. Bell, N. Keen, L. Ligocki, M. Lijewski, and B. van Straalen. Performance and Scaling of Locally-structured Grid Methods for Partial Differential Equations. Journal of Physics: Conference Series, 78:1–13, 2007.

[22] H. De Sterck, T. A. Manteuffel, S. F. McCormick, J. W. Nolting, J. W. Ruge, and L. Tang. Efficiency-Based h- and hp-refinement Strategies for Finite Element Methods. Num. Lin. Alg. Appl., 15:89–114, 2008.

[23] L. F. Diachin, R. Hornung, P. Plassmann, and A. M. Wissink. Parallel Adaptive Mesh Refinement. In Parallel Processing for Scientific Computing. SIAM, 2006.

[24] W. Dörfler. A Convergent Adaptive Algorithm for Poisson's Equation. SIAM J. Numer. Anal., 33:1106–1124, 1996.

[25] R.D. Falgout and U.M. Yang. hypre: a Library of High Performance Preconditioners. In Computational Science - ICCS 2002 Part III, volume 2331, pages 632–641. Springer-Verlag, 2002.

[26] W. Gui and I. Babuška. The h, p, and hp Versions of the Finite Element Method in 1 Dimension, Parts I, II, III. Numerische Mathematik, 49:577–683, 1986.

[27] V.E. Henson and U.M. Yang. BoomerAMG a Parallel Algebraic Multigrid Solver and Preconditioner. Applied Numerical Mathematics, 41:155–177, 2002.

[28] D. A. Knoll and L. Chacon. Coalescence of Magnetic Islands, Sloshing, and the Pressure Problem. Physics of Plasmas, 13(3):032307, 2006.

[29] E. Lee, T. A. Manteuffel, and C. R. Westphal. Weighted-norm First-Order System Least-Squares (FOSLS) for Problems with Corner Singularitties. SIAM J. Numer. Anal, 44:1974–1996, 2006.

[30] P. MacNeice, K. M. Olson, R. de Fainchtein C. Mobarry, and C. Packer. PARAMESH: A Parallel Adaptive Mesh Refinement Community Toolkit. Computer Physics Communications, 126:330–354, 2000.

[31] T. A. Manteuffel, S. F. McCormick, J. W. Nolting, J. W. Ruge, and G. D. Sanders. Further Results on Error Estimators for Local Refinement with First-Order System Least Squares (FOSLS). Num. Lin. Alg. Appl., 17(2-3):387–413, 2010.

[32] P. Morin, R. H. Nochetto, and K. G. Siebert. Data Oscillation and Convergence of Adaptive FEM. SIAM J. Numer. Anal., 38:466–488, 2000.

[33] P. Morin, R. H. Nochetto, and K. G. Siebert. Convergence of Adaptive Finite Element Methods. SIAM Review, 44:631–658, 2002.

[34] Josh Nolting. Efficiency-based Local Adaptive Refinement for FOSLS Finite Elements. PhD thesis, University of Colorado, Applied Mathematics Department, 2008.

[35] C. D. Norton, G. Lyzenga, J Parker, and R. E. Tisdale. Developing Parallel GeoFESTP using the PYRAMID AMR Library. NASA Jet Propulsion Laboratory, Tech. Rep., 2004.

[36] B. Philip, L. Chacon, and M. Pernice. Implicit Adaptive Mesh Refinement for 2D Reduced Resistive Magnetohydrodynamics. J. Comp. Phys, 227(20):8855–8874, October 2008.

[37] Oliver Röhrle. Multilevel First-Order System Least-Squares for Quasilinear Elliptic Partial Differential Equations. PhD thesis, University of Colorado, Applied Mathematics Department, 2008.

[38] U. Rüede. Mathematical and Computational Techniques for Multilevel Adaptive Methods, volume 13 of Frontiers in Applied Mathematics. SIAM, Philadephia, PA, 1993.

[39] J. W. Ruge. Fospack users manual. Unpublished, v1.0 edition, 2000.

[40] H. R. Strauss. Nonlinear, Three-Dimensional Magnetohydrodynamics of Noncircular Tokamaks. Physics of Fluids, 19:134–140, 1976.

[41] H. Sundar, R. S. Sampath, and G. Biros. Bottom-Up Construction and 2:1 Balance Refinement of Linear Octrees in Parallel. SIAM J. Sci. Comp. (SISC), 30:2675–2708, 2008.

[42] T. Tu, D. O'Hallaron, and O. Ghattas. Scalable Parallel Octree Meshing for Terascale Applications. In Proceedings of ACM/IEEE SC '05, 2005.

[43] R. Verfürth. A Review of A Posteriori Error Estimation and Adaptive Mesh-Refinement Techniques. Wiley-Teubner, 1995.

[44] A. M. Wissink, D. Hysom, and R. D. Hornung. Enhancing Scalability of Parallel Structured AMR Calculation. In Proceedings of ICS '03, 2003.

[45] X. Zhang. Multilevel Schwarz Methods. Numer. Math, 63:521–539, 1992.