8-2009

Adaptive Discontinuous Galerkin Finite Element Methods

Haihang You
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Haihang You entitled "Adaptive Discontinuous Galerkin Finite Element Methods." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Jack Dongarra, Major Professor

We have read this dissertation and recommend its acceptance:

Ohannes Karakashian, James Plank, Michael Thomason, Shirley Moore

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Haihang You entitled “Adaptive Discontinuous Galerkin Finite Element Methods.” I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Jack Dongarra, Major Professor

We have read this dissertation
and recommend its acceptance:

Ohannes Karakashian

James Plank

Michael Thomason

Shirley Moore

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of Graduate School

(Original signatures are on file with official student records.)

ADAPTIVE DISCONTINUOUS GALERKIN FINITE ELEMENT METHODS

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Haihang You

August 2009

Dedication

To Albert and Robert, I always love you!

To my father You Chenan, mother Li Helin, brother Yuanhang, sister Liangyu.

To my uncle Li Hefeng for looking after me when I was away from home for the first time.

To the memory of uncle Li Hebiao, I miss you all the time.

Acknowledgments

I would like to express my immense gratitude to my advisors, Professor Jack Dongarra and Professor Ohannes Karakashian, for the inspiration, valuable comments and expert guidance throughout the process of research. My sincere gratitude to my Dissertation Committee, Dr. James Plank, Dr. Shirley Moore and Dr. Michael Thomason.

Abstract

The Discontinuous Galerkin Method is one variant of the Finite Element Methods for solving partial differential equations, which was first introduced by Reed and Hill in 1970's [27]. Discontinuous Galerkin Method (DGFEM) differs from the standard Galerkin FEM that continuity constraints are not imposed on the inter-element boundaries. It results in a solution which is composed of totally piecewise discontinuous functions. The absence of continuity constraints on the inter-element boundaries implies that DG method has a great deal of flexibility at the cost of increasing the number of degrees of freedom. This flexibility is the source of many but not all of the advantages of the DGFEM method over the Continuous Galerkin (CGFEM) method that uses spaces of continuous piecewise polynomial functions and other "less standard" methods such as nonconforming methods. As DGFEM method leads to bigger system to solve, theoretical and practical approaches to speed it up are our main focus in this dissertation. This research aims at designing and building an adaptive discontinuous Galerkin finite element method to solve partial differential equations with fast time for desired accuracy on modern architecture.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Discontinuous Galerkin Method, first introduced by Reed and Hill in the 1970's [27], is one variant of the Finite Element Methods for solving partial differential equations. The Discontinuous Galerkin Method (DGFEM) differs from the standard or continuous Galerkin FEM (SGFEM or CGFEM) in that continuity constraints are not imposed on the inter-element boundaries, resulting in a solution that is composed of totally piecewise discontinuous functions. The absence of continuity constraints on the inter-element boundaries implies that the DGFEM has a great deal of flexibility at the cost of increasing the number of degrees of freedom. This flexibility is the source of many but not all of the advantages of the DGFEM method over the CGFEM. The CGFEM uses spaces of continuous piecewise polynomial functions and other "less standard" methods such as nonconforming methods. Noncorforming methods are characterized by the imposition of continuity at certain points on the inter-element boundaries.

As the DGFEM method results in larger linear systems than the other versions, theoretical and practical approaches to speed it up become very important especially when one wishes to measure its competitiveness. Indeed, at the same time as theoretical and implementational aspects of the DGFEM are developed, one must undertake comparative studies with other Finite Elements methods especially in the area of efficiency. Indeed, a main theme of this research has been to successfully incorporate some aspects of the standard Finite Element Methods into the DGFEM, thus creating a hybrid and more efficient method.

The recently released TOP500 list [25] of the world's fastest supercomputers depicts some important trends in the area of high performance computing: clusters represent the most common architecture and multi-core processors represent the dominant chip architecture. These trends have a big influence on research and development in high performance computing. To achieve high performance on such systems, the software has to be 1) scalable on a distributed memory system with tens of thousands of CPUs, 2) capable of on-chip parallelism, which takes advantage of multi-core chip architecture with shared memory threading, and 3) tuned to have better cache locality and enhance instruction level parallelism. The Discontinuous Galerkin Method with an additive Schwartz preconditioner shows its full potential at all these different levels of parallel optimization. First of all, it is natural to split the whole problem into small pieces by domain decomposition for distributed computing. Such domain decomposition is relatively easy to carry out with the discontinuous scheme, since it does not require continuity along the boundary of elements. Secondly, each domain maintains a row of blocked dense matrices, i.e. a diagonal block (the stiffness matrix of the domain), and a list of non-diagonal blocks (flux and penalty terms along boundaries of domain), and a list of small blocked sparse matrices (stiffness matrices of subdomains) if using domain decomposition as preconditioner. The computational tasks for an iterative method for solving a linear system are a list of sparse matrix vector products, which can be further parallelized on a multi-core architecture. And at last, to achieve peak performance on modern systems with new techniques such as longer pipelines, deeper memory hierarchies, and hyper threading technologies, we have to generate highly optimized libraries for dense and sparse linear algebra kernels [14, 35], [7], [38]. Typical transformations include loop blocking [31, 37], loop unrolling [2], and loop permutation, fusion and distribution [5, 24]. One aspect of the adaptivity of our software is its ability to generate a computationally intense kernel as it is installed on one system. For example, solving huge sparse linear systems with dense blocks is an important part of the Galerkin method. One can speed up the sparse linear solver by generating a fast matrix vector multiplication function for the known matrix size.

To summarize, the contributions of our research are the following:

- We have revisited an old aspect of the standard Galerkin method concerning the assembly

of the stiffness matrix A. As an alternative to explicitly forming A, we have used instead a factorization of A of the form RDR^T and compared the relative efficiencies of performing matrix-vector multiplications with A versus RDR^T. While it may be counterintuitive to expect that replacing one such operation with three can be advantageous, it turns out that exploiting the structures of R and D results in a competitive algorithm that shows better scalability and performance on multicore systems. This result is not only interesting in the context of the standard Galerkin method itself, but also impacts the DGFEM since we also use the matrix A as a preconditioner in solving the linear systems resulting from the DGFEM.

- We have developed a nonoverlapping additive Schwartz domain decomposition preconditioning algorithm for the fast iterative solution of the linear systems for the DGFEM. The main theme here is to use the CGFEM to precondition the DGFEM. The benefits of this approach are twofold. First, this preconditioner (the matrix A above) is smaller than its discontinuous counterpart. Second, a difficulty caused by so-called penalty terms of the DGFEM are bypassed resulting in an improvement in the condition number of the preconditioned system. We include a new and rigorous analysis that shows that the condition number of the preconditioned system is the expected (and optimal) $O(H/h)$ where H and h are a measure of the coarse and fine meshes respectively.

- We have developed an adaptive finite element algorithm(ASER) based on a posteriori error estimates developed earlier. This algorithm implements the ideas and techniques outlined above and also implements a drastic cutting marking strategy resulting in a decrease in the number adaptive cycles needed to achieve the prescribed tolerance.

"If we can enhance computational efficiency the method may in the end outperform state-of-the art finite volume solvers, especially when DGFEM is combined with h- and p- adaptation. $\cdots$ We believe therefore that DGFEM has a huge potential as a next generation flow solver technology. " — CENAERO(Centre of Excellence in Aeronautical Research) and its CFD Multi-physics group, Swansea, UK, 4-6 April 2005.

Chapter 2

Continuous Galerkin Method

2.1 Introduction

In 1943, Richard Courant introduced the Finite Element Method [10] (FEM) for approximating solutions of partial differential equations. Since then FEM has been studied and developed to be a powerful and widely used method for numerical solutions of partial differential equations. In this chapter we revisit the Continuous Galerkin Method and exploit an approach for performing matrix-vector multiplications without actually forming the stiffness matrix.

2.2 Poisson's Equation

Let us consider the following second order elliptic problem for Poisson's equation:

$$-\Delta u = f \quad \text{in } \Omega, \tag{2.1}$$

$$u = g_D \quad \text{on } \Gamma_D, \tag{2.2}$$

$$\nabla u \cdot n = g_N \quad \text{on } \Gamma_N. \tag{2.3}$$

where $\Omega \subset \mathbf{R^d}$, $d = 2,3$, Γ_D denotes the Dirichlet boundary, Γ_N denotes the Neumann boundary, and $\partial\Omega = \Gamma_D \cup \Gamma_N$ with $\mathbf{n}$ being the unit outward normal vector to $\partial\Omega$.

Let $V = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D\}$ be the space of so-called *test functions*. We can obtain a

weak formulation of the PDE above by multiplying Eq(2.1) with $v \in V$ and integrating over Ω:

$$-\int_\Omega (\Delta u) v dx = \int_\Omega f v dx. \tag{2.4}$$

Integrating the left side by parts and using the fact that $v = 0$ on Γ_D and $\frac{\partial u}{\partial \mathbf{n}} = g_N$ on Γ_N, we have

$$-\int_\Omega (\Delta u) v dx = \int_\Omega \nabla u \cdot \nabla v dx - \int_{\Gamma_N} g_N v ds \tag{2.5}$$

Let u be the solution of Eq(2.1), then u is also the solution of following variational problem: Find $u \in H^1(\Omega)$ satisfying $u\big|_{\Gamma_D} = g_D$ such that

$$a(u,v) = F(v), \quad \forall v \in V, \tag{2.6}$$

where

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v dx \tag{2.7}$$

$$F(v) = \int_\Omega f v dx + \int_{\Gamma_N} g_N v ds. \tag{2.8}$$

There is an alternative way to calculate u which we shall use to formulate the standard Galerkin formulation. Assume that we have a function $g \in H^1(\Omega)$ that agrees with the Dirichlet data g_D on Γ_D. Letting $u = g + \tilde{u}$ with $\tilde{u} \in V$ and using it in (2.6), there follows

$$a(\tilde{u}, v) = F(v) - \int_\Omega \nabla g \cdot \nabla v dx.$$

The advantage of this formulation is to work with $\tilde{u}$ which is in the same space V as the test functions.

2.3 Finite Element Spaces

The Galerkin Finite Element Method consists in projecting the solution of a particular partial differential equation into a finite dimensional space of functions and using the weak formulation developed above. Typically, these spaces are constructed over a nonoverlapping partition of Ω. We usually call such a partition a *mesh* and note that it is composed of *cells*. These cells could be triangles, rectangles, tetrahedra or other shapes. The commonly used term *element* should be reserved to denote a specific association of a cell type with a family of function spaces. These functions consist of piecewise polynomial functions, i.e., the restriction of such functions into any given cell is a polynomial of a degree chosen by the user. This choice is motivated by the obvious ease of use of polynomial functions. Another important characteristic of the Finite Element Method is that the basis functions of these spaces have small supports, e.g. a small patch of elements. The intended effect is to obtain matrices that are sparse. Indeed, this is a feature that distinguishes the FEM (and also the Finite Difference Method) from spectral methods.

The construction of finite element function spaces starts with that of so-called *local basis functions* for the vector space $\mathcal{P}_q(K)$ of polynomials of total degree $q \geq 1$ defined on K. These bases are adapted to the type or shape of the particular elements or cells of the mesh. Since we will restrict ourselves to triangles in 2-d, we shall use the Lagrangian basis functions. Depending on the degree of polynomials used, these basis functions are naturally associated with certain points of the triangle, or degrees of freedom [9].

Example: Linear Lagrange Elements. Denote the three vertices of a triangle $K \in \mathcal{T}_h$ by v^0, v^1, v^2(see Figure 2.1). There exists a basis consisting of three affine functions $\phi_K^{(0)}, \phi_K^{(1)}, \phi_K^{(2)}$ in the variables x, y such that

$$\phi_K^{(i)}(v^j) = \delta_{ij}, \quad i, j = 0, 1, 2.$$

Example: Quadratic Lagrange Elements. In addition to the three vertices, let v^3, v^4, v^5 be the midpoints of sides v^0v^1, v^1v^2, v^2v^0 respectively (see Figure 2.2) There exists a basis consisting of six functions $\phi_K^{(j)},\ j = 0, \ldots, 5$ satisfying

$$\phi_K^{(i)}(v^j) = \delta_{ij},\ i, j = 0, \ldots, 5.$$

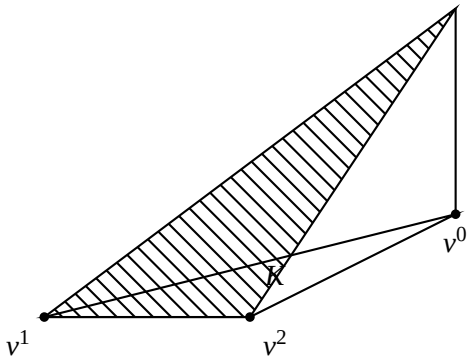
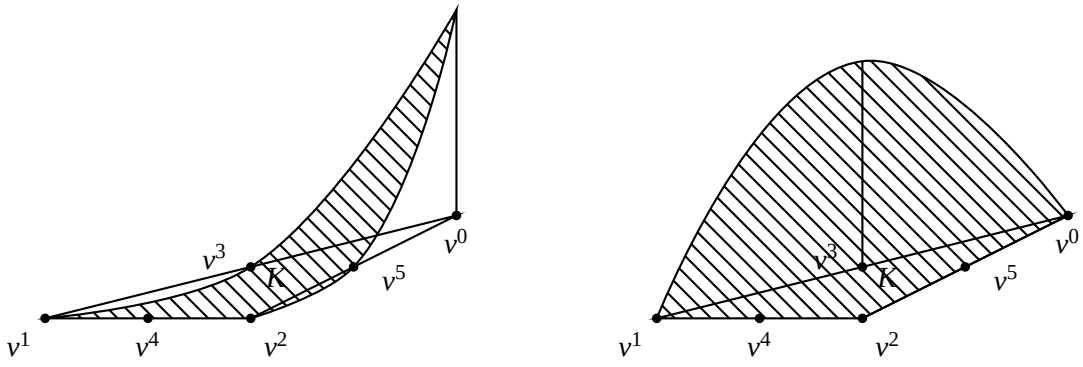


Figure 2.1: Linear basis function $\phi_K^{(0)}$



(a) $\phi_K^{(0)}$ (b) $\phi_K^{(3)}$

Figure 2.2: Quadratic basis functions $\phi_K^{(0)}$ and $\phi_K^{(3)}$

In general, the space $\mathcal{P}_q(K)$ of polynomials in (x,y) of total degree q has dimension $m_q = \frac{1}{2}(q+1)(q+2)$. Lagrangian basis functions for each can be constructed. As shown in the above examples, these basis functions are associated with the *local nodes* $x_K^{(j)},\ j = 0,\ldots,m_q-1$.

By extending these local functions by zero outside of K, we obtain functions that are defined on all of Ω).

2.4 Standard Galerkin Formulation

Let $\mathcal{T}_h = \{K_i : i = 1,2,...,m_h\}$ be a mesh on Ω such that $\Omega = \cup_{K\in\mathcal{T}_h} K = K_1 \cup K_2 \cup ... \cup K_{m_h}$. We assume that $\mathcal{T}_h$ is locally quasi-uniform and that each cell in it is starlike. (See section 3.1.3 for definitions).

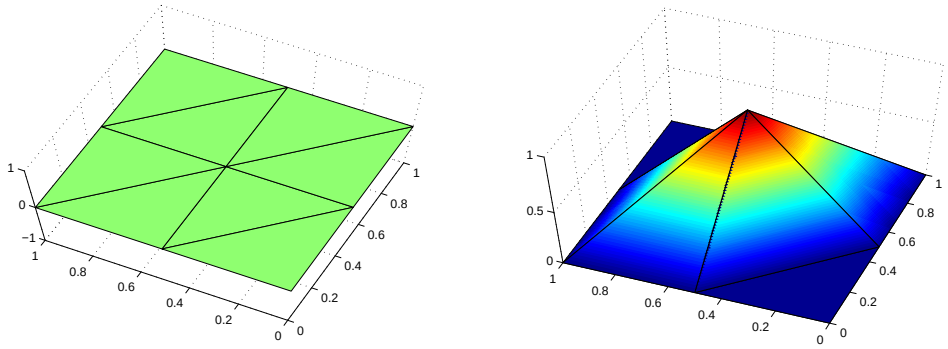
Using the local basis functions (local to each cell) $\phi_K^{(j)}$ introduced above, we construct a global space of continuous functions defined on $\mathcal{T}_h$. We let

$$S_h^q = \{v \mid v\big|_K \in P_q(K),\ K \in \mathcal{T}_h,\ v\big|_{\Gamma_D} = 0\}.$$

Basis functions for the spaces S_h^q are easy to construct on conforming meshes. Let us recall that a mesh is conforming if whenever two cells, say triangles, are adjacent, i.e. they share an edge, then this edge is a full edge for both triangles. We also say that a conforming mesh is characterized by the absence of *hanging nodes*. It is important to note that basis elements for the continuous spaces S_h^q are extremely difficult or even impossible to construct for general nonconforming meshes. Thus, whereas a standard Galerkin formulation can be defined on nonconforming meshes, the Galerkin approximation cannot be calculated on such meshes due to the unavailability of bases for S_h^q. One of the contributions of this work is to show a way to do this by constructing appropriate embedding and projection operators/matrices.

Assuming that the mesh is conforming, we let $\mathcal{N}_h$ denote the collection of all local degrees of freedom $x_K^{(j)},\ K \in \mathcal{T}_h$. More precisely,

$$\mathcal{N}_h = \{\nu \mid \nu = x_K^{(j)}, K \in \mathcal{T}_h,\ j = 0,\ldots,m_q-1,\ \text{ and } \nu \notin \Gamma_D\}.$$



(a) A triangulated 2D mesh. (b) Support of linear ϕ_j

Figure 2.3: Linear basis functions on a triangulated 2D mesh.

Note that a given element in $\mathcal{N}_h$ is identified with a set of local degrees of freedom. Also, note that we do not include in $\mathcal{N}_h$ nodes that belong to Γ_D since the Galerkin approximation is given by g_D on Γ_D and the test functions vanish there.

We define the global basis functions ϕ_ν for S_h^q as follows: With each $\nu \in \mathcal{N}_h$ we associate a basis function ϕ_ν satisfying

$$\phi_\nu(\mu) = \begin{cases} 1 & \text{if } \nu = \mu \\ 0 & \text{if } \nu \neq \mu \end{cases} \tag{2.9}$$

When basis functions are linear, Figure2.3(b) shows the support of ϕ_j consists of triangles that share the common node x_i that is at the center of the mesh in Figure 2.3(a).

Now we can formulate the Standard Galerkin Method for Eq(2.1) from the variational formulation Eq(2.6) as follows: Let $g_h \in H^1(\Omega)$ be a function which agrees with g_D at the nodes belonging to Γ_D. We seek $\tilde{u}_h \in S_h^q$ that satisfies

$$a(\tilde{u}_h, v_h) = F(v_h) - \int_\Omega \nabla g_h \cdot \nabla v_h, \forall v_h \in S_h^q. \tag{2.10}$$

Then the standard Galerkin approximation is given by $u_h = \tilde{u}_h + g_h$. Indeed, a function such as g_h can be easily constructed by interpolating the data g_D on the Dirichlet nodes. Furthermore, g_h is nonzero only on a thin layer adjacent to Γ_D.

This formulation can be recast as a linear system of equations that may be solved by a variety of methods, typically iterative when the number of unknowns is large, say more than a thousand.

We express the finite element solution u_h defined by (2.10) as a linear combination $u_h(\mathbf{x}) = \sum_{j=1}^{N} \xi_j \phi_j(\mathbf{x})$ of the basis functions of S_h^q. Using this expression in (2.10), we obtain the linear system in the unknown vector

$$A\xi = b \tag{2.11}$$

where

$$a_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, dx \tag{2.12}$$

$$b_i = F(\phi_i) - \int_\Omega \nabla g_h \cdot \nabla\phi_i \, dx, \tag{2.13}$$

A is called the **StiffnessMatrix** and is symmetric positive definite with each element defined in Eq(2.12).

2.5 The Stiffness Matrix: To assemble or not to assemble

If a direct solver is to be used to solve the system (2.11), then the stiffness matrix A must be explicitly formed. This is accomplished as follows. First, for each cell $K \in \mathcal{T}_h$, a $m_q \times m_q$ *local stiffness* matrix A_K is calculated according to

$$(A_K)_{ij} = \int_K \nabla\phi_K^{(j)} \cdot \nabla\phi_K^{(i)} \, dx$$

where $\phi_K^{(j)}$ are the basis functions local to K. Since the latter functions are polynomials on K, the integrals can be evaluated exactly using some quadrature rule of sufficient accuracy. These calculations can be performed efficiently on a so-called master or reference cell $\hat{K}$ using affine transformations between the cell K and the reference cell. (See Appendix A for more details).

Now noting that a global basis function ϕ_i is a sum of some local basis functions,

$$\phi_i = \sum_{\mathbf{x}_K^{(j)} = \mathbf{x}_i} \phi_K^{(j)},$$

```
Initialize A to zero.
for K ∈ T_h do
    for i = 0, ..., m_q − 1 do
        for j = 0, ..., m_q − 1 do
            if x_K^(i) ∉ Γ_D and x_K^(j) ∉ Γ_D then
                A_νμ = A_νμ + (A_K)_ij where ν = x_K^(i), μ = x_K^(j)
            end
        end
    end
end
```

Algorithm 1: Assembly of the stiffness matrix

A is calculated (assembled) from the local blocks A_K using Algorithm 1

When using an iterative method, especially one such as the Conjugate Gradient method that involves matrix-vector multiplications, one has the option of not assembling the matrix A. Rather, the matrix-vector multiplications are performed using the decomposition:

$$A = RDR^T \tag{2.14}$$

thus turning one matrix-vector multiplication with A into three matrix-vector multiplications. While this idea is known to practitioners of the FEM, no experimental study of the relative efficiencies of the two approaches exists to our knowledge. Later in this section we exhibit results of comparative numerical experiments using up-to-date optimization techniques. But first, we need to describe the matrices D and R. As far as the dimensions of these matrices are concerned, let

$$\begin{aligned} |\mathcal{T}_h| &= \text{number of cells(triangles) in } \mathcal{T}_h \\ m_q &= \text{number of DOF's per cell} \\ N &= \text{number of global DOF's} \end{aligned}$$

D is the $m_q|\mathcal{T}_h| \times m_q|\mathcal{T}_h|$ block diagonal matrix such that the $m_q \times m_q$ blocks along the diagonal are precisely the local stiffness matrices A_K. Indeed, D is A in unassembled form.

R^T is the $m_q|\mathcal{T}_h| \times N$ matrix which identifies a global node or DOF to the set of its local counterparts. To visualize R^T, we may think of it as $|\mathcal{T}_h|$ *slabs* R_K^T of m_q rows each corresponding to cell

K and where

$$(R_K^T)_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_K^{(i)} = \mathbf{x}_j \\ 0 & \text{otherwise.} \end{cases}$$

Even though some of its rows (corresponding to Dirichlet nodes) are zero, the matrix R^T has full column rank. Another interpretation of R^T is to view it as a change of basis matrix between the local bases and the global ones. Indeed, a given function in S_h^q can be expressed as

$$\sum_{K\in\mathcal{T}_h}\sum_{j=0}^{m_q-1}\alpha_K^{(j)}\phi_K^{(j)} \text{ and also as } \sum_{\nu=1}^{N}\beta_\nu\phi_\nu.$$

R^T satisfies

$$\alpha = R^T\beta.$$

Figure 2.4 is a mesh with $|\mathcal{T}_h| = 8$ triangles: K_0, K_1, $\cdots$, K_7, with Dirichlet boundary Γ_D and Neumann boundary Γ_N. There are $N = 6$ degree of freedoms denoted by boldface numbers. So A is a 6×6 matrix. For simplicity, the linear basis functions are used for this example. As each triangle's vertex is given a index of 0 to 2, $m_q = 3$ and the Dirichlet block matrix D^i is a 3×3 matrix. Each element of A is assembled from corresponding Dirichlet block matrices. For example,

$$a_{33} = D_{11}^1 + D_{11}^2 + D_{11}^3 + D_{00}^4 + D_{22}^6 + D_{00}^7 \tag{2.15}$$

A can be factorized by Eq(2.14) with matrices: R, D, and R^T. Matrix D is a 24×24 block diagonal matrix with 3×3 diagonal blocks. Matrix R is a 6×24 sparse matrix with only 0 and 1.

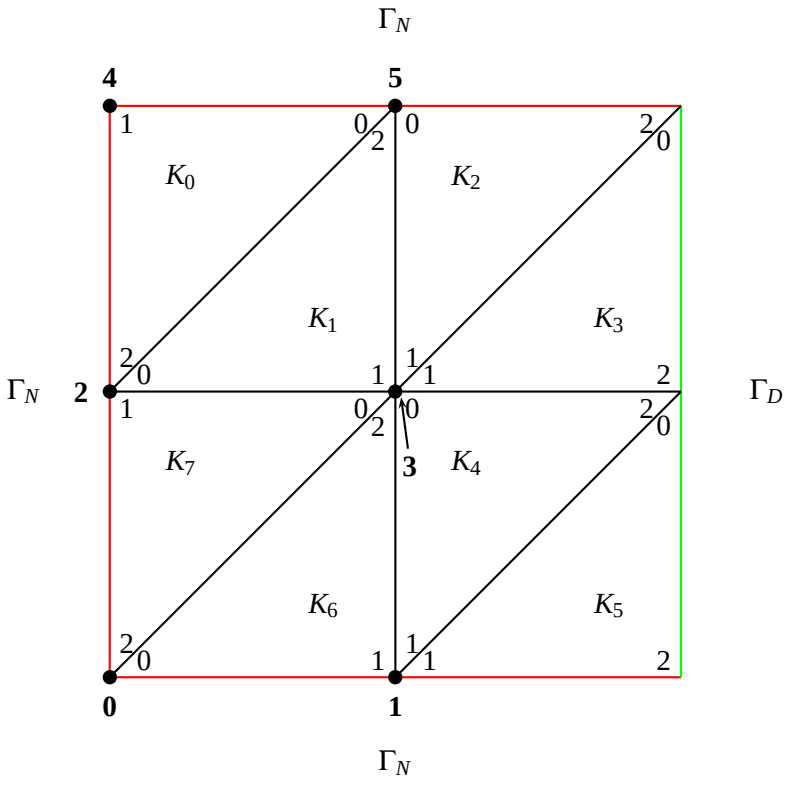


Figure 2.4: Mesh with indexing of vertices, triangles, and local vertex indexing of each triangle

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \tag{2.16}$$

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.17}$$

$$D = \begin{bmatrix} D^0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & D^1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & D^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & D^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D^4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & D^5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & D^6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^7 \end{bmatrix} \tag{2.18}$$

With the most commonly used sparse matrix storage format, compressed sparse row(CSR), the sparse matrix-vector multiplication usually runs at 10% or less of the machine's peak performance [34]. As researchers try to optimize the sparse matrix-vector multiplication by transform-

ing matrix storage format CSR to block compressed sparse row(BCSR), which allows optimization techniques such as unrolling and register-level tiling of each block matrix-vector multiplication [19], the performance improvement is limited by the random memory access pattern of sparse matrix multiplication. And this type of optimization sometimes is not practical as the transformation time is too long. With Eq(2.14) we turn a sparse matrix-vector multiplication into three matrix-vector multiplications. But R only contains 1's, $R \times x$ can be programed with additions, and $R^T \times x$ can be programed with data stores, and we also can save memory space by not storing the array of 1's. As D preserves the dense diagonal blocks, it has better data locality. We can apply optimization techniques such as unrolling, register-level blocking, vector operations, and multi-threading, which usually can not be applied easily to sparse matrix-vector multiplication due to indirect indexing. We write the optimized version of diagonal block matrix-vector multiplication written in assembly, and combine that with multi-threading on the RDR^T format. The preliminary results show encouraging speedup on Intel Quad core architectures. We believe that we can further improve the performance with more aggressive tuning efforts similar to ATLAS [35].

2.6 Experiment and Performance

The following is a testing problem:

$$\begin{aligned} -\Delta u &= 128\pi^2 sin(8\pi x)sin(8\pi y) \qquad \text{in } \Omega \\ u &= 0 \qquad\qquad\qquad\qquad\quad \text{on } \partial\Omega \end{aligned} \tag{2.19}$$

The solution of Problem(2.19) is: $u = sin(8\pi x)sin(8\pi y)$ as shown in Figure 2.5, which is a smooth non-polynomial solution and oscillatory across the domain. We choose an iterative method as our solver: conjugate gradient method [6], since the linear system is sparse symmetric positive-definite. This method is one of the best iterative method for solving a symmetric, positive-definite linear system. The pseudo code is given in Algorithm(2).

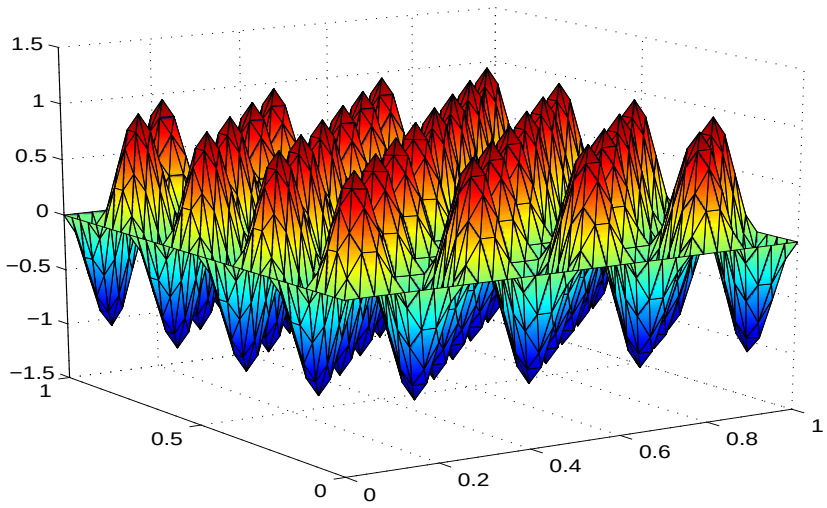


Figure 2.5: Solution of Problem(2.19) on a mesh with 4096 triangles.

Input: vector x_0 can be an approximate initial solution or 0

$r_0 = b - Ax_0$;
$p_0 = r_0$;
$k = 0$;
while *true* **do**
  $\alpha_k = \frac{r_k^\top r_k}{p_k^\top A p_k}$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $r_{k+1} = r_k - \alpha_k A p_k$;
  if $|r_{k+1}| \leq \varepsilon$ **then**
    exit
  end
  $\beta_k = \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$;
  $p_{k+1} = r_{k+1} + \beta_k p_k$;
  $k = k + 1$;
end
Output: x_{k+1}

Algorithm 2: Conjugate Gradient Method

Table 2.1: Machine Specifications

Feature	Intel Xeon (Quad core)
Processor Speed	2.4GHz
L1 Cache	8KB
L2 Cache	8MB
Number of Sockets	4
Number of Cores	16
OS	Linux
Compiler	icc 11.0

We set up the experiment as following:

- The experiments are conducted on an Intel Xeon Quad core architecture with specifications shown in Table 2.1.
- Initial triangulations using `triangle` were done using maximum area constraint $a = 0.1$, and the finest mesh has 1048576 triangles.
- The iterative solver terminates at accuracy of 10^{-16}.

We conduct experiments to compare the performance of CG method with three different matrix multiplications:

- Matric Vector multiplication (MV) with sparse matrix CSR storage format.
- Reference MV with RDR^T diagonal dense block format.
- Optimized MV with RDR^T diagonal dense block format, written in assembly with SSE2 vector operations.

Experimental results in Figure 2.6 and Figure 2.7 show the total time spent for solution with various number of threads enabled. Since most of SSE2 instructions require 16 byte alignment, we chose to implement block diagonal matrix vector multiplication for degree of 2 and 3 polynomial basis functions for now. The block sizes are 6 and 10 respectively, so as long as the starting pointer is 16-byte aligned, then all the matrix blocks that are consecutively allocated will be 16-byte aligned

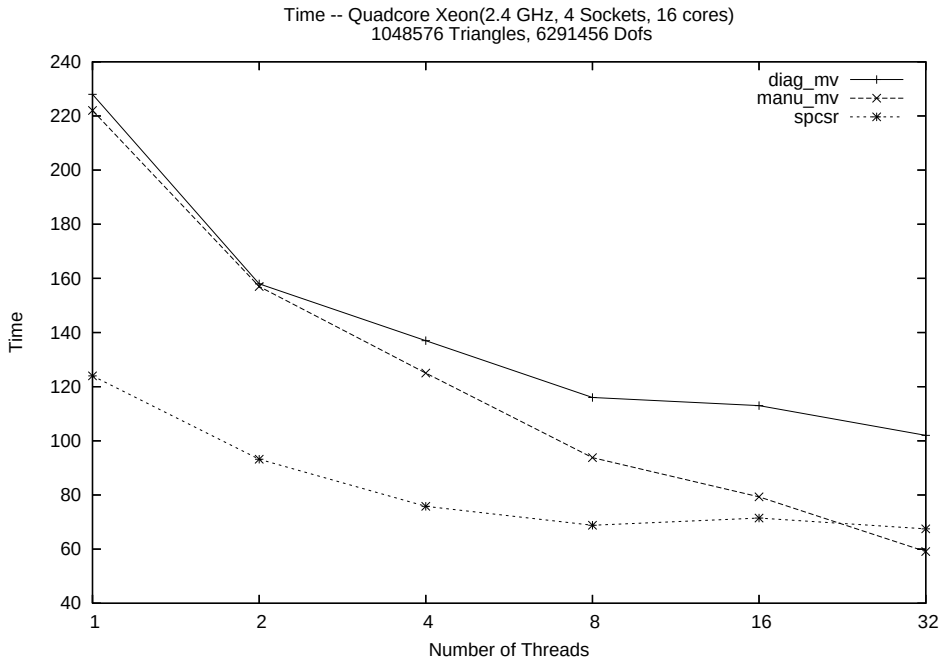


Figure 2.6: Comparison of computation time of CG method with CSR and RDR^T matrix storage format, degree of polynomial is 2

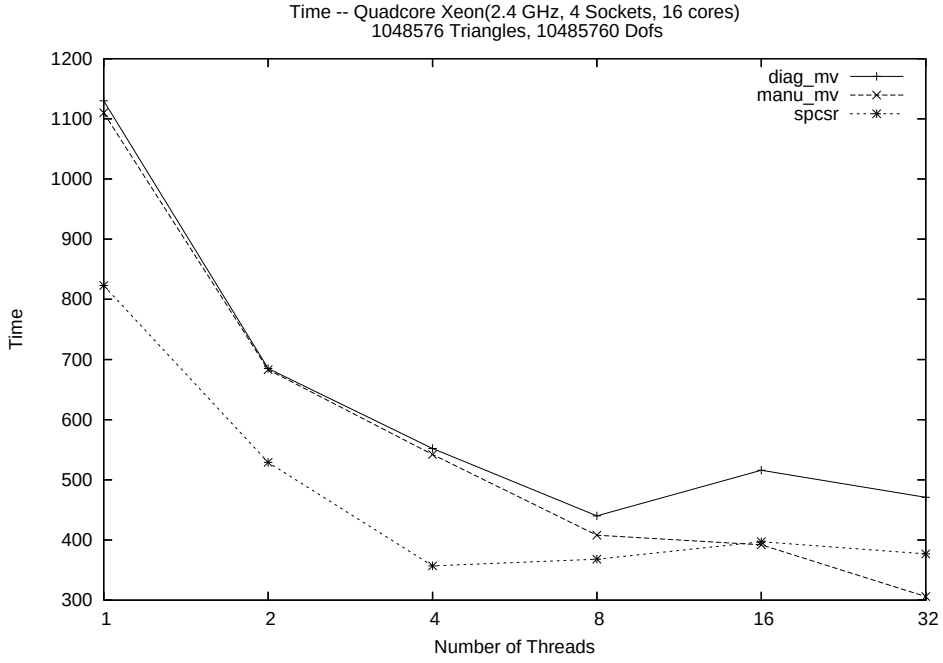


Figure 2.7: Comparison of computation time of CG method with CSR and RDR^T matrix storage format, degree of polynomial is 3

too. Implementation of block diagonal matrix vector multiplication for degrees of 1 and 4 will be our future work. We can see in the Figures that optimized RDR^T MV has better scalability on a multicore system. On the other hand, sparse MV starts with good performance with a single thread compared with RDR^T MV, but with poor data locality due to indirect memory access, it does not scale well on a multicore system.

Chapter 3

Nonoverlapping Additive Schwarz Preconditioners

3.1 Introduction

In the past fifteen years, extensive research has been done on developing domain decomposition methods for solving the systems of algebraic equations that arise from various discretizations of partial differential equations. The discretization methods that have been covered include finite difference methods, finite volume methods, (conforming, nonconforming and mixed) finite element methods, boundary element methods, spectral methods and mortar finite element methods (cf. [16, 26, 33, 36] and references therein). On the other hand, very few domain decomposition results have been known in the literature for discontinuous Galerkin methods (cf. [17, 23, 29]).

The work we present here is an improvement on [17] at both theoretical and algorithmic levels. First of all, the notation and proofs have been simplified and some unnecessary assumptions removed, and a more general problem is treated. Secondly, the use of continuous coarse mesh spaces leads to better preconditioning of the iterative solver of the linear systems and thus to faster execution.

We begin by summarizing some concepts and facts introduced in earlier chapters that will be used in this chapter. To begin, we recall that we are interested in solving the following boundary value problem:

Let $\Omega \subset \mathbf{R}^d$, $d = 2,3$, be a bounded open polyhedral domain. We consider the following boundary value problem:

$$-\Delta u = f \quad \text{in } \Omega, \tag{3.1}$$

$$u = g_D \quad \text{on } \Gamma_D, \tag{3.2}$$

$$\nabla u \cdot n = g_N \quad \text{on } \Gamma_N, \tag{3.3}$$

where $\partial\Omega := \Gamma = \Gamma_D \cup \Gamma_N$ and n is the unit normal vector exterior to Ω. We assume that Γ_D has positive measure, $f \in L^2(\Omega)$, $g_N \in L^2(\Gamma_N)$.

The discontinuous Galerkin formulation for the above problem leads to a system of linear equations of the form

$$A\mathbf{x} = \mathbf{b}. \tag{3.4}$$

It is not hard to show that the (2-norm) condition number of A is of the order $O(\underline{h}^{-2})$ where $\underline{h} = \min_{K \in \mathcal{T}_h} h_K$. Hence, the system (3.4) becomes ill-conditioned for small $\underline{h}$. The ill-conditioning worsens in situations when local refinement leads to extremely small values of $\underline{h}$. In addition, the size of the linear system becomes large. Consequently, it is not efficient to solve it directly using the classical iterative methods. On the other hand, if one can find a symmetric positive definite matrix B (the preconditioner) such that BA is well-conditioned, then any of the classical iterative methods (in particular, the Conjugate Gradient method) works effectively on the preconditioned system

$$BA\mathbf{x} = B\mathbf{b}. \tag{3.5}$$

Our goal here is to develop some additive Schwarz preconditioners, based on domain decomposition, for the linear system (3.4) and to solve the preconditioned systems using the Conjugate Gradient method. For background knowledge and a general theory on the Schwarz method, we refer to [26, 33, 36]. The major novelty of our approach is to use coarse mesh spaces of continuous functions. The reason for preferring continuous spaces over discontinuous ones will be explained later. We stress however that the overall method is still the discontinuous Galerkin method as im-

plemented on the working (fine mesh) $\mathcal{T}_h$, whereas continuous spaces are used only to construct the preconditioner B.

3.1.1 Preliminaries

We introduce notation and list some basic facts that will be used often in this chapter. For appropriate definitions we refer to earlier chapters or the references quoted.

3.1.2 Sobolev Spaces

Let $D \subset \mathbf{R}^d$, $d = 2, 3$, be a bounded open polyhedral domain. For integer $m \geq 0$, $H^m(D)$ will denote the (Hilbert) Sobolev space with inner product

$$(u,v)_{m,D} = \sum_{|\alpha| \leq m} \int_D D^\alpha u D^\alpha v \quad \text{and norm} \quad \|u\|_{m,D} = (u,u)_{m,D}^{1/2}.$$

(cf. [1]). To simplify the notation, we shall drop m when its value is zero. Also, we shall often encounter functions that vanish on a subset Γ_D of the boundary ∂D. We thus let

$$H^1_{0,\Gamma_D} = \{v \in H^1(\Omega), v = 0 \text{ on } \Gamma_D\}.$$

Extensive use will be made of edge/surface integrals. Therefore, for a $(d-1)$-dimensional subset e of $\mathbf{R}^d$, we set

$$\langle u,v \rangle_e = \int_e uv\,ds \quad \text{and} \quad |u|_e = \langle u,u \rangle_e^{1/2}.$$

3.1.3 Triangulations

Let $\mathcal{T}_h = \{K_i : i = 1, 2, \ldots, m_h\}$ $h > 0$ be a family of star-like partitions (triangulations) of the domain Ω parametrized by $0 < h \leq 1$. We assume the following:

(i) For all $h > 0$,the elements of $\mathcal{T}_h$ satisfy the minimal angle condition.

(ii) $\mathcal{T}_h$ is locally quasiuniform; that is, if two elements K_j and K_ℓ are *adjacent*, i.e. their boundaries have a nonempty intersection, then $\mathrm{diam}(\mathrm{K_j}) \approx \mathrm{diam}(\mathrm{K}_\ell)$.

The condition of local quasiuniformity, in contrast with global quasiuniformity, is compatible with local refinement.

We define $\mathcal{E}_h^I$ and $\mathcal{E}_h^B$ to be the set of all interior and boundary edges (faces in the case $d = 3$), respectively, as follows:

$$\mathcal{E}_h^I = \{e = \partial K_j \cap \partial K_\ell, \quad \mu_{d-1}(\partial K_j \cap \partial K_\ell) > 0\},$$
$$\mathcal{E}_h^B = \{e = \partial K \cap \partial\Omega, \quad \mu_{d-1}(\partial K \cap \partial\Omega) > 0\}, \qquad \mathcal{E}_h = \mathcal{E}_h^I \cup \mathcal{E}_h^B.$$

For each $e \in \mathcal{E}_h^I$, we denote the two triangles that "share" it by K^+ and K^-, respectively. Which of the two is K^+ is completely arbitrary but not irrelevant! If $e \in \mathcal{E}_h^B$, then $e = \partial K^+ \cap \partial\Omega \equiv \partial K \cap \partial\Omega$. We assume that for each $e \in \mathcal{E}_h^B$, either $e \subset \Gamma_D$ or $e \subset \Gamma_N$. We then set $\mathcal{E}_h^B = \mathcal{E}_h^D \cup \mathcal{E}_h^N$, where $\mathcal{E}_h^D$ and $\mathcal{E}_h^N$ are, respectively, the set of boundary edges on Γ_D and on Γ_N. From the previous assumption, we have $\mathcal{E}_h^D \cap \mathcal{E}_h^N = \emptyset$.

Given a partition or mesh $\mathcal{T}_h$ of Ω, we find it convenient to use the so-called *broken* or discontinuous Sobolev spaces $H^m(\mathcal{T}_h) = \Pi_{K\in\mathcal{T}_h} H^m(K)$. Elements of these spaces are not functions in the proper sense given that they may be multivalued on interior edges/faces of the partition $\mathcal{T}_h$. However, since such edges have $(d-1)$-dimensional measure zero, we can still treat the elements of these spaces as functions.

It is essential to be able to define values of functions in $H^m(\mathcal{T}_h)$ and V^h on the edges e. Thus, for $v \in H^m(\mathcal{T}_h)$, $m \geq 1$, and $e \in \mathcal{E}_h^I \cup \mathcal{E}_h^B$, v_e^+ will denote the trace on e of the restriction v^+ of v to K^+. Similarly we define v_e^- for $e \in \mathcal{E}_h^I$.

We also define *jumps* and *averages* of such traces as follows:

$$[v] = v_e^+ - v_e^-,\ e \in \mathcal{E}_h^I, \qquad [v] = v_e^+,\ e \in \mathcal{E}_h^B,$$
$$\{v\} = \tfrac{1}{2}\left(v_e^+ + v_e^-\right),\ e \in \mathcal{E}_h^I, \qquad \{v\} = v_e^+,\ e \in \mathcal{E}_h^B.$$

Finally, for $v \in H^2(\mathcal{T}_h)$ and an interior edge $e \in \mathcal{E}_h^I$, we define the *average* and *jump* of the normal derivative of v by

$$\{\partial_n v\}_e = \frac{1}{2}\left(\nabla v^+ + \nabla v^-\right)\cdot \mathbf{n}^+ \quad \text{and} \quad [\partial_n v]_e = \nabla v^+ \cdot \mathbf{n}^+ - \nabla v^- \cdot \mathbf{n}^+,$$

respectively, where $\mathbf{n}^+$ is the unit outward normal to K^+

3.1.4 The discontinuous Galerkin approximation

We define the bilinear form $a_h^\gamma(\cdot,\cdot) : H^2(\mathcal{T}_h) \times H^2(\mathcal{T}_h) \to R$ by

$$\begin{aligned} a_h^\gamma(u,v) &= \sum_{K\in\mathcal{T}_h} (\nabla u, \nabla v)_K \\ &\quad - \sum_{e\in\mathcal{E}_h^I\cup\mathcal{E}_h^D} \Big(\langle \{\partial_n u\}, [v]\rangle_e + \langle \{\partial_n v\}, [u]\rangle_e - \gamma h_e^{-1} \langle [u], [v]\rangle_e \Big). \end{aligned} \tag{3.6}$$

For the construction and motivation behind a variety of discontinuous Galerkin methods we refer to the survey paper [3]. The above bilinear form is consistent with the boundary value problem (3.1)-(3.3) in the sense that if u is a solution thereof, then using integration by parts one can show that for all $v \in H^1(\mathcal{T}_h)$,

$$a_h^\gamma(u,v) = F(v) := (f,v) + \sum_{e\in\mathcal{E}_h^N} \langle g_N, v\rangle_e - \left\langle g_D, \partial_n v - \gamma h_e^{-1} v\right\rangle_e.$$

To define the discontinuous Galerkin formulation for the BVP (3.1)-(3.3), we introduce the discontinuous finite element space

$$V^h = \mathcal{P}_q(\mathcal{T}_h) := \{v : v|_K \in \mathcal{P}_q(K), K \in \mathcal{T}_h, q \geq 1\}$$

where $\mathcal{P}_q$ is the space of polynomial of degree less than or equal to q.

Then, we define the discontinuous Galerkin approximation u_h^γ as the element in V^h that satisfies

$$a_h^\gamma(u_h^\gamma, v) = F(v), \quad \forall v \in V^h. \tag{3.7}$$

The bilinear form a_h^γ is symmetric, continuous and coercive on V^h. Specifically, if we define the DG-norm

$$\|v\|_{DG} = \Big\{ \sum_{K\in\mathcal{T}_h} \|\nabla v\|_K^2 + \gamma \sum_{e\in\mathcal{E}_h^I\cup\mathcal{E}_h^D} h_e^{-1}|[v]|_e^2 \Big\}^{1/2},$$

then

$$a_h^\gamma(u,v) \le c\|u\|_{DG}\|v\|_{DG}, \quad \forall u,v\in V^h. \tag{3.8}$$

Furthermore, there exists a constant γ_0 depending on q and the minimum angles of the cells such that for $\gamma \ge \gamma_0$,

$$a_h^\gamma(v,v) \ge c\|v\|_{DG}^2, \quad \forall v\in V^h. \tag{3.9}$$

Choosing a basis $\{\phi_j,\ j=1,\dots,J\}$ for V^h, the above formulation leads to the system (3.4) with A being the $J\times J$ *stiffness* matrix $A_{ij} := a_h^\gamma(\phi_j,\phi_i)$ and the J-vector b given by $b_i = F(\phi_i)$. The matrix A is symmetric since a_h^γ is symmetric and positive definite as a consequence of (3.9).

3.1.5 Some useful results

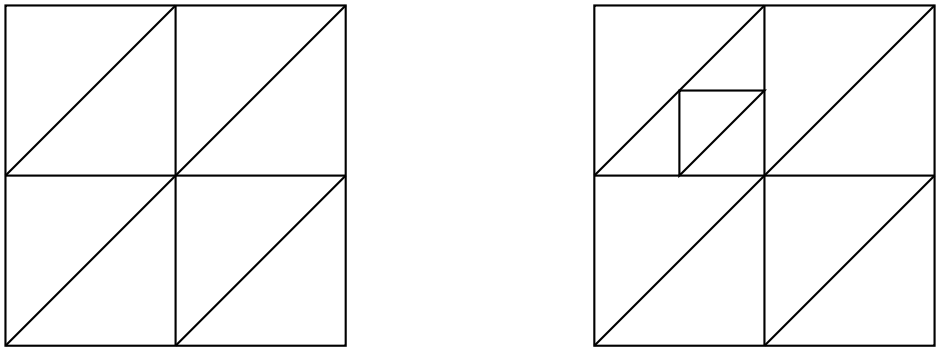
Theorem 3.1.1. *Let D be a starlike domain and let $u\in H^m(D)$ for some $m\ge 0$. Then there exists a $\chi\in\mathcal{P}_q(D)$ such that*

$$|u-\chi|_{j,D} \le c h_D^{i-j}|u|_i, \quad 0\le j\le i\le \min\{m,q+1\}. \tag{3.10}$$

This is a basic approximation property based on the Taylor polynomial. For a proof we refer to [4] and [8].

The following two inequalities known as trace and inverse inequalities, respectively, are well known in finite elements, and their proofs can be found in [8].

$$|v|_{\partial D}^2 \le c\left(h_D^{-1}\|v\|_D^2 + h_D\|\nabla v\|_D^2\right) \quad \forall v\in H^1(D), \tag{3.11}$$



(a) Conforming mesh (b) Non-conforming mesh

Figure 3.1: Conforming and non-conforming mesh

where $h_D = \text{diam}(D)$;

$$|v|_{j,D} \leq c h_D^{i-j} |v|_{i,D} \quad \forall v \in \mathcal{P}_q(D),\ 0 \leq i \leq j \leq 2. \tag{3.12}$$

A less standard property that has proved very useful in the context of discontinuous Galerkin methods concerns the approximation of discontinuous piecewise polynomials by continuous piecewise polynomials of the same degree.

We shall call a mesh $\mathcal{T}_h$ *conforming* if whenever two elements of the mesh share an edge (face in 3 dimensions), the latter must be a full edge for both elements. Otherwise the mesh will be called *nonconforming*. See Figure 3.1

Theorem 3.1.2. *Let $\mathcal{T}_h$ be a conforming or nonconforming mesh consisting of triangles when $d = 2$, and tetrahedra when $d = 3$. Then for any $v_h \in V^h$ the following approximation results hold:*

(i) *There exists $\chi_h \in V^h \cap H^1(\Omega)$ satisfying*

$$\sum_{K \in \mathcal{T}_h} \xi_K \|v_h - \chi_h\|_K^2 \leq c \sum_{e \in \mathcal{E}_h^I} \xi_K h_e |[v_h]|_e^2, \tag{3.13}$$

$$\sum_{K \in \mathcal{T}_h} \zeta_K \|\nabla(v_h - \chi_h)\|_K^2 \leq c \sum_{e \in \mathcal{E}_h^I} \zeta_K h_e^{-1} |[v_h]|_e^2, \tag{3.14}$$

(ii) *There exists* $\chi_h \in V^h \cap H^1_{0,\Gamma_D}$ *satisfying*

$$\sum_{K\in\mathcal{T}_h} \xi_K \|v_h - \chi_h\|_K^2 \leq c \sum_{e\in\mathcal{E}_h^{I\cup D}} \xi_K h_e |[v_h]|_e^2, \tag{3.15}$$

$$\sum_{K\in\mathcal{T}_h} \zeta_K \|\nabla(v_h - \chi_h)\|_K^2 \leq c \sum_{e\in\mathcal{E}_h^{I\cup D}} \zeta_K h_e^{-1} |[v_h]|_e^2. \tag{3.16}$$

Here, $\mathcal{E}_h^{I\cup D} = \mathcal{E}_h^I \cup \mathcal{E}_h^D$ *is the collection of interior and Dirichlet type boundary edges.* ξ_K *and* ζ_K *are quantities that will be chosen appropriately and so that they vary in a locally quasi uniform manner similar to* h_K *and* h_e*.*

3.2 The non-overlapping Schwarz methods

In this section, we develop some two-level non-overlapping additive Schwarz preconditioners for the discontinuous Galerkin method. Similar results were obtained in [17] with important differences from the treatment presented herein which we enumerate now

1. The problem treated in [17] involved the simpler case of the homogeneous Dirichlet boundary condition.

2. The subspace used in [17] for the coarse mesh correction was large enough to include piecewise constant functions. Here we aim at using a much smaller space consisting of piecewise linear functions that are continuous in Ω and that vanish on the Dirichlet type boundary Γ_D. The benefits of working with a smaller space are twofold. First, the resulting matrix is much smaller. More significantly, the incompatibility resulting from the penalty terms is totally eliminated. We explain this point in detail later.

3. The proofs are considerably simplified. In some cases, e.g. Proposition 3.2.1, a restrictive assumption is removed via a a new and direct treatment.

3.2.1 Formulation of the additive Schwarz preconditioner

It is well-known (cf. [33, 36]) that the first step towards constructing additive Schwarz preconditioners is to have a valid subspace decomposition of the finite element space V^h. For the discontinuous Galerkin method considered in this paper, since $V^h \subset L^2(\Omega)$ and no continuity constraints are imposed on the functions in V^h, it is easy to construct such a space decomposition. This is in sharp contrast with the situation in the standard as well as nonconforming Galerkin formulation (cf. [18, 28]).

Let $\mathcal{T}_S$ denote a partition of Ω into p non-overlapping subdomains $\Omega_j,\ j = 1, \ldots, p$. We assume that $\mathcal{T}_S$ is aligned with $\mathcal{T}_h$ in the sense that each Ω_j is some union of cells in $\mathcal{T}_h$. Therefore to each subdomain Ω_j we associate in a natural way a subspace V_j of V^h given by

$$V_j = \{v \in V^h | v = 0 \text{ in } \Omega \setminus \overline{\Omega}_j\}.$$

In other words, V_j is simply the restriction of V^h to Ω_j. Also, given the discontinuous nature of the functions in V^h, we have the following direct sum decomposition

$$V^h = V_1 \oplus V_2 \oplus \cdots \oplus V_p. \tag{3.17}$$

The above subspace decomposition will not produce a good preconditioner. What is needed is a coarse mesh $\mathcal{T}_H$ and a corresponding coarse subspace V^H (or preferably V_0) of V^h where the residuals will be "projected and corrected" just as in a multigrid method. This is referred to as the coarse-mesh correction. Since the novelty of our approach resides in the choice of V_0, we make a detailed list of the properties of $\mathcal{T}_H$.

(A1) $\mathcal{T}_H$ is aligned with both $\mathcal{T}_h$ and $\mathcal{T}_S$ in the sense that every cell in $\mathcal{T}_S$ is a union of cells in $\mathcal{T}_H$ and every cell in $\mathcal{T}_H$ is a union of cells in $\mathcal{T}_h$. In particular, for each $D \in \mathcal{T}_H$, there is a subset $\mathcal{T}_D$ of $\mathcal{T}_h$ such that $D = \cup_{K \in \mathcal{T}_D} K$.

(A2) D is starlike in the sense that there exists $\mathbf{x}_0 \in D$ and a constant $c_D > 0$ such that

$$(\mathbf{x} - \mathbf{x}_0) \cdot \mathbf{n} \geq c_D \quad \text{for almost all } \mathbf{x} \in \partial D, \tag{3.18}$$

where $\mathbf{n}$ denotes the unit outward normal vector to ∂D. Furthermore for some constant $c = O(1)$,

$$c_D \geq cH_D$$

where H_D is the diameter of D.

(A3) The partition $\mathcal{T}_D$ is quasi-uniform in the sense that the cells from $\mathcal{T}_h$ that make up D are all of similar sizes. In particular, for a given $D \in \mathcal{T}_H$, we let

$$h_D = \max_{K \in \mathcal{T}_D} h_K.$$

Remark 3.2.1. *Assumption (A2) implies that D is not too "thin"; indeed D satisfies the minimum angle condition. However we are not assuming that D is convex. Also to allow for local refinement of $\mathcal{T}_h$, two cells in two different D's may be of vastly different sizes.*

Remark 3.2.2. *Assumption (A2) and the fact that elements of $\mathcal{T}_h$ satisfy the minimum angle condition imply that the number of cells in any given D is bounded by $(H_D/h_D)^d$.*

We now introduce the coarse mesh subspace V_0 of V^h

$$V_0 = \{v \in \mathcal{P}_1(\mathcal{T}_H), v \in C^0(\Omega), v = 0 \text{ on } \Gamma_D\},$$

In other words, elements of V_0 are continuous piecewise linear functions that vanish on Γ_D.

With the subspaces $V_0, V_1, \ldots, V_p$ at hand, we define local or subdomain bilinear forms $a_i(\cdot,\cdot)$, $j = 1, \ldots, p$ and a coarse space bilinear form $a_0(\cdot,\cdot)$ as the restrictions of $a_h^\gamma(\cdot,\cdot)$ to the subspaces $V_0, V_1, \ldots, V_p$ respectively; i.e.

$$a_j(u,v) = a_h^\gamma(u,v), \quad \forall u, v \in V_j,\ j = 0, 1, \ldots, p. \tag{3.19}$$

It is clear that these bilinear forms inherit the symmetry and coercivity properties of $a_h^\gamma(\cdot,\cdot)$. Hence the corresponding (stiffness) matrices $A_0, A_1, \ldots, A_p$ are symmetric, positive definite. A_0 is similar in structure to the matrix A corresponding to the form $a_h^\gamma(\cdot,\cdot)$ but of course much smaller. On the other hand, the matrices $A_1, \ldots, A_p$ are closely related to A in that they are indeed the Jacobi blocks of A corresponding to the individual subdomains.

At this juncture, we are able to motivate the reason for requiring the coarse space functions to be continuous. We defined the coarse space bilinear form $a_0(u,v)$ to be equal to $a_h^\gamma(u,v)$ whenever u and v belong to the coarse space V_0. When we examine the penalty terms, we realize that in order for equality to hold the parameter γ in $a_0(u,v)$ must be larger than the γ in $a_h^\gamma(u,v)$ to compensate for the differences in the respective edge lengths $h_e, e \in \mathcal{E}_H^I \cup \mathcal{E}_H^D$ and $h_e, e \in \mathcal{E}_h^I \cup \mathcal{E}_h^D$. This can be done in the case of uniform refinement but not when the mesh is locally refined. The continuity of the elements of V_0 completely eliminates this problem.

The next Lemma, whose proof is obvious, exhibits a relation between a_h^γ and the subdomain bilinear forms paralleling the direct sum decomposition (3.17) and also illuminates the previous comment.

Lemma 3.2.1. *For $u, v \in V^h$, let $u_i, v_i \in V_i, i = 1, \ldots, p$ be given (uniquely) by $u = \sum_{i=1}^p u_i, \quad v = \sum_{i=1}^p v_i$. Then, the following identity holds*

$$a_h^\gamma(u,v) = \sum_{i=1}^p a_i(u_i, v_i) + I(u,v), \tag{3.20}$$

where $I(\cdot,\cdot)$ is the *interface* bilinear form given by

$$\begin{aligned} I(u,v) \;=\; & \sum_{e \in \mathcal{S}} \Big(\frac{1}{2} \left\langle \partial_n u^+, v^- \right\rangle_e - \frac{1}{2} \left\langle \partial_n u^-, v^+ \right\rangle_e - \gamma h_e^{-1} \left\langle u^+, v^- \right\rangle_e \\ & + \frac{1}{2} \left\langle \partial_n v^+, u^- \right\rangle_e - \frac{1}{2} \left\langle \partial_n v^-, u^+ \right\rangle_e - \gamma h_e^{-1} \left\langle v^+, u^- \right\rangle_e \Big), \end{aligned} \tag{3.21}$$

and $\mathcal{S}$ is the "skeleton" of the subdomain partition defined by

$$\mathcal{S} = \{ e \in \mathcal{E}_h^I \text{ and } e \in \partial\Omega_i \text{ for some } i \}.$$

In essence, the interface bilinear form $I(\cdot,\cdot)$ contains those edge integrals in $a_h^\gamma(\cdot,\cdot)$ that are not contained in all the subdomain bilinear forms $a_i(\cdot,\cdot)$. Also, contrary to the latter, the form $I(\cdot,\cdot)$ will not be involved in the calculations but will prove useful in the analysis and the method.

In order to construct the additive Schwarz preconditioner, we introduce the projection operators $T_j : V^h \to V_j$, $j = 0,\ldots,p$ according to

$$a_j(T_j u, v) = a_h^\gamma(u,v) \qquad \forall v \in V_j,\ j = 0,1,2,\cdots,p. \tag{3.22}$$

These operators are well defined since the bilinear forms involved are coercive. The additive Schwarz operator T is defined by

$$T = T_0 + T_1 + \cdots + T_p. \tag{3.23}$$

Following the framework given in [16, 33, 36], the additive Schwarz method consists in replacing the discrete problem 3.4 by the equation

$$Tu = g, \qquad\qquad g = \sum_{j=0}^{p} g_j, \tag{3.24}$$

where $g_j = T_j u$ is defined as the solution of

$$a_j(g_j, v) = F(v) \qquad \forall v \in V_j^h,\ j = 0,1,2,\cdots,p. \tag{3.25}$$

In matrix notation, the additive Schwarz preconditioner corresponds to choosing the matrix B (3.5) as

$$B = R_0^T A_0^{-1} R_0 + (R_1^T A_1 R_1 + \cdots + R_p^T A_p R_p)^{-1} \tag{3.26}$$

where A_j is the stiffness matrix corresponding to $a_j(\cdot,\cdot)$ and R_j^T is the matrix representation of the embedding operator $i : V_j^h \to V^h$, $j = 0,\ldots,p$.

Now the question is whether the preconditioned system (3.24) is well-conditioned, in particular,

whether the condition number of T, or equivalently that of the matrix BA, depends "favorably" on the mesh sizes h and H. These questions will be addressed in the next subsection.

3.2.2 Condition number estimate for the additive Schwarz method

To estimate the condition number of T, we shall use the general abstract convergence theory of Schwarz methods given in [33]. We shall do so by verifying that a set of three *Assumptions* are satisfied and by estimating the constants $C_0^2, \rho(\mathcal{E})$ and ω appearing there in terms of the parameters of our method. (cf. page 155 of [33])

The verification of the first assumption requires showing that for all $u \in V^h$

$$\sum_{i=0}^{p} a_i(u_i, u_i) \leq C_0^2 a_h^{\gamma}(u,u), \tag{3.27}$$

for *some* representation $u = \sum_{i=0}^{p} u_i$.

To establish (3.27), we will need some preliminary results. The first result concerns a trace inequality that holds on the boundary of $D \in \mathcal{T}_H$. Its proof is found in [17].

Lemma 3.2.1. *For any $u \in H^1(\mathcal{T}_D)$, there holds the following trace inequality*

$$|u|_{\partial D}^2 \leq cH_D^{-1}\|u\|_D^2 + cH_D\Big(\sum_{K\in\mathcal{T}_D}\|\nabla u\|_K^2 + \sum_{e\in\mathcal{E}_h^I(D)} h_e^{-1}|[u]|_e^2\Big). \tag{3.28}$$

The next preliminary result concerns the approximation a discontinuous function by a (globally) constant function. This was proved in [17] under a convexity assumption. The proof we present herein is new and more direct and avoids this assumption.

Proposition 3.2.1. *Let $D \in \mathcal{T}_H$ satisfy assumptions (A1)-(A3) and let $u \in H^1(\mathcal{T}_D)$. Then, there exists a function $\bar{v}$ which is constant on D and such that*

$$\|u-\bar{v}\|_D \leq c = cH_D\left\{\sum_{K\in\mathcal{T}_D}\|\nabla u\|_K^2 + \sum_{e\in\mathcal{E}_h^I(D)} h_e^{-1}|[u]|_e^2\right\}^{1/2} \tag{3.29}$$

where $\mathcal{E}_h^I(D)$ denotes the set of edges in $\mathcal{T}_D$ in the interior of D.

Proof. We first approximate u by a function v which is piecewise constant on $\mathcal{T}_D$. Indeed, for $K \in \mathcal{T}_D$, let $v_K = v|_K$ be a constant function that approximates $u|_K$ according to the general approximation result 3.10; namely $\|u - v_K\|_K \leq c h_K \|\nabla u\|_K$. Thus,

$$\|u - v\|_D^2 \leq c \sum_{K \in \mathcal{T}_D} h_K^2 \|\nabla u\|_K^2. \tag{3.30}$$

Let the value of v on K be denoted by α_K. The average of the $\alpha_K's$ is given by $\beta = \frac{1}{|\mathcal{T}_D|} \sum_{K \in \mathcal{T}_D} \alpha_K$, where $|\mathcal{T}_D|$ is the number of cells in $\mathcal{T}_D$. Letting $\bar{v}$ be the constant function on D with value β, we have

$$\|v - \bar{v}\|_D^2 = \sum_{K \in \mathcal{T}_D} \|v - \bar{v}\|_K^2 = \sum_{K \in \mathcal{T}_D} \text{measure}(K) |\alpha_K - \beta|^2.$$

Since cells in D are of similar diameter h_D, we have $\text{measure}(K) = c(h_D)^d$. Thus,

$$\|v - \bar{v}\|_D^2 = c(h_D)^d \sum_{K \in \mathcal{T}_D} |\alpha_K - \beta|^2. \tag{3.31}$$

Now for any $K \in \mathcal{T}_D$,

$$|\alpha_K - \beta|^2 = \frac{1}{|\mathcal{T}_D|^2} \left| \sum_{K' \in \mathcal{T}_D} |\alpha_K - \alpha_{K'}| \right|^2.$$

We now make two important observations: In view of assumptions (A1)-(A3) and the fact that cells in $\mathcal{T}_D$ are of similar size h_D, the number $|\mathcal{T}_D|$ of cells in D is about $(\frac{H_D}{h_D})^d$. Furthermore, given any pair $K, K' \in \mathcal{T}_D$, there exists a path $\pi(K,K') = \{K_1, \ldots, K_{|\pi(K,K')|}\}$ of cells in $\mathcal{T}_D$ such that

(i) K_i and K_{i+1} share an edge e; so we may think of the path as a set of edges in $\mathcal{E}_h^I(D)$.

(ii) The number $|\pi(K,K')|$ of cells, i.e. edges, in the path is $O(H_D/h_D)$.

Hence, using the discrete Cauchy-Shwarz inequality, we get

$$|\alpha_K - \beta|^2 \leq \frac{|\pi(K,K')|}{|\mathcal{T}_D|^2} \sum_{K' \in \mathcal{T}_D} \sum_{e \in \pi(K,K')} |[\alpha]|^2,$$

where $[\alpha]$ is the difference of the values of $\alpha's$ across the edge e; i.e. if $e = \partial K^+ \cap \partial K^-$ then

$[\alpha] = \alpha_{K^+} - \alpha_{K^-}$. Hence, summing over K we obtain

$$\begin{aligned}\sum_{K\in\mathcal{T}_D} |\alpha_K - \beta|^2 &\leq \frac{|\pi(K,K')|}{|\mathcal{T}_D|^2} \sum_{K\in\mathcal{T}_D}\sum_{K'\in\mathcal{T}_D}\sum_{e\in\pi(K,K')} |[\alpha]|^2 \\ &\leq c\left(\frac{H_D}{h_D}\right)^2 \sum_{e\in\mathcal{E}_h^I(D)} |[\alpha]|^2.\end{aligned}$$

We now observe that for $e \in \mathcal{E}_h^I(D)$, $[\alpha] = [v]_e$, i.e. the jump of v across e. Furthermore, $|[\alpha]|^2 = h_D^{1-d}|[v]|_e^2$. Hence,

$$\sum_{K\in\mathcal{T}_D} |\alpha_K - \beta|^2 \leq c\left(\frac{H_D}{h_D}\right)^2 h_D^{2-d} \sum_{e\in\mathcal{E}_h^I(D)} h_e^{-1}|[v]|_e^2. \tag{3.32}$$

Using (3.32) in (3.31) we obtain

$$\|v - \bar{v}\|_D^2 \leq H_D^2 \sum_{e\in\mathcal{E}_h^I(D)} h_e^{-1}|[v]|_e^2. \tag{3.33}$$

It remains to estimate the term $\sum_{e\in\mathcal{E}_h^I(D)} h_e^{-1}|[v]|_e^2$ in terms of u. Writing $[v] = v^+ - u^+ + [u] + u^- - v^-$, we have from the trace inequality (3.11)

$$\begin{aligned}h_e^{-1}|[v]|_e^2 &\leq 3h_e^{-1}\left(|v^+ - u^+|_e^2 + |[u]|_e^2 + |u^- - v^-|_e^2\right) \\ &\leq ch_e^{-1}|[u]|_e^2 + ch_e^{-1}\sum_{K=K^+,K^-}\left(h_e^{-1}\|v-u\|_K^2 + h_e\|\nabla(v-u)\|_K^2\right).\end{aligned}$$

Since v is piecewise constant, $\nabla v = 0$. Hence, using (3.30) we obtain

$$h_e^{-1}|[v]|_e^2 \leq ch_e^{-1}|[u]|_e^2 + \left(\|\nabla u\|_{K^+}^2 + \|\nabla u\|_{K^-}^2\right). \tag{3.34}$$

The conclusion now follows from (3.30),(3.33), (3.34) and the triangle inequality. □

We next obtain a bound for the interface bilinear form I.

Proposition 3.2.2. *There exists a constant c such that for any $w \in V^h$*

$$|I(w,w)| \leq c \sum_{D\in\mathcal{T}_H} \Bigg\{ H_D^{-1} h_D^{-1} \sum_{K\in\mathcal{T}_D} \|w\|_K^2 + H_D h_D^{-1} \Bigg(\sum_{K\in\mathcal{T}_D} \|\nabla w\|_K^2 + \sum_{e\in\mathcal{E}_h^I(D)} h_e^{-1} |[w]|_e^2 \Bigg) \Bigg\}. \tag{3.35}$$

Proof. Using the definition (3.21), we have

$$I(w,w) = \sum_{e\in\mathcal{S}} \Big(\langle \partial_n w^+, w^- \rangle_e - \langle \partial_n w^-, w^+ \rangle_e - 2\gamma h_e^{-1} \langle w^+, w^- \rangle_e \Big). \tag{3.36}$$

Using the Cauchy-Schwarz inequality, the trace and inverse inequalities (3.11), (3.12) and the a.g.m.i., we get

$$|I(w,w)| \leq \sum_{K\in\mathcal{T}_h} \|\nabla w\|_K^2 + c\gamma \sum_{e\in\mathcal{S}} h_e^{-1} \big(|w^+|_e^2 + |w^-|_e^2 \big). \tag{3.37}$$

Since both $\mathcal{T}_h$ and $\mathcal{T}_H$ are aligned with $\mathcal{T}_S$, each $e \in \mathcal{S}$ belongs to the boundary of some D in $\mathcal{T}_H$. Thus,

$$|I(w,w)| \leq \sum_{K\in\mathcal{T}_h} \|\nabla w\|_K^2 + c\gamma \sum_{D\in\mathcal{T}_H} h_d^{-1} |w|_{\partial D}^2. \tag{3.38}$$

The result now follows from the trace estimate in Lemma 2.2 □

We can now verify assumption (3.27).

Theorem 3.2.1. *For any $u \in V^h$ there exists a decomposition $u = \sum_{j=0}^{p} u_j,\ u_j \in V_j,\ j = 0,\ldots,p$ for which* (3.27) *holds with $C_0^2 = \dfrac{H}{h}$.*

Proof. The main task here is to construct u_0. This is done in two stages. First, we let $\bar{v}$ be the piecewise constant function on $\mathcal{T}_H$ that approximates u on each D in $\mathcal{T}_H$ in the sense of (3.29). We then let u_0 be the element of the coarse mesh space V_0 which approximates $\bar{v}$ in the sense of Theorem 3.1.2. We then define $u_1,\ldots,u_p$ as uniquely given by

$$u - u_0 = u_1 + \cdots + u_p.$$

Now from (3.20) we have

$$a_h^\gamma(u-u_0,u-u_0)=\sum_{j=1}^{p}a_j(u_j,u_j)+I(u-u_0,u-u_0).$$

Hence, using Schwarz's inequality on the bilinear forms and the fact that $a_0(u_0,u_0)=a_h^\gamma(u_0,u_0)$, we have

$$\begin{aligned}\sum_{j=0}^{p}a_j(u_j,u_j) &= a_h^\gamma(u-u_0,u-u_0)+a_h^\gamma(u_0,u_0)-I(u-u_0,u-u_0)\\ &\le 2a_h^\gamma(u,u)+3a_h^\gamma(u_0,u_0)+|I(u-u_0,u-u_0)|. \end{aligned}\tag{3.39}$$

Now since u_0 is continuous on Ω and vanishes on Γ_D,

$$a_h^\gamma(u_0,u_0)=\sum_{K\in\mathcal{T}_h}\|\nabla u_0\|_K^2=\sum_{D\in\mathcal{T}_H}\|\nabla u_0\|_D^2=\sum_{D\in\mathcal{T}_H}\|\nabla(u_0-\bar v\|_D^2 \tag{3.40}$$

since $\bar v$ is piecewise constant on $\mathcal{T}_H$. Using the approximation property (3.16) with $\xi_K=1$ and with $\mathcal{T}_H$ instead of $\mathcal{T}_h$, we have

$$\begin{aligned}\sum_{D\in\mathcal{T}_H}\|\nabla(u_0-\bar v\|_D^2 &\le c\sum_{e\in\mathcal{E}_H^I\cup\mathcal{E}_H^D}h_e^{-1}|[\bar v]|_e^2\le c\sum_{D\in\mathcal{T}_H}H_D^{-1}\sum_{e\in\mathcal{E}_h\cap\partial D}|[\bar v]|_e^2\\ &\le c\sum_{D\in\mathcal{T}_H}H_D^{-1}\sum_{e\in\mathcal{E}_h\cap\partial D}\Big(|[u]|_e^2+|[u-\bar v]|_e^2\Big).\end{aligned}\tag{3.41}$$

Now the term $\sum_{D\in\mathcal{T}_H}H_D^{-1}\sum_{e\in\mathcal{E}_h\cap\partial D}|[u]|_e^2$ is certainly bounded by $a_h^\gamma(u,u)$. Also, using the trace inequality (3.28) and the approximation property (3.29) and noting that $[\bar v]=0$ for $e\in\mathcal{E}_h^I(D)$, we have

$$\sum_{D\in\mathcal{T}_H}H_D^{-1}\sum_{e\in\mathcal{E}_h\cap\partial D}|[u-\bar v]|_e^2\le\sum_{k\in\mathcal{T}_h}\|\nabla u\|_K^2+c\sum_{e\in\mathcal{E}_h^I\cup\mathcal{E}_h^D}h_e^{-1}|[u]|_e^2\le ca_h^\gamma(u,u).$$

Thus from (3.40) and (3.41) we obtain

$$a_h^\gamma(u_0,u_0)\le ca_h^\gamma(u,u). \tag{3.42}$$

It remains to bound the term $I(u-u_0,u-u_0)$. To do this, we use the bound (3.35) of Proposition (3.2.2) with $w=u-u_0$ to get

$$\begin{aligned}\Big|I(u-u_0,u-u_0)\Big| &\leq c \quad \sum_{D\in\mathcal{T}_H}\frac{H_D}{h_D}\Big(\sum_{K\in\mathcal{T}_D}\|\nabla u\|_K^2+\sum_{e\in\mathcal{E}_h^I(D)}h_e^{-1}|[u]|_e^2\Big)\\ &\leq c \quad \frac{H}{h}a_h^\gamma(u,u). \end{aligned}\tag{3.43}$$

The conclusion of the theorem now follows from (3.39), (3.42) and (3.43). □

Verifying *Assumption 2* consists in obtaining a bound for the spectral radius $\rho(\mathcal{E})$ of the $p\times p$ matrix $\mathcal{E}$ given as follows: Let $0\leq\mathcal{E}_{ij}\leq\tilde{C}$ be the minimal values such that

$$|a_h^\gamma(u_i,u_j)|\leq\mathcal{E}_{ij}a_h^\gamma(u_i,u_i)^{\frac{1}{2}}a_h^\gamma(u_j,u_j)^{\frac{1}{2}},\quad u_i\in V_i,u_j\in V_j,\ i,j=1,\cdots,p.$$

That such values exist is a consequence of the continuity and coercivity of $a_h^\gamma(\cdot,\cdot)$ as expressed in (3.8) and (3.9). The important thing however is to obtain a small bound on ρ. To do so, we observe that if two subdomains Ω_i and Ω_j are not adjacent, then $a_h^\gamma(u_i,u_j)=0$ for $u_i\in V^i,\ u_j\in V^j$. For the remaining cases, we take $\mathcal{E}_{ij}=\tilde{C}$, the constant in (3.8). We also let ν be the maximum number of adjacent subdomains any one given subdomain may have. Thus, if follows at once from Gershgorin's circle theorem that

$$\rho(\mathcal{E})\leq\tilde{C}(1+\nu).\tag{3.44}$$

In practice ν is usually ≤ 5. Even for "unusual" subdomain partitions, this number is not expected to be large.

As for *Assumption 3*, Let $\omega\in(0,1]$ be the minimum constant such that

$$a_h^\gamma(u_i,u_i)\leq\omega a_i(u_i,u_i),\quad\forall u_i\in V_i^h,\ i=0,\ldots,p.\tag{3.45}$$

Recall that we defined the subdomain bilinear forms $a_i(\cdot,\cdot)$ precisely by $a_i(u_i,u_i)=a_h^\gamma(u_i,u_i),\ i=0,\ldots,p$; thus (3.45) holds trivially with $\omega=1$.

With this, the central result of the chapter is at hand:

Table 3.1: Machine Specifications

Feature	Intel Xeon (Quad core)
Processor Speed	2.66GHz
L1 Cache	8KB
L2 Cache	8MB
Number of Sockets	4
Number of Cores	8
OS	Linux
Compiler	icc 11.0

Theorem 3.2.1. *The condition number* κ *of the operator* T *and equivalently of the matrix* BA *of the additive Schwarz method defined in this section satisfies*

$$\kappa \leq c\frac{H}{h}. \tag{3.46}$$

Proof. This is an immediate consequence of Lemma 3 in chapter 5 of [33] and our estimates (3.27), (3.44) and (3.45). □

3.3 Experiments and Performance

The following is a testing problem:

$$\begin{aligned} -\Delta u &= 128\pi^2 sin(8\pi x)sin(8\pi y) \qquad \text{in } \Omega \\ u &= 0 \qquad\qquad\qquad\qquad\quad \text{on } \partial\Omega \end{aligned} \tag{3.47}$$

The solution of Problem(3.47) is: $u = sin(8\pi x)sin(8\pi y)$, which is a smooth non-polynomial solution and oscillatory across the domain.

The experiments are set up as follows:

- The experiments are conducted on an Intel Xeon Quad core architecture with specifications shown in Table 3.1.
- Initial triangulations using `triangle` were done using maximum area constraint $a = 0.1$, and

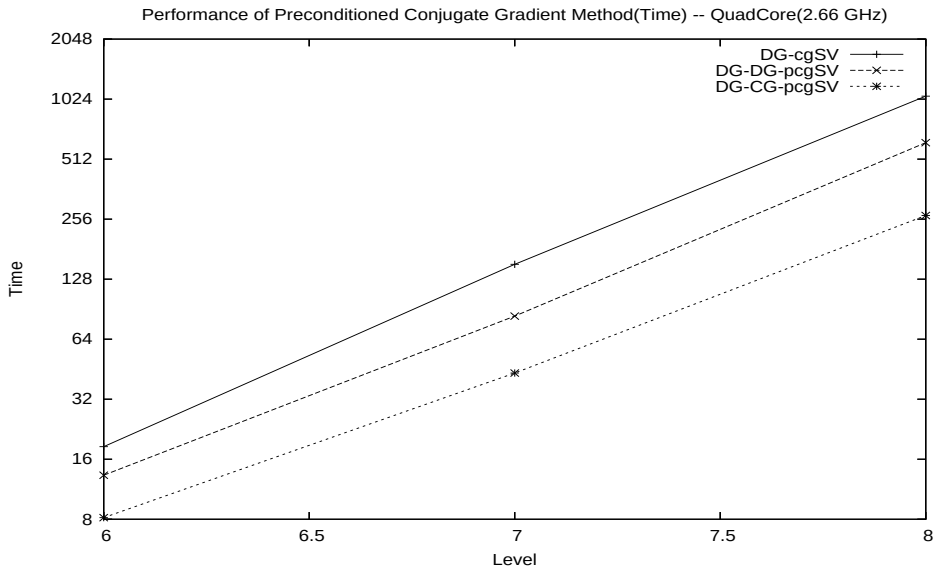


Figure 3.2: Comparison of computation time of iterative methods: CG and PCG

the finest mesh has 1048576 triangles.

- The iterative solver terminates at accuracy of 10^{-16}.

We conduct experiments to compare the performance of CG and PCG methods in term of number of iterations and compute time. See Figure 3.2 and 3.3:

- DG-cgSV is for Conjugate Gradient method.
- DG-DG-pcgSV is for Preconditioned Conjugate Gradient method with discontinuous Galerkin coarse mesh correction.
- DG-CG-pcgSV is for Preconditioned Conjugate Gradient method with continuous Galerkin coarse mesh correction.

We observe from Figure 3.3 that number of iterations for the conjugate gradient method increases exponentially as the problem size increases. As we indicated at the beginning of this chapter, each time the mesh is refined it leads to small value of $\underline{h} = \min_{K \in \mathcal{T}_h} h_K$. Consequently the (2-norm) condition number of the linear system 3.4 increases, being of order $O(\underline{h}^{-2})$. On the other hand, the number of iterations stays roughly the same for the preconditioned conjugate gradient method, in

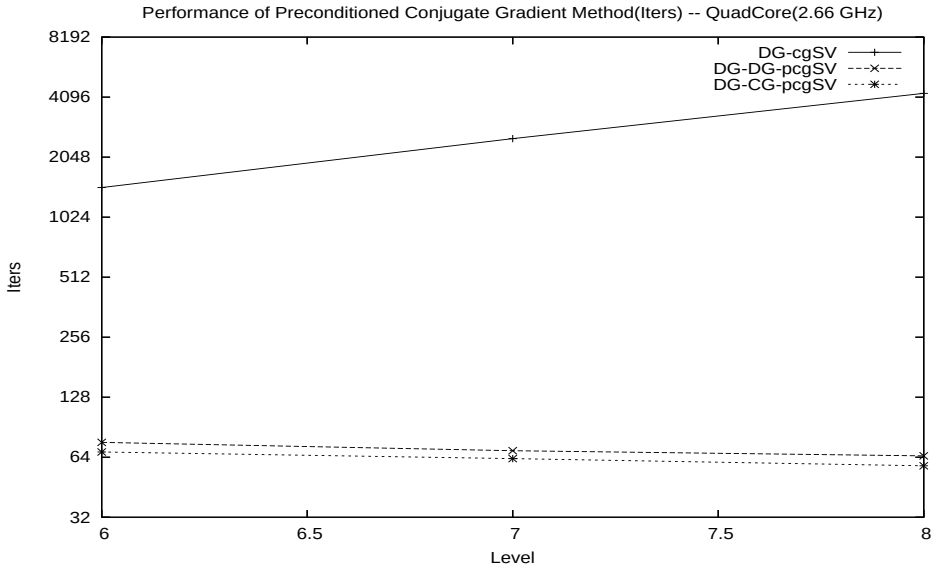


Figure 3.3: Number of iterations comparison of iterative methods: CG and PCG

agreement with Theorem 3.2.1. From Figure 3.2, we can see that PCG methods are much faster than CG method. PCG method with continuous Galerkin coarse mesh correction is faster than PCG method with discontinuous Galerkin coarse mesh correction. The difference is due to the fact that the linear system of continuous Galerkin FEM is much smaller than that of discontinuous Galerkin FEM.

Chapter 4

Adaptive Algorithm

Adaptive methods based on reliable a posteriori error estimates are essential for many large scale computations. An adaptive algorithm is basically an iterative algorithm consisting of a number of cycles of the form: $Solve \Rightarrow Estimate \Rightarrow Mark \Rightarrow Refine$

1. Given a mesh, compute a solution on this mesh;
2. Estimate the error using an a posteriori error estimator, if the error tolerance reached, then STOP;
3. Make/Refine the mesh;
4. Repeat steps 1 to 3 until the error is reduced to the desired level.

To achieve the prescribed error level, after each cycle of computation the a posteriori estimator3 provides information for the mesh refinement. The a posteriori estimator provides bounds by inequalities on both sides for the true error. The upper estimate can be used as the stopping criterion, while the lower estimate shows the precision of the estimator. This method produces a relatively much smaller linear system to solve while reaching the same level of accuracy. Given the set of local error estimates for every triangle K of the triangulation $\mathcal{T}$, the process to obtain a new triangulation has two parts: 1) selecting triangles to be refined(*marking strategy*) and 2) construction of the new triangulation(*refinement strategy*). We chose uniform refinement for each selected triangle in

the software. We compare two marking algorithms in the following section: dörfler algorithm [15] and drastic cutting algorithm.

4.1 Marking Algorithm

Let $\mathcal{T}_0$ be the initial triangulation which is fine enough to start with. At each adaptive iteration, let $\{\eta_K$: all $K \in \mathcal{T}_h\}$ be the error estimates computed. The global estimated error is:

$$\eta_{\mathcal{T}}{}^2 = \sum_{K \in \mathcal{T}_h} \eta_K{}^2 \tag{4.1}$$

and the largest error estimate is:

$$\eta_{max} = \max_{K \in \mathcal{T}_h} \eta_K \tag{4.2}$$

4.1.1 Dörfler Marking Algorithm

The Dörfler marking algorithm (see Algorithm(3)) constructs a set of elements $S \subset \mathcal{T}_h$ that is as small as possible satisfying $\eta_S{}^2 \geq \theta \eta_{\mathcal{T}}{}^2$. The choice of θ determines the fraction of the global estimator that one wants to refine. Choosing θ close to one would produce uniform refinement, i.e., all $K \in \mathcal{T}_h$ are refined. Choosing θ small would only choose a few elements to be marked each adaptive iteration, thus resulting in many adaptive iterations (and many solves) to reach the desired tolerance *htol*. When θ is small, one will usually obtain a more optimal mesh, but at a very large cost in adaptive iterations and overall solving time. Choosing a value from 0 to 1 for variable ν determines how fine the procedure will work. Smaller values of ν allow the marking strategy to step through the range of the estimator with finer step size. However, it is not necessary if all elements are sorted by their error estimates. The complete adaptive process stops when $\eta_{\mathcal{T}} \leq htol$ or $S = \emptyset$.

```
Choose θ ∈ (0,1) ;
Choose ν ∈ (0,1) ;
S = ∅ ;
sum := 0 ;
τ := 1 ;
while sum < θη_T^2 do
    τ := τ − ν ;
    foreach K ∈ T_h do
        if K is not marked then
            if η_K > τη_max then
                Mark K, S = S + K ;
                sum = sum + η_K^2 ;
            end
        end
    end
end
Output: S
```

Algorithm 3: Dörfler marking algorithm

4.1.2 Drastic Cutting Algorithm

We now present another marking strategy: drastic cutting marking algorithm (see Algorithm(4)). Let $|K|$ denote the area of a triangle, and let $|\Omega|$ denote the area of the domain. To reach the condition: $\sum_K ||\nabla e||_K^2 \leq htol^2$, it is sufficient to distribute errors as follows:

$$||\nabla e||_K \leq \sqrt{\frac{|K|}{|\Omega|}} htol \tag{4.3}$$

The a priori estimation in the energy norm converges with the rate $O(h^{q-1})$ [20]. For each triangle in the mesh $\mathcal{T}_h$ with error estimate η_K, to achieve the condition shown in Eq. 4.4 we can predict the level of uniform refinement to apply on the triangle.

$$\eta_K \leq \sqrt{\frac{|K|}{|\Omega|}} htol \tag{4.4}$$

And the level of refinement is

$$reflvl = (int)\frac{log_2(\sqrt{\frac{|\Omega|}{|K|}}\frac{\eta_K}{htol})}{log_2(2^{\frac{1}{r-1}})} \tag{4.5}$$

The complete adaptive process stops when $\eta_{\mathcal{T}} \leq htol$ or $S = \emptyset$.

$S = \emptyset$;
foreach $K \in \mathcal{T}_h$ **do**
  $diff = \sqrt{\frac{|\Omega|}{|K|}}\frac{\eta_K}{htol}$;
  if $diff \geq 1$ **then**
    $reflvl = (int)log_2(diff^{\frac{1}{r-1}})$;
    if $reflvl = 0$ **then**
      $reflvl = 1$;
    end
    Mark $K(reflvl)$, $S = S + K$;
  end
end
Output: S

Algorithm 4: Drastic cutting marking algorithm

4.2 Accumulate SER Algorithm

As mentioned above, an adaptive algorithm is basically an iterative algorithm in the form: $Solve \Rightarrow Estimate \Rightarrow Mark \Rightarrow Refine$. For a PDE problem, the linear system is sparse symmetric positive and definite. Iterative methods such as conjugate gradient or preconditioned conjugate gradient are popular choices for solving the system at each adaptive iteration. Usually the initial guess for an iterative solver is a *zero* vector. Instead of using a *zero* vector, we take the result vector from previous iteration, embed it onto finer space of the current iteration, and use it as our initial guess vector for the iterative linear solver. We also can set the stopping condition for the iterative solver according to tolerance of the adaptive method. The *Accumulate SER Algorithm* is shown in Algorithm(5).

Let $\mathcal{T}_0$ be initial triangulation ;
$tol = 10^{-12}$;
v_0 = IterSolve($\mathcal{T}_0$, 0, tol) ;
η_0 = ErrorEst($\mathcal{T}_0$, v_0) ;
$i = 0$;
while $\eta_i < htol$ **do**
    $\mathcal{T}_{i+1}$ = MarkRefine($\mathcal{T}_i$, v_i, η_i) ;
    x^0_{i+1} = Embed(v_i, $\mathcal{T}_i$, $\mathcal{T}_{i+1}$) ;
    v_{i+1} = IterSolve($\mathcal{T}_i$, x^0_{i+1}, $htol$) ;
    η_{i+1} = ErrorEst($\mathcal{T}_{i+1}$, v_{i+1}) ;
    $i = i+1$;
end
Output: v

Algorithm 5: Accumulate SER algorithm

4.3 Experiments

Now let us show that the proposed drastic cutting marking strategy, compared with the Dörfler marking algorithm, speeds up the overall adaptive algorithm. The test problems are smooth, oscillatory, and singular respectively. For each experiment, we show our results in a table, which has entries: *CPUTime* is the wall clock time of whole adaptive process; *Computetime* is the time spent on solving the system at all iterative steps; *est* is the calculated error estimate; $||e||$ and $||\nabla e||$ are errors comparing with real solution; $|\mathcal{T}_h|$ is the number of triangles at the end of process; dof is number of degrees of freedom at the end of process; *SER iter* is total number of adaptive iterations; *Lvl* is the deepest level of mesh that the adaptive process reached. We also show the final mesh that the adaptive method generates. All the experiments were carried out on a machine with 2.66GHz Intel® Xeon™Quadcore CPUs with 4MB of L2 cache.

4.3.1 Smooth Solution Problem

Domain Ω: Figure 4.1

$$\begin{aligned} -\Delta u &= 2\pi^2 \sin(\pi x)\sin(\pi y) \quad \text{in } \Omega \\ u &= 0 \qquad\qquad\qquad\quad \text{on } \Gamma_D \end{aligned} \tag{4.6}$$

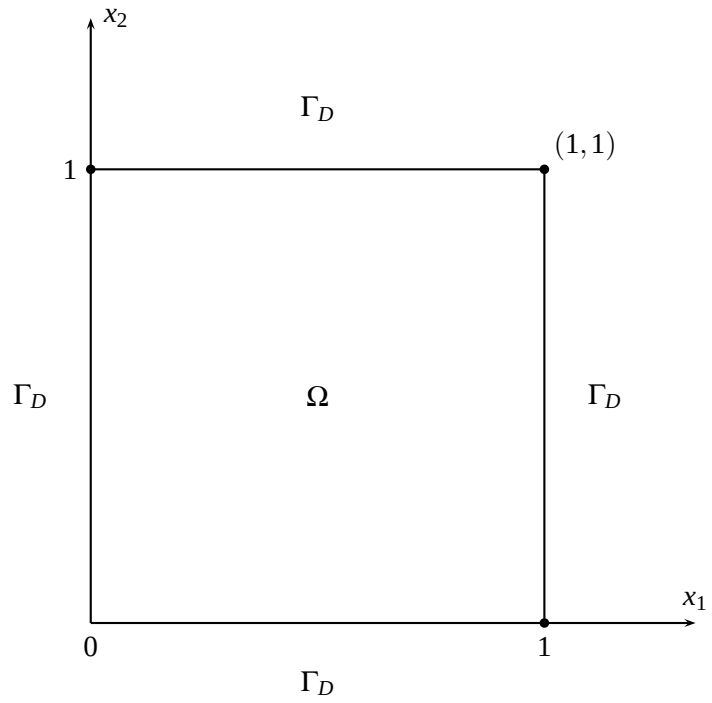


Figure 4.1: Square Domain

Exact solution: $u = \sin(\pi x)\sin(\pi y)$.

The problem has homogeneous Dirichlet boundary conditions and the solution is solely driven by the forcing function. The domain is a 1×1 box. Initial triangulation is done with maximum area constraint $a = 0.1$.

For the $r = 2$ case, we set $\gamma = 5$ and the target adaptive tolerance is $htol = 4.5e-2$. See Table 4.1, Figure 4.2 and Figure 4.3.

4.3.2 Oscillatory Solution Problem

Domain Ω: Figure 4.1

$$\begin{aligned} -\Delta u &= 128\pi^2 \sin(8\pi x)\sin(8\pi y) \quad && \text{in } \Omega \\ u &= 0 && \text{on } \Gamma_D \end{aligned} \tag{4.7}$$

Exact solution: $u = \sin(8\pi x)\sin(8\pi y)$.

This problem has a smooth non-polynomial solution which is oscillatory across the domain. The

Table 4.1: Comparison of Marking Algorithm: Problem4.6, r=2

	$\theta = 0.5$	$\theta = 0.7$	$\theta = 0.9$	DC
CPU time(sec)	7.43	5.80	4.45	3.19
Compute time(sec)	0.321	0.255	0.196	0.297
est	4.4352e-02	4.0369e-02	4.2811e-02	4.4670e-02
$\|\|e\|\|$	2.1115e-3	2.1095e-3	1.5598e-3	8.5283e-3
$\|\|\nabla e\|\|$	0.0230	0.0210	0.0213	0.0474
$\|\mathcal{T}_h\|$	37552	43894	40234	39469
dof	112656	131682	120702	118407
SER iter	33	17	9	5
Lvl	6	6	6	6

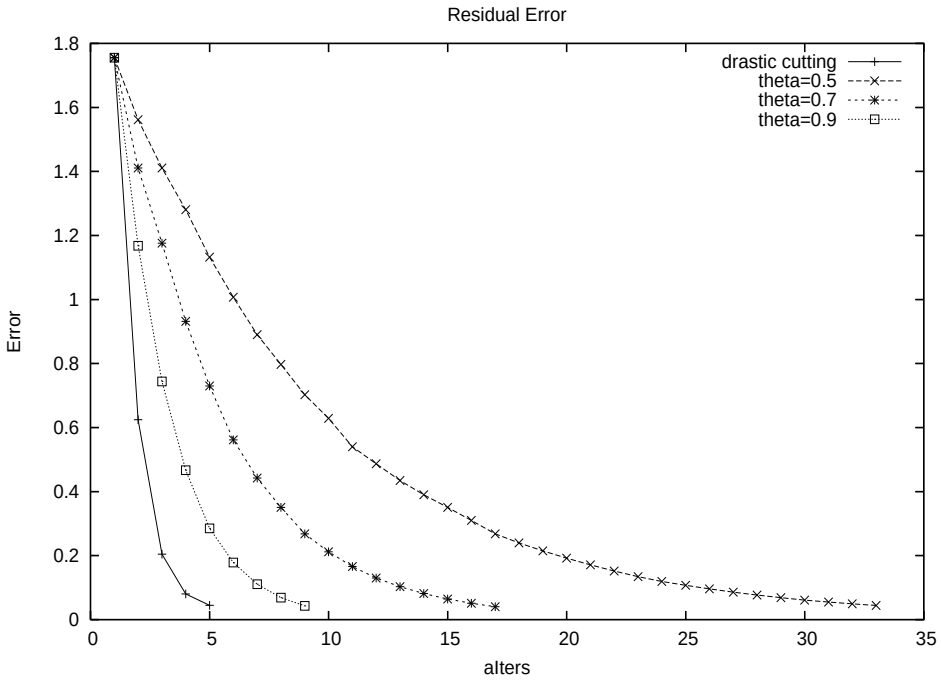
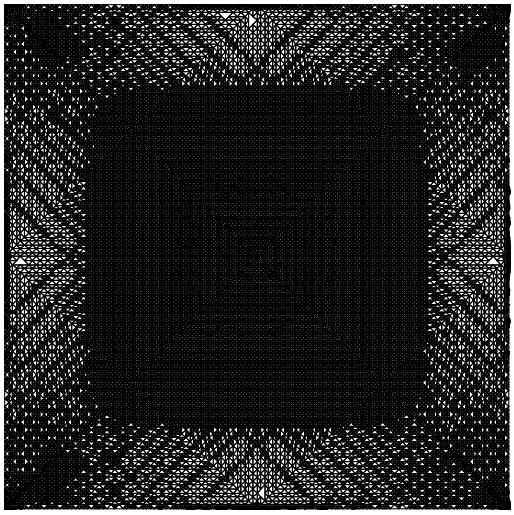
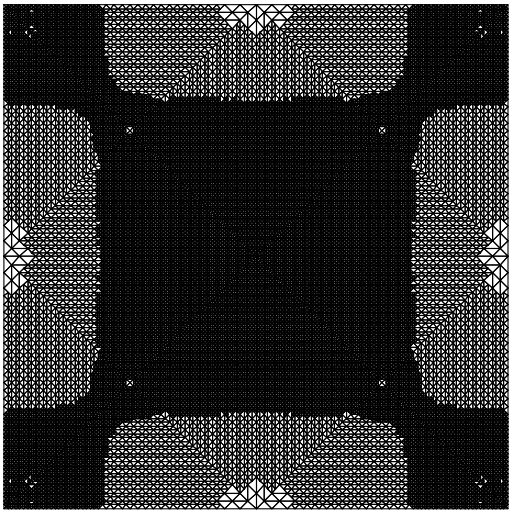


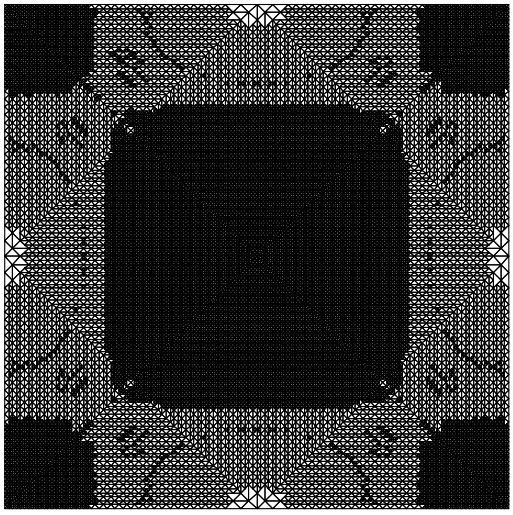
Figure 4.2: Residual Error: Problem4.6, r=2



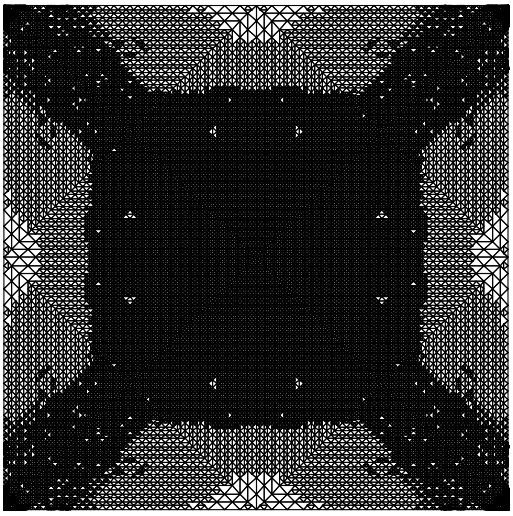
(a) $\theta = 0.5$, $r = 2$



(b) $\theta = 0.7$, $r = 2$



(c) $\theta = 0.9$, $r = 2$



(d) Drastic Cutting, $r = 2$

Figure 4.3: Adaptive Meshes: Problem4.6, r=2

problem has homogeneous Dirichlet boundary conditions and the solution is solely driven by the forcing function. For the $r = 3$ case, we set $\gamma = 4$ and the target adaptive tolerance is $htol = 0.25$. See Table 4.2, Figure 4.4 and Figure 4.5.

4.3.3 Singular Solution Problem

Domain Ω: Figure 4.6

$$\begin{aligned} -\Delta u &= 0 && \text{in } \Omega \\ u &= r^{2/3}\sin(2/3\theta) && \text{on } \Gamma_D \end{aligned} \tag{4.8}$$

Exact solution: $u = r^{2/3}\sin(2/3\phi)$.

This problem has a point singularity in the first derivative at the origin. This problem really stresses how well the adaptive algorithms work. Note also that the solution to this problem is solely driven by the trace of the solution on the boundary. For the $r = 2$ case, we set $\gamma = 7$ and the target adaptive tolerance is $htol = 0.01$. See Table 4.3, Figure 4.7 and Figure 4.8.

4.3.4 Comparison with DGADPT

DGADPT is an adaptive discontinuous galerkin finite element software package written by Michael Saum [30]. The following is the list of results of performance comparisons between our implementation(ASER) and DGADPT which shows that ASER is two to four times faster than DGADPT, see Figure 4.9. These tests were made on a 2.13GHz Intel® Core™2 Duo with 4MB of L2 cache.

- Smooth Solution Problem(Eqn. 4.6): $r = 2$, $htol = 0.03$, $\gamma = 25$;
- Oscillatory Solution Problem(Eqn. 4.7): $r = 3$, $htol = 0.1$, $\gamma = 25$;
- Singular Solution Problem(Eqn. 4.8): $r = 2$, $htol = 0.01$, $\gamma = 25$;

Table 4.2: Comparison of Marking Algorithm: Problem4.7, r=3

	$\theta = 0.5$	$\theta = 0.7$	$\theta = 0.9$	DC
CPU time(sec)	11.6	8.05	7.30	5.39
Compute time(sec)	0.723	0.45	0.391	0.535
est	2.3617e-01	2.3040e-1	1.9306e-1	2.5068e-1
$\|\|e\|\|$	5.2401e-3	3.7737e-3	5.5286e-3	1.2549e-2
$\|\|\nabla e\|\|$	0.0954	0.117	0.157	0.556
$\|\mathcal{T}_h\|$	31663	33190	39580	47938
dof	189978	199140	237480	287628
SER iter	40	19	10	6
Lvl	6	6	6	8

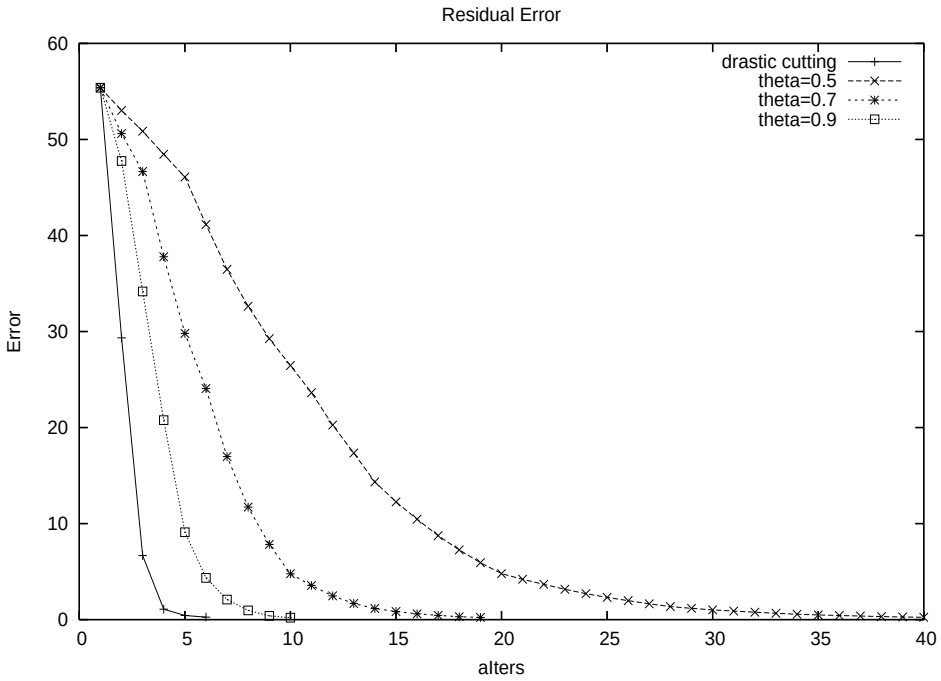
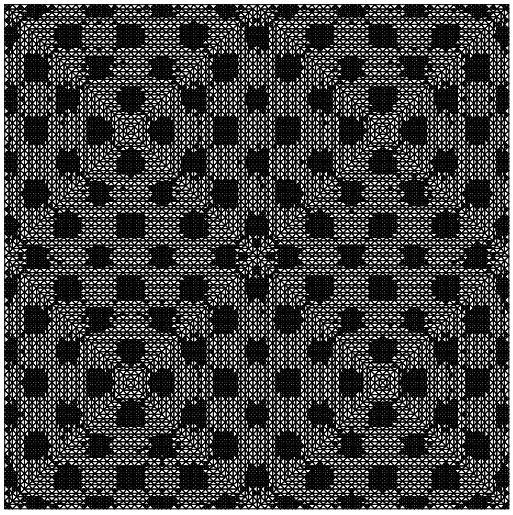
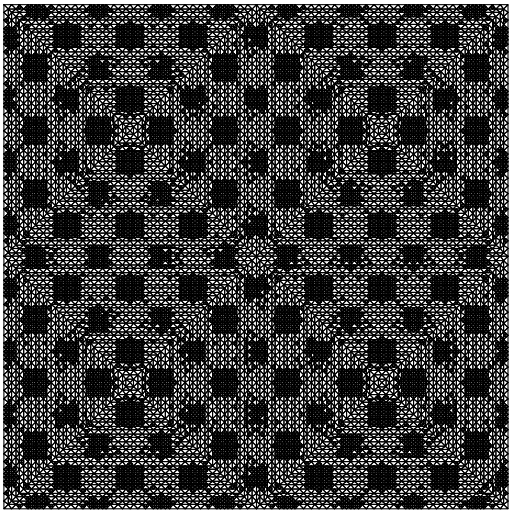


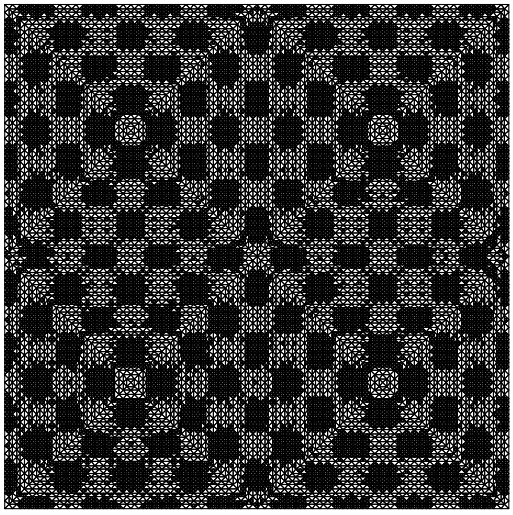
Figure 4.4: Residual Error: Problem4.7, r=3



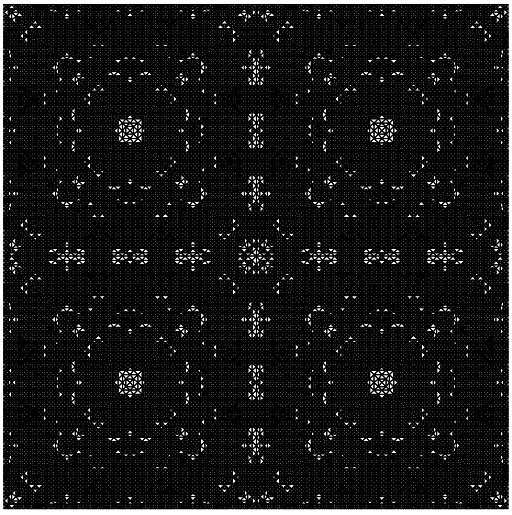
(a) $\theta = 0.5$, $r = 3$



(b) $\theta = 0.7$, $r = 3$



(c) $\theta = 0.9$, $r = 3$



(d) Drastic Cutting, $r = 3$

Figure 4.5: Adaptive Meshes: Problem4.7, r=3

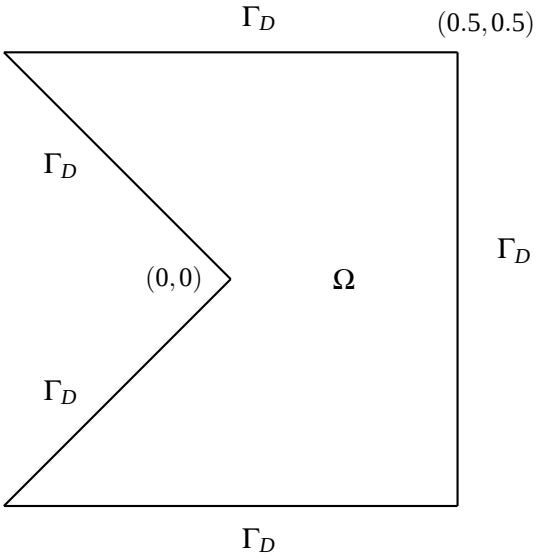


Figure 4.6: Notch Domain

Table 4.3: Comparison of Marking Algorithm: Problem4.8, r=2

	$\theta = 0.5$	$\theta = 0.7$	$\theta = 0.9$	DC
CPU time(sec)	4.82	4.35	3.58	3.54
Compute time(sec)	0.42	0.4	0.2	0.38
est	1.0064e-02	8.6360e-03	9.6333e-03	9.8907e-03
$\|\|e\|\|$	2.6660e-4	3.8521e-4	1.9695e-4	3.4549e-4
$\|\|\nabla e\|\|$	5.37e-3	5.03e-3	5.07e-3	5.65e-3
$\|\mathcal{T}_h\|$	17727	23694	24804	23211
dof	53181	71082	74412	69633
SER iter	28	16	10	5
Lvl	15	15	9	12

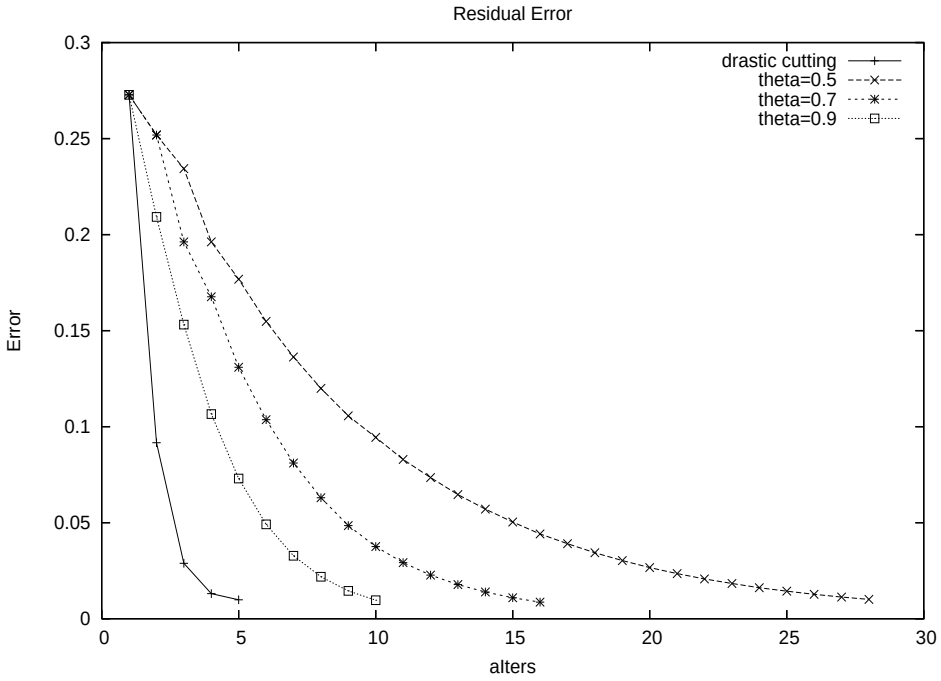
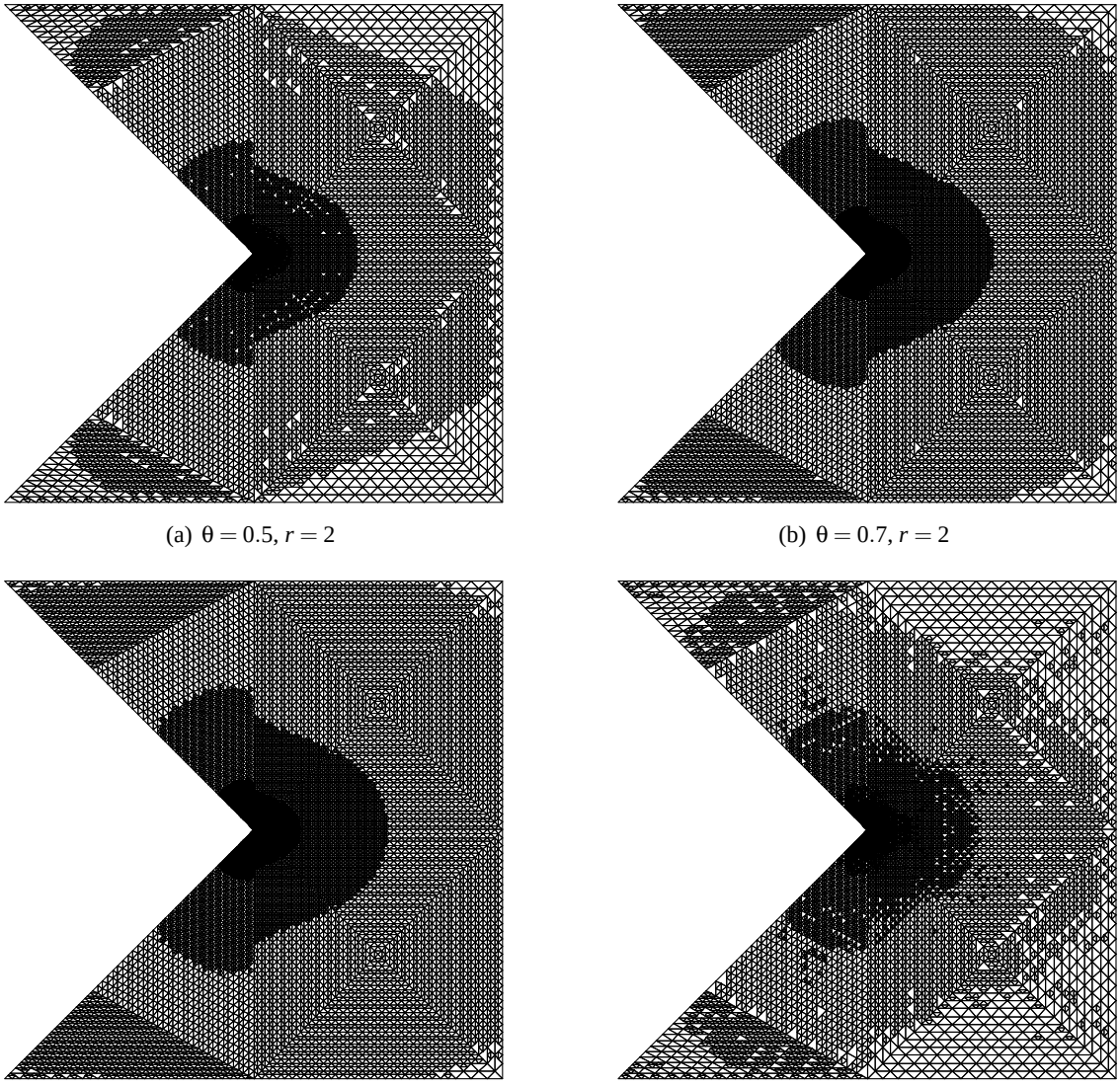


Figure 4.7: Residual Error: Problem4.8, r=2



(a) $\theta = 0.5$, $r = 2$

(b) $\theta = 0.7$, $r = 2$

(c) $\theta = 0.9$, $r = 2$

(d) Drastic Cutting, $r = 2$

Figure 4.8: Adaptive Meshes: Problem4.8, r=2

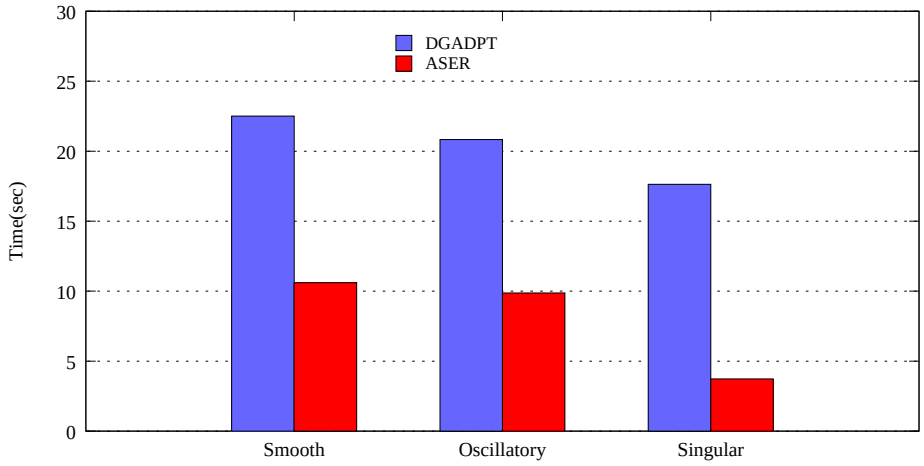


Figure 4.9: Performance comparison of DGADPT and ASER

4.4 Discussion

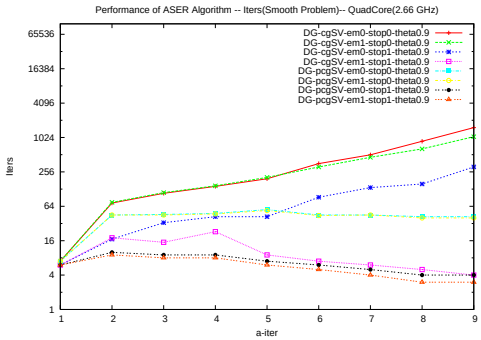
We can see that the Accumulate SER Algorithm does a pretty good job in terms of performance, see Figure 4.10 to Figure 4.15. Especially the application of adaptive tolerance to the iterative solver dramatically lowers the number of iterations.

- The adaptive algorithm intends to lower the size of problem at each adaptive iteration, so the *pcg* iterative solver does not show much performance improvement compared with the *cg* iterative solver. The reason is that *pcg* pays the penalty as it does the preconditioning including solving coarse mesh correction, a series of subdomains, and embedding and projection operations.

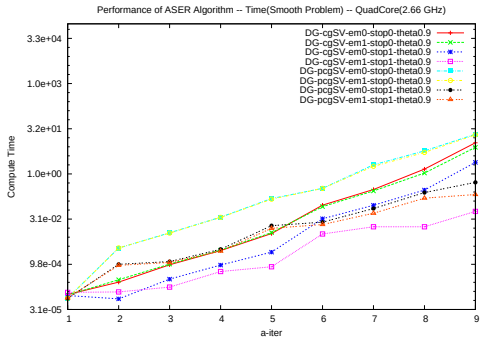
- When we have to solve the problem at a very fine level, the *pcg* iterative solver is still a better choice since the savings in the number of iterations will offset the penalty. Each iteration of the iterative solver costs a lot more when the linear system's size increases.

- Embedding the result from the previous adaptive iteration is not as effective as tailoring the iterative solver with adaptive tolerance $htol$. Performance is even better when we combine both together. ASER lowers the number of iterations of the *cg* method so dramatically that it is comparable with *pcg*. The performance of ASER with *cg* method is overall the best.

- ASER with Drastic cutting outperforms the fixed θ marking approach. As we put a lot of effort into optimizing the algorithm and code for fast solver at each adaptive level, the mesh refinement and memory management is becoming the most time-consuming part. From Figure 4.2, 4.4 and 4.7, we can see that drastic cutting dramatically lowers the number of adaptive iterations, saving time on mesh generation, but it takes more time for computing the solution as the previous result is not that close to the solution at the current level.

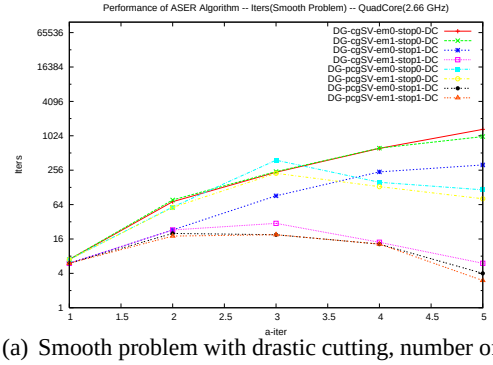


(a) Smooth problem with $\theta = 0.9$, number of iterations at each adaptive step

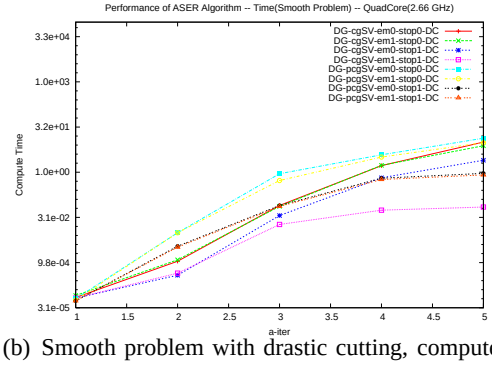


(b) Smooth problem with $\theta = 0.9$, compute time at each adaptive step

Figure 4.10: Smooth problem: ASER algorithm

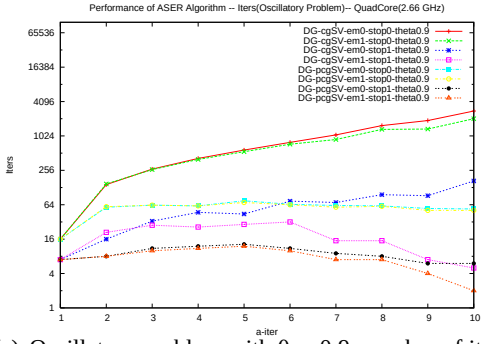


(a) Smooth problem with drastic cutting, number of iterations at each adaptive step

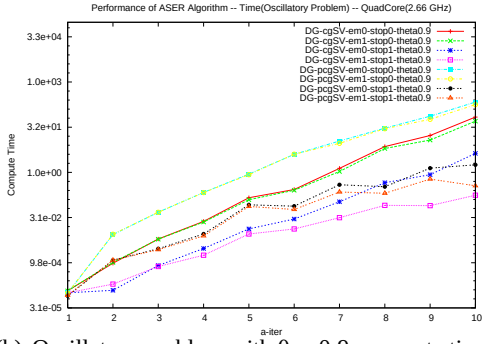


(b) Smooth problem with drastic cutting, compute time at each adaptive step

Figure 4.11: Smooth problem: ASER algorithm with drastic cutting

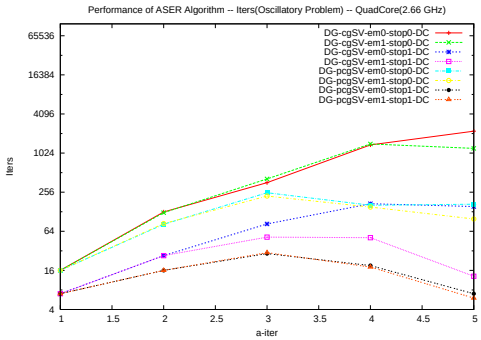


(a) Oscillatory problem with $\theta = 0.9$, number of iterations at each adaptive step

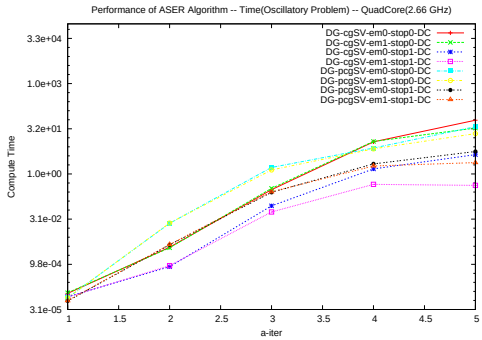


(b) Oscillatory problem with $\theta = 0.9$, compute time at each adaptive step

Figure 4.12: Oscillatory problem: ASER algorithm

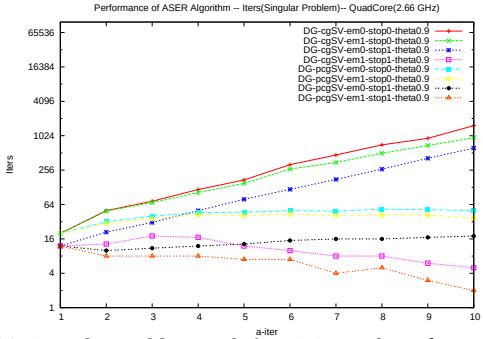


(a) Oscillatory problem with drastic cutting, number of iterations at each adaptive step

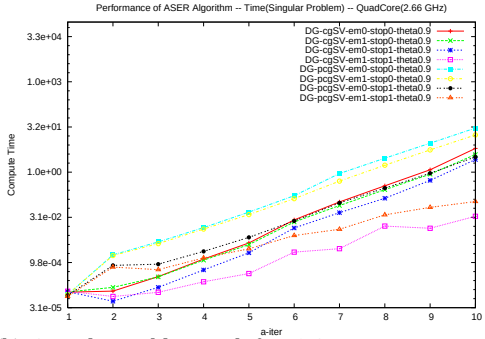


(b) Oscillatory problem with drastic cutting, compute time at each adaptive step

Figure 4.13: Oscillatory problem: ASER algorithm with drastic cutting

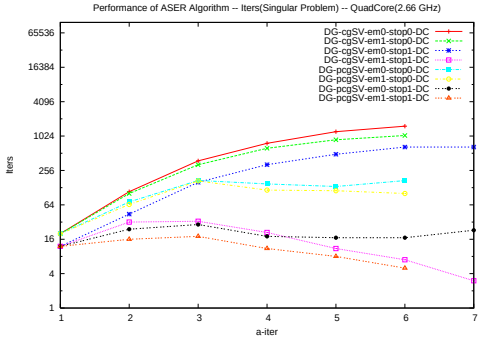


(a) Singular problem with $\theta = 0.9$, number of iterations at each adaptive step

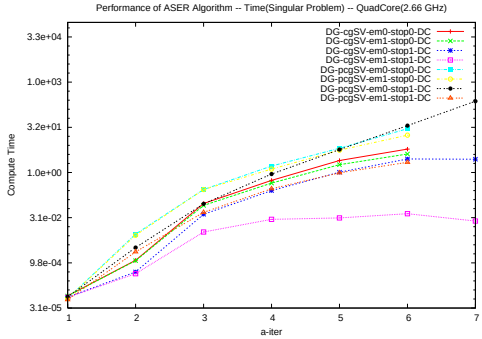


(b) Singular problem with $\theta = 0.9$, compute time at each adaptive step

Figure 4.14: Singular problem: ASER algorithm



(a) Singular problem with drastic cutting, number of iterations at each adaptive step



(b) Singular problem with drastic cutting, compute time at each adaptive step

Figure 4.15: Singular problem: ASER algorithm with drastic cutting

Chapter 5

Implementation and Data Structure

5.1 Introduction

This chapter describes the design and implementation of Adaptive Finite Element Method software. The program is written in the C programming language. Our main goal of this research is to develop software for solving PDE problem as fast as possible on a wide variety of platforms. The modular design allows us to test different algorithms and implementations as a research project, as well as maintain the code stability. Figure 5.1 shows the diagram of the software.

The Mesh Generator component takes a geometry description input file, generates meshes with vertices, edges, and triangles, and stores information in a database. It also calculates a numerical data block for each geometric object, which is used to build the linear system for solution. At initialization stage, it utilizes the triangulation software: Triangle [32], which is a Two-Dimensional quality mesh generator and Delaunay triangulator. This component is invoked at every iterative step. The Linear System Generator component goes through elements in the current mesh, gathers numerical data blocks and assembles the linear system. The Reordering tool module reduces the bandwidth of sparse symmetric matrices. The Domain decomposition module generates submatrices used as the preconditioner. The Solver component solves a linear system with conjugate gradient method or preconditioned conjugate gradient method. The Adaptive Error Estimator and Refinement modules take a solution and calculate the error. If the error is less than the termination

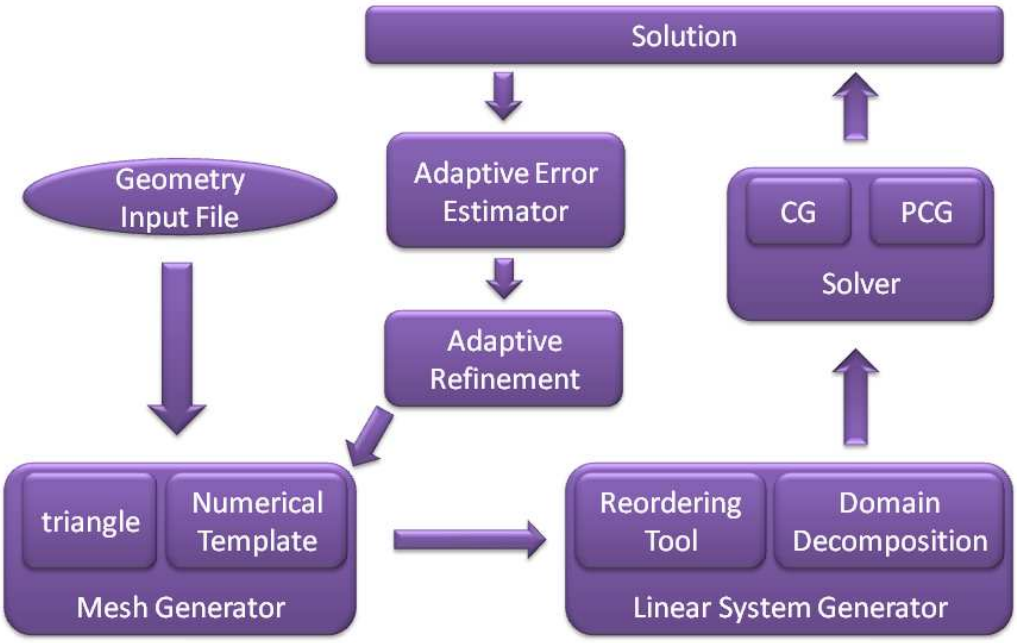


Figure 5.1: Diagram of ADFEM software.

condition, these modules output the solution; otherwise the mark elements to be refined and feed them to the Mesh Generator.

In the rest of the chapter, we describe basic data structures and components of the software.

5.2 Data Structure

Given a 2D domain $\Omega \subset \mathbf{R^2}$, we can have a quasi uniform triangular mesh that covers Ω. The mesh consists of a set of non-overlapped triangles, and basic components are vertex, edge, and triangles.

5.2.1 Vertex

Structure NODE_T, see Figure 5.2, is the data structure we defined for a vertex. *x* and *y* are coordinates of the vertex. We also assign each vertex a unique *id* for sorting and searching purpose. Each

```
typedef struct nodestruct {
  double x;
  double y;
  unsigned int lvl;
  unsigned int id;
  short bm;
  double value;
} NODE_T;
```

Figure 5.2: Node structure

time the mesh is refined new vertices are generated that will be part of future mesh too. We use *lvl* to indicate the level on which each vertex is created. *value* is to record the solution of the PDE.

5.2.2 Edge

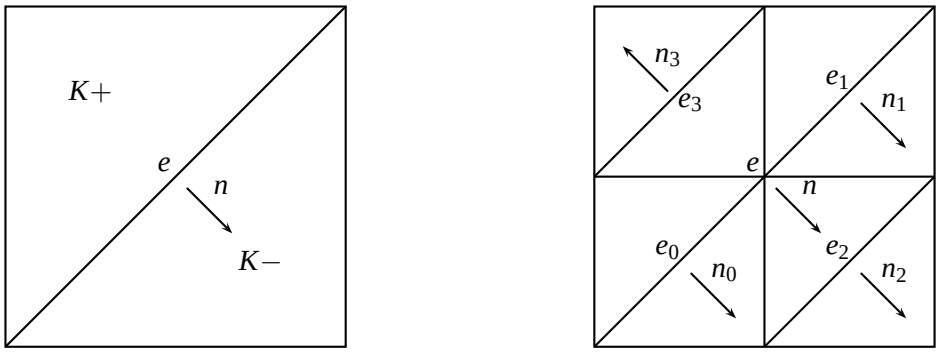
Structure EDGE_T, see Figure 5.3, is the data structure we defined for edge. There are two types of edges: internal and boundary. Field *type* indicates that the edge is internal or boundary, and also Dirichlet or Neumann boundary. At initialization time we calculate the length of an edge and its normal, which is stored in structure EDGEDATAG_T, see Figure 5.4 . Field $info$ point to the EDGEDATAG_T when *lvl* is 0, otherwise it points to EDGE_T at level 0. Figure 5.5(b) shows that e_0, e_1, e_2 and e_3 from one level finer mesh are half length of e, and maintain the same normal except that the normal of e_3 has opposite direction, as indicated by field *sign*. We use *lvl* to indicate the level on which each edge is created. The actual length of an edge on level *lvl* is $1/2^{lvl}$ times the length of its original ancestor edge in the initial mesh. For a given mesh, Figure 5.5(a) shows that an internal edge is an edge of two triangles: $K+$ and $K-$. $K+$ is the triangle that edge normal point outward, and $K-$ is the triangle that edge normal point toward. Note that $K-$ is NULL if the edge on the boundary. $Kploc$ and $Kmloc$ indicates the position of the edge on triangles $K+$ and $K-$ (see Figure 5.6(a)).

```
typedef struct edgestruct {
  void *info;
  struct tristruct *Kplus, *Kminus;
  NODE_T *endp[2];
  NODE_T *midpt;
  unsigned Kploc : 4;
  unsigned Kmloc : 4;
  unsigned type : 2;
  unsigned leaf : 1;
  unsigned new : 1;
  unsigned dedge : 1;
  unsigned : 3;
  unsigned int bm : 6;
  unsigned int bidx : 24;
  unsigned int : 2;
  unsigned int lvl;
  short refine, sign;
  unsigned int id, idx;
  struct edgestruct *child, *parent, *prev, *next;
  double *off;
} EDGE_T;
```

Figure 5.3: Edge structure

```
typedef struct edggeomdata {
  double edgelen;
  double norml[2];
} EDGDATAG_T;
```

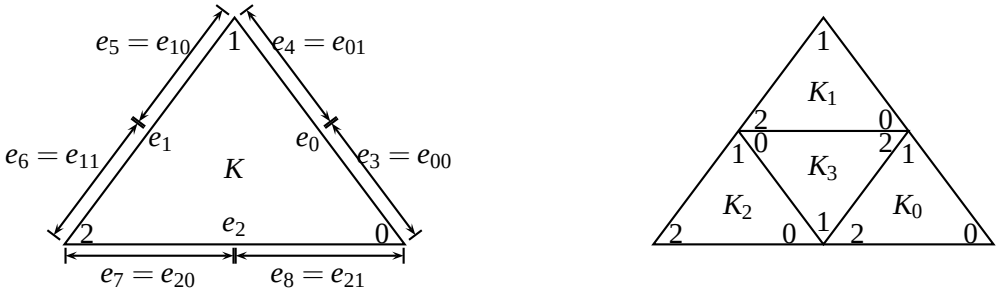
Figure 5.4: Edgedata structure



(a) Edge e and its normal

(b) e_0, e_1, e_2, e_3

Figure 5.5: Edge and child edges in fine mesh.



(a) K

(b) $\{K_0, K_1, K_2, K_3\}$

Figure 5.6: Local Ordering for Triangle K,$\{K_0, K_1, K_2, K_3\}$

5.2.3 Triangle

Figure 5.7, is the data structure we defined for triangle. At initialization time we calculate the area of a triangle and its affine transformation matrix, which is stored in structure TRIDATAG_T, see Figure 5.8 . Field $info$ points to the TRIDATAG_T when lvl is 0, otherwise it points to TRIANGLE_T at level 0. Figure 5.6(b) shows that K_0, K_1, K_2 and K_3 from one level finer mesh are a quarter of area of K in Figure 5.6(a), and maintain the same affine transformation matrix except that K_3 has opposite direction, which is indicated by field *sign*. TRIANGLE_T maintains pointers to 3 vertices and 3 edges of a triangle. lvl indicates the level on which the triangle is created. The actual area of a triangle on level lvl is $1/4^{lvl}$ times the area of its original ancestor triangle in the initial mesh.

5.2.4 PDE Data

Each interior edge has an off-diagonal matrix block that describes the interaction between elements along the edge. Each triangle structure maintains a symmetric positive stiffness matrix block that describes the interactions of degrees of freedom (dof) of the element. These data blocks are N by N matrices. The third type of data object is simply one or more vectors of length n (dof) associated with each element, used to maintain the solution obtained during the solve process and element right hand side (RHS) vectors.

5.2.5 Mesh

Given a domain description we have an initial mesh with vertices, edges and triangles. As showed in Figure 5.9, we build up a tree to store every element(triangle). Each triangle in the initial mesh is a root node of the tree, and they are also put into link list for quick access. Each triangle is refined by regular subdivision which creates four children triangles by connecting the midpoints of the three edges of the parent triangle . All children triangles are similar to the parent triangle (see Figure 5.6(b)). Every child triangle has a link to its parent, and parent links to K_0. There is also a link list of child triangles. We also create a tree for edges since edges has the same hierarchical structure as triangles (see Figure 5.5(b)). In edge and triangle structures, Figure 5.3 and 5.7, pointers: *child*,

```
typedef struct tristruct {
  void *info;
  EDGE_T *edges[3];
  NODE_T *corners[3];
  unsigned int lvl;
  unsigned leaf : 1;
  unsigned recalc : 1;
  unsigned nbrstate : 1;
  unsigned new : 1;
  unsigned cid : 2;
  unsigned bm  : 2;
  unsigned refine: 6;
  unsigned visited: 2;
  short sign;
  unsigned int id, idx, didx, lidx;
  struct tristruct *child, *parent, *prev, *next;
  double *sd[2], *rhs, *x, estp;
} TRIANGLE_T;
```

Figure 5.7: Triangle structure

```
typedef struct trigeomdata {
  double area2;
  double atrf[2][2];
} TRIDATAG_T;
```

Figure 5.8: Tridata structure

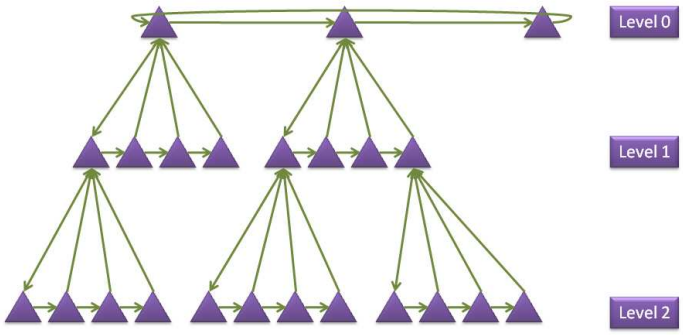
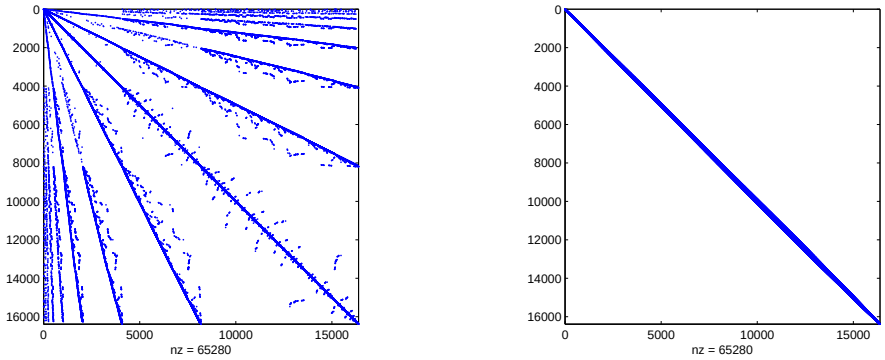


Figure 5.9: Tree

parent, *prev* and *next*, are used to build the tree system. Although the mesh data structure keeps a record of all geometric data structures, we only generate PDE data block , matrices and vectors as needed to solve the problem at certain level. In practice, a dynamic mesh structure is created by gathering all the leaves from the edge and triangle tree, and matrix(LHS) and vector(RHS) are assembled.

5.3 Reordering

The variational formulation of a PDE problem produces a large sparse symmetric positive definite linear system that needs to be solved at each adaptive iteration. The sparsity and symmetry of the linear system make it an easy choice of an iterative solver such as conjugate gradient method and preconditioned conjugate gradient method [6]. The performance of these iterative methods is bounded by sparse matrix vector multiplication. Figure 5.10(a) shows that each non-zero entry scatters everywhere, causing bad data spatial locality. As a result, the number of cache misses is high and performance suffers. Reordering is the treatment we apply to reduce the bandwidth of the matrix; consequently the data locality is greatly improved (see Figure 5.10(b)). Proposed in 1969, Cuthill-McKee algorithm is a popular and the simplest method for reducing bandwidth of sparse



(a) Sparse matrix (b) Reordered sparse matrix

Figure 5.10: Comparison of non-reordered and reordered sparse matrices

Input: Choose a peripheral vertex x and set $R := x$.

while $i < |R|$ *and* $|R| < N$ **do**

  Construct the adjacency set A_i of R_i, where R_i is the i-th component of R, and exclude the vertices we already have in R ;

  $A_i := \mathrm{Adj}(\mathrm{R_i}) \setminus R$;

  Sort A_i with ascending vertex order ;

  Append A_i to the set R ;

end

Output: R

Algorithm 6: Cuthill-McKee Algorithm

matrices. A mesh can be visualized as a graph with each triangle as a vertex and each edge as a linkage. The Cuthill-McKee algorithm, described in Algorithm(6), is a reindexing process of the vertices of the graph, and it reduces the bandwidth of the corresponding matrix. Figure 5.11 shows that performance of the conjugate gradient method with a reordered matrix is about 10% better than with the non-reordered matrix. There is a total of 16384 triangles in the mesh. The bandwidth is reduced from 12288 to 129 after reordering.

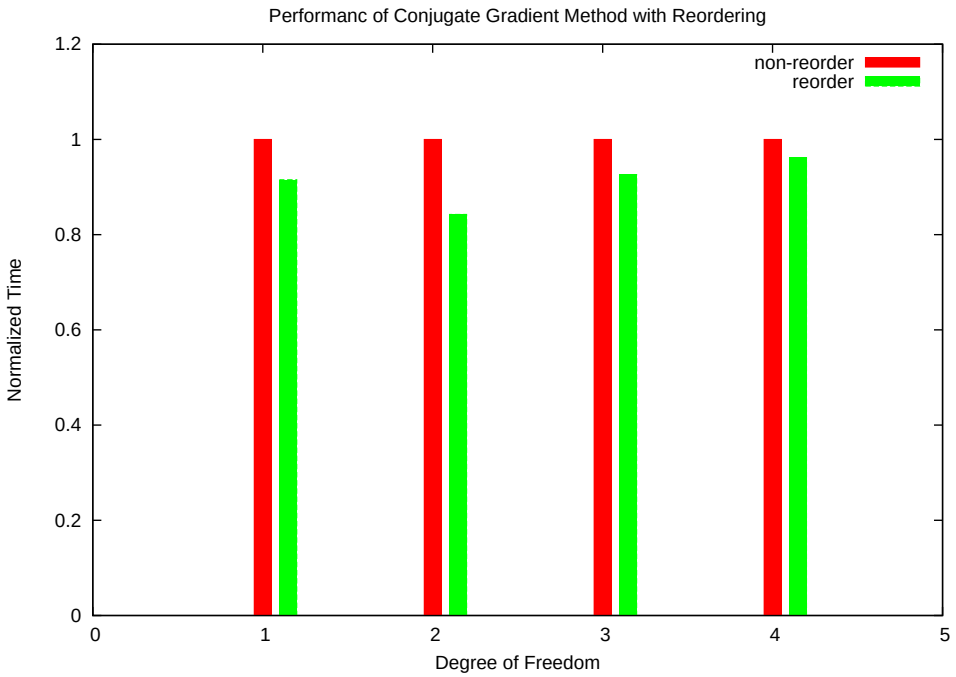


Figure 5.11: Performance comparison of conjugate gradient method with non-reordered and reordered sparse matrices

5.4 Embedding and Projection Operators

The PCG method, specifically the preconditioned conjugate gradient method described in Algorithm (7), requires coarse mesh corrections by solving the error problem $Ae = r$. As we transfer data(vector) between finite element spaces, there are two different types of operators: 1) Embedding operator, also called interpolation operator, is defined as translating a vector from a lower dimensional subspace into a higher dimensional subspace; 2) Projection operator, also called restriction operator, is defined as translating a vector from a higher dimensional subspace into a lower dimensional subspace. Let I_H^h denote the embedding operator and I_h^H denote the projection operator, where $I_h^H = {I_H^h}^T$. We define two embedding operators in the software. The first one works within discontinuous subspaces at the level of coarse and fine meshes; we call it DG Embedding Operator. The second one works between continuous and discontinuous subspaces at the same level of mesh; we call it CDG Embedding Operator.

5.4.1 DG Embedding Operator

Let $\{x_i : i = 0,...N-1\}$ denote the (x,y) coordinates of the N degrees of freedom on the reference element $\hat{K}$ and $\{\phi_i : i = 0,...N-1\}$ denote the corresponding basis functions on the reference element $\hat{K}$. Let $\{x_{k,j} : k = 0,...3; j = 0,...N-1\}$ denote the (x,y) coordinates of the N degrees of freedom on the four children of $\hat{K}$, $\{\hat{K}_0, \hat{K}_1, \hat{K}_2, \hat{K}_3\}$, see Figure 5.6. The embedding process is the interpolation of values of basis functions on $\hat{K}$ to children of $\hat{K}$. Eq(5.1) defines the matrix of embedding operator. For example, with degree of one basis functions we can calculate the embedding operator, see Eq(5.2). We can see that I_H^h is a $4N \times N$ matrix with each $N \times N$ block corresponding to a child triangle. It maps a vector $u \in \mathbb{R}^N$ into a vector $v \in \mathbb{R}^{4N}$.

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$;
$\mathbf{z}_0 := \mathbf{M}^{-1}\mathbf{r}_0$;
$\mathbf{p}_0 := \mathbf{z}_0$;
$k := 0$;
while *true* **do**
  $\alpha_k := \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{z}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$;
  $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$;
  $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$;
  if $|r_{k+1}| \leq \varepsilon$ **then**
    exit
  end
  $\mathbf{z}_{k+1} := \mathbf{M}^{-1}\mathbf{r}_{k+1}$;
  $\beta_k := \frac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{z}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{z}_k}$;
  $\mathbf{p}_{k+1} := \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$;
  $k := k+1$;
end
Output: x_{k+1}

Algorithm 7: Preconditioned Conjugate Gradient Method

$$I_H^h = \begin{bmatrix} \phi_0(x_{0,0}) & \phi_0(x_{0,1}) & \cdots & \phi_0(x_{0,N-1}) \\ \phi_1(x_{0,0}) & \phi_1(x_{0,1}) & \cdots & \phi_1(x_{0,N-1}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N-1}(x_{0,0}) & \phi_{N-1}(x_{0,1}) & \cdots & \phi_{N-1}(x_{0,N-1}) \\ \phi_0(x_{1,0}) & \phi_0(x_{1,1}) & \cdots & \phi_0(x_{1,N-1}) \\ \phi_1(x_{1,0}) & \phi_1(x_{1,1}) & \cdots & \phi_1(x_{1,N-1}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N-1}(x_{1,0}) & \phi_{N-1}(x_{1,1}) & \cdots & \phi_{N-1}(x_{1,N-1}) \\ \phi_0(x_{2,0}) & \phi_0(x_{2,1}) & \cdots & \phi_0(x_{2,N-1}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N-1}(x_{2,0}) & \phi_{N-1}(x_{2,1}) & \cdots & \phi_{N-1}(x_{2,N-1}) \\ \phi_0(x_{3,0}) & \phi_0(x_{3,1}) & \cdots & \phi_0(x_{3,N-1}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N-1}(x_{3,0}) & \phi_{N-1}(x_{3,1}) & \cdots & \phi_{N-1}(x_{3,N-1}) \end{bmatrix} \tag{5.1}$$

$$I_{H1}^h = \begin{bmatrix} 1.00 & 0.00 & 0.00 \\ 0.50 & 0.50 & 0.00 \\ 0.50 & 0.00 & 0.50 \\ 0.50 & 0.50 & 0.00 \\ 0.00 & 1.00 & 0.00 \\ 0.00 & 0.50 & 0.50 \\ 0.50 & 0.00 & 0.50 \\ 0.00 & 0.50 & 0.50 \\ 0.00 & 0.00 & 1.00 \\ 0.00 & 0.50 & 0.50 \\ 0.50 & 0.00 & 0.50 \\ 0.50 & 0.50 & 0.00 \end{bmatrix} \tag{5.2}$$

5.4.2 CDG Embedding Operator

Continuous and Discontinuous Galerkin Methods are node-based and element-based methods respectively. Let $IDX_{cg} = \{n_i : i = 0,...L\}$ denote indexes of all the nodes in the mesh. Let $\{K_j : j = 0,...M\}$ denote all the elements, each element is a triangle in 2D case, $\{n_{j,k} \in IDX_{cg} : j = 0,...M; k = 0,...N\}$ denote indexes of each degree of freedom of a triangle (see Table 5.1). Thus we can build an embedding operator by putting 1 at row $j \times N + k$ and column $n_{j,k}$. It is a $MN \times L$ matrix, where M is number of triangles, N is number of degree of freedom of each triangle, and L is number of nodes on the mesh. I_H^h maps a vector $u \in \mathbb{R}^L$ into a vector $v \in \mathbb{R}^{MN}$. Given a 2D mesh showed in Figure 5.12 and degree of one basis functions, we can have a node mapping table, see Table 5.2. In this example, we build a CDG embedding operator I_{H1}^h, Eq(5.3), which is a 24×9 sparse matrix with 1 for non-zeros.

Table 5.1: CDG node mapping

	0	1	$\cdots$	N
K_0	$n_{0,0}$	$n_{0,1}$	$\cdots$	$n_{0,N}$
K_1	$n_{1,0}$	$n_{1,1}$	$\cdots$	$n_{1,N}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
K_M	$n_{M,0}$	$n_{M,1}$	$\cdots$	$n_{M,N}$

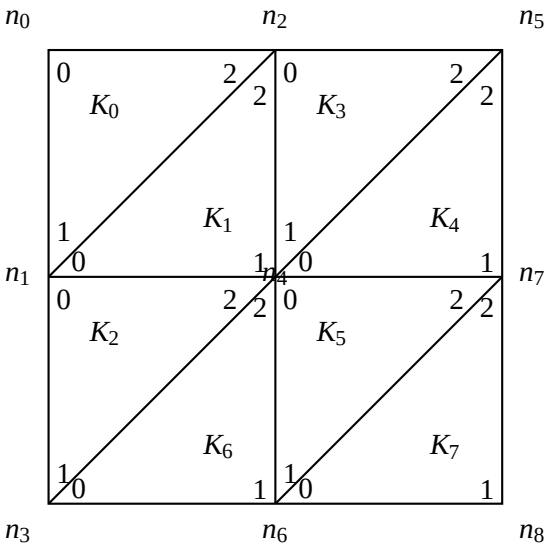


Figure 5.12: A example of CDG node mapping: 2D mesh

Table 5.2: A example of CDG node mapping: CDG node mapping

	0	1	2
K_0	n_0	n_1	n_2
K_1	n_1	n_4	n_2
K_2	n_1	n_3	n_4
K_3	n_2	n_4	n_5
K_4	n_4	n_7	n_5
K_5	n_4	n_6	n_7
K_6	n_3	n_6	n_4
K_7	n_6	n_8	n_7

$$
I_{H1}^{h} = \begin{bmatrix}
1 & & & & & & & & \\
 & 1 & & & & & & & \\
 & & 1 & & & & & & \\
 & 1 & & & & & & & \\
 & & & & 1 & & & & \\
 & & 1 & & & & & & \\
 & 1 & & & & & & & \\
 & & & 1 & & & & & \\
 & & & & 1 & & & & \\
 & & 1 & & & & & & \\
 & & & & 1 & & & & \\
 & & & & & 1 & & & \\
 & & & & 1 & & & & \\
 & & & & & & & 1 & \\
 & & & & & 1 & & & \\
 & & & & 1 & & & & \\
 & & & & & & 1 & & \\
 & & & & & & & 1 & \\
 & & & 1 & & & & & \\
 & & & & & & 1 & & \\
 & & & & 1 & & & & \\
 & & & & & & 1 & & \\
 & & & & & & & & 1 \\
 & & & & & & & 1 &
\end{bmatrix} \tag{5.3}
$$

5.5 Summary

To summarize, we have designed and implemented a fast adaptive discontinuous Galerkin FEM software package that has approximately 60000 line of codes. It includes data structure, algorithm designs, dense and sparse linear algebra, multi-thread and MPI programming, performance analysis, etc.

Chapter 6

Parallel Implementation

6.1 Introduction

The recently released TOP500 list [25] of the world's fastest supercomputers depicts some important trends in the area of high performance computing: clusters represent the most common architecture and multi-core processors represent the dominant chip architecture. These trends have a big influence on research and development in high performance computing. To achieve high performance on such systems, the software has to be scalable on a distributed memory system with tens of thousands of CPUs, capable of on-chip parallelism that takes advantage of multi-core chip architecture using shared memory threading, and tuned to have better cache locality and enhanced instruction level parallelism. The Discontinuous Galerkin Method with a domain decomposition preconditioner shows its full potential at all these different levels of parallel optimization. Fist of all, it is natural to split the whole problem into small pieces by domain decomposition for distributed computing. This domain decomposition is relatively easy to carry out with the discontinuous scheme, since it does not require continuity along the boundary of elements. Secondly, each domain maintains a row of blocked sparse matrices, i.e. a diagonal block (the stiffness matrix of the domain), and a list of non-diagonal blocks (flux and penalty terms along boundaries of domain), and a list of small blocked sparse matrices (stiffness matrices of subdomains) if using domain decomposition as a preconditioner. Then on each node, the computational tasks for an

iterative method of solving a linear system are a list of sparse matrix vector products, which can be further parallelized on a multi-core architecture with multi-threaded programming.

Currently the Message Passing Interface(MPI) standard is the de-facto parallel computing standard. It is widely used by many scientific programs as their communication layer. To reach good performance on a massive parallel machine, it is important to design the software to overlap computation and communication. In the rest of this chapter, we show the parallel implementation of the discontinuous Galerkin finite element method with data structures, parallel iterative method, and its performance.

6.2 Parallel Design and Data Structure

6.2.1 Domain Decomposition

Domain decomposition is a popular technique to make use of parallel computers. Coupled with Message Passing Interface(MPI), it has become a widely used technique to design software for distributed memory architectures. This technique divides the whole computation into many smaller tasks. Each task contains a local computation part, which is a standalone computation without interprocessor communication, and another part communicates with its neighbors and distant processors for exchanging data. Domain decomposition, which we described in Chapter 3, is a method to divide the large linear system into smaller problems and produce a preconditioner to speed up solving the entire system. In this chapter, domain decomposition refers to data decomposition; see [13], [11], and [12] for more details.

6.2.2 METIS

"METIS is a family of programs for partitioning unstructured graphs and hypergraphs and for computing fill-reducing orderings of sparse matrices. The underlying algorithms used by METIS are based on a state-of-the-art multilevel paradigm that has been shown to produce high quality results and scale to very large problems." — http://glaros.dtc.umn.edu/gkhome/views/metis

For a given 2D domain $\Omega \subset \mathbf{R}^2$, we can generate a quasi uniform triangular mesh that covers Ω.

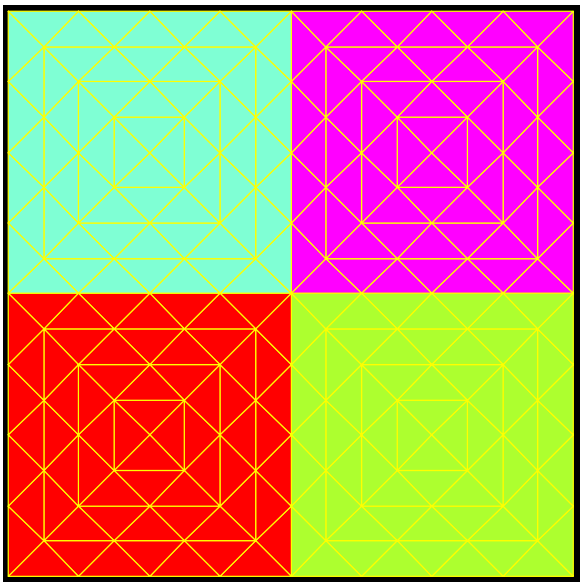


Figure 6.1: A mesh with 4 subdomains.

The mesh consists of a set of non-overlapped triangles, and basic components are vertex, edge, and triangles. We divide the whole domain into subdomains. Then a mesh consists of a set of non-overlapped subdomains, and a subdomain contains basic components. It is an intermediate level between mesh and basic components. In some sense we can treat a subdomain as a mesh object, simplifying the process of parallel implementation. METIS is freely available graph partitioning software that generates subdomains that satisfy our requirements. Figure 6.1 shows an example of four subdomains produced by METIS. See [22], [21] for more about METIS.

6.2.3 Parallel Iterative Solver

As we mentioned in previous chapters, Galerkin finite element methods produce a symmetric positive definite linear system. We use iterative solvers such as conjugate gradient and preconditioned conjugate gradient method for solving the linear system. It is straight-forward to parallelize the CG method by distributing the matrix and vectors and computing the vector operations and matrix-vector multiplication in parallel. We list the pseudo-code for conjugate gradient method in Algorithm 8.

We can group all computations into three categories:

Input: vector x can be an approximate initial solution or 0

1 $r = Ax$;
2 $r = b - r$;
3 $p = r$;
4 $k = 0$;
5 $\gamma = r^\top r$;
6 **while** $sqrt(\gamma) > \varepsilon$ **do**
7 $\quad v = Ap$;
8 $\quad \delta = v^\top p$;
9 $\quad \alpha = \frac{\gamma}{\delta}$;
10 $\quad x = x + \alpha p$;
11 $\quad r = r - \alpha v$;
12 $\quad \gamma_0 = \gamma$;
13 $\quad \gamma = r^\top r$;
14 $\quad \beta = \frac{\gamma}{\gamma_0}$;
15 $\quad p = r + \beta p$;
16 $\quad k = k + 1$;
17 **end**

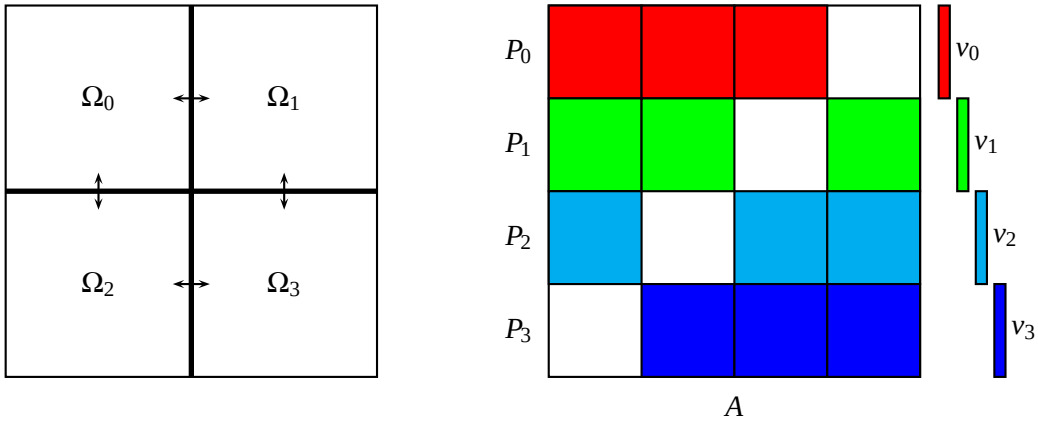
Output: x

Algorithm 8: Pseudo-code for Conjugate Gradient Method

1. Local computation:
   - scalar operations: line 4, 6, 9, 12, 14 and 16;
   - vector operations: line 2, 3, 10, 11, 15;
2. Local computation with communication to combine local results: line 5, 8, 13;
3. Parallel computation: line 1, 7;

The computations listed in the second category are dot products, which require communication to combine local results to the global value. We choose the collective operation *MPI_ALLREDUCE* provided by MPI for such communication. Parallel implementation of matrix-vector product is the key to make the iterative method scalable.

Figure 6.2(a) is an example of partitioning of a given domain with four subdomains $\Omega_0, \cdots, \Omega_3$ and necessary communication between adjacent subdomains. We have four processors $P_0, \cdots, P_3$, and each process corresponds to a subdomain. Figure 6.2(b) is a global view of data distribution of matrix and vector on four processes. Each colored row of matrix A and vector v_i is generated from



(a) Ω_0, Ω_1, Ω_2, and Ω_3 (b) Processes P_0, P_1, P_2, P_3, and local data

Figure 6.2: Data distribution of matrix-vector multiplication on processes P_0, P_1, P_2, and P_3

the local mesh without communication, since discontinuous Galerkin method is a triangle based method. So data exchange is limited between processes that correspond to adjacent subdomains, minimizing the global communication. As the matrix-vector multiplication proceeds, the only communication comes from each subdomain's neighbors.

6.2.4 Data Structure

As shown in Chapter 5, we create a tree structure to store geometric data for edges, triangles, etc. For the parallel implementation, we maintain such a hierarchical structure for each subdomain. Each subdomain has a copy of the initial mesh and creates a sub-tree by refining the elements belonging to the subdomain. Since refinement, PDE data block and matrix generation is local, this approach minimizes the global communication. Figure 6.3 shows the tree structure of the parallel implementation, with each colored sub-tree belonging to a separate subdomain.

Domain Interface Structure

Figure 6.4 is the data structure we defined for the interface of a subdomain to its adjacent subdomain. It contains information about the edges and triangles along the interface of a subdomain.

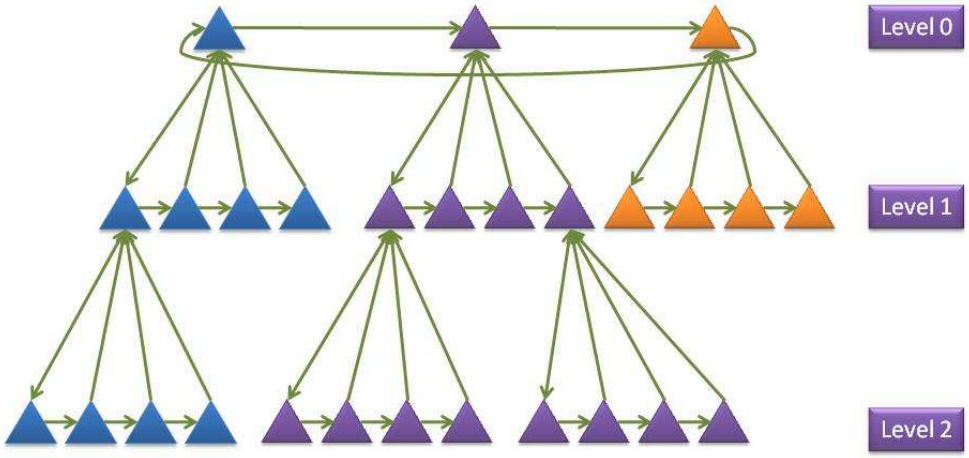


Figure 6.3: Domain tree structure

```
typedef struct dg_interface_domain_t {
  int id;
  int nbr_id;
  int nbr_sidx;
  int nbr_totT;
  int numBE;
  int *betidx;
  EDGE_T **bearray;
  int numBIT;
  int *bitidx;
  TRIANGLE_T **bitarray;
  int numBOT;
  int *botidx;
  TRIANGLE_T **botarray;
  SPMV_BCSR *spm;
} DG_INTERFACE_DOMAIN_T;
```

Figure 6.4: Domain interface structure

Field *id* is the subdomain's id, and *nbr_id* is the id of the subdomain on the other side of the interface. Field *nbr_sidx* is the starting global index of the neighboring subdomain's triangles, and Field *nbr_totT* is the total number of triangles in the neighboring subdomain. *numBE* is the number of the edges on the interface. *numBIT* and *numBOT* are the number of triangles inside subdomain and outside the subdomain along the interface accordingly. *betidx*, *bitidx* and *botidx* are the local indices of edges, and triangles along the interface. *spm* is the off-diagonal matrix block shown in Figure 6.2(b).

Domain Structure

Figure 6.5 is the data structure we defined for a subdomain. Field *id* is the subdomain id. Field *level* indicates the level of mesh to which the subdomain belongs. Field *sidx* is the starting global index of the subdomain's triangles. Fields $totT$, $totE$, and $totBE$ are the number of triangles, edges, and boundary edges of the subdomain. Fields $totIF$ and $IFlist$ are the number of interfaces of the subdomain and the list of interfaces, respectively. *gtotT* and *gtotE* are the total number of triangles and edges, respectively, of the whole mesh at the current level. *mesh* and *dmesh* are the local copy of the initial mesh. *cddomain* and $fddomain$ are the subdomains on the coarse and fine level accordingly.

```
typedef struct dg_dynamic_domain_t {
  int id;
  int level;
  int sidx;
  int totT, totE, totBE, totIF;
  int *totTs, *totEs;
  int gtotT, gtotE;
  DG_MESH_T *mesh;
  DG_DYNMESH_T *ddmesh;
  DG_INTERFACE_DOMAIN_T *IFlist;
  struct dg_dynamic_domain_t *cddomain;
  struct dg_dynamic_domain_t *fddomain;
} DG_DYNDOMAIN_T;
```

Figure 6.5: Domain structure

6.3 Experiment and Performance

The following is a testing problem:

$$\begin{aligned} -\Delta u &= 2x(1-x)+2y(1-y) \qquad \text{in } \Omega \\ u &= 0 \qquad\qquad\qquad\quad \text{on } \partial\Omega \end{aligned} \tag{6.1}$$

The solution of Problem(6.1) is: $u = xy(1-x)(1-y)$ as shown in Figure 6.6, which is a smooth polynomial solution across the domain.

We carried out our experiments on the parallel computer Jaguar[1] at Oak Ridge National Laboratory. Jaguar is a Cray XT4 system with 7832 XT4 compute nodes. Go to http://www.nccs.gov/computing-resources/jaguar for more information. We ran our experiments with 4 nodes up to 128 nodes, and results are shown in Figure 6.7 and 6.8.

Here are the observations:

- We can see that the conjugate gradient method can scale perfectly.
  - localizing data and limiting the communication to each subdomain's neighbors;

[1]This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

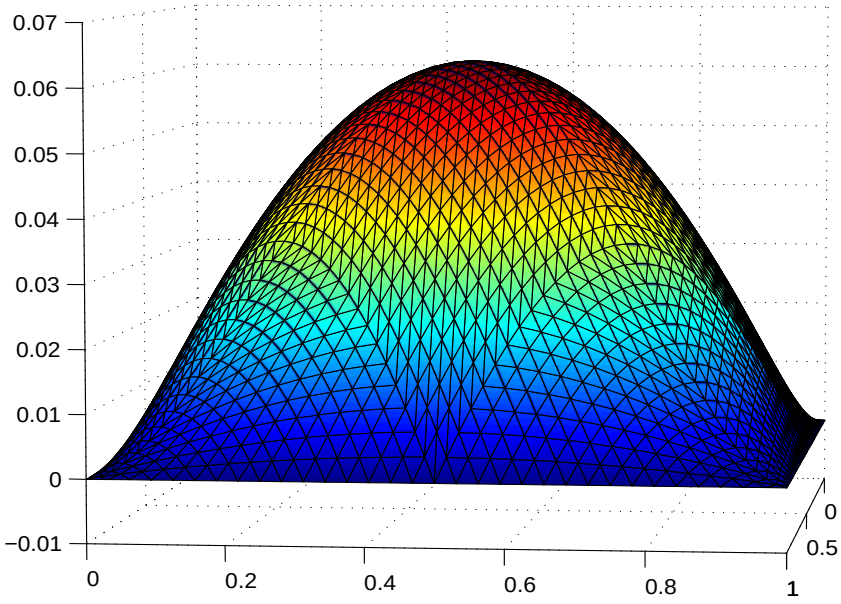


Figure 6.6: Solution of Problem(6.1).

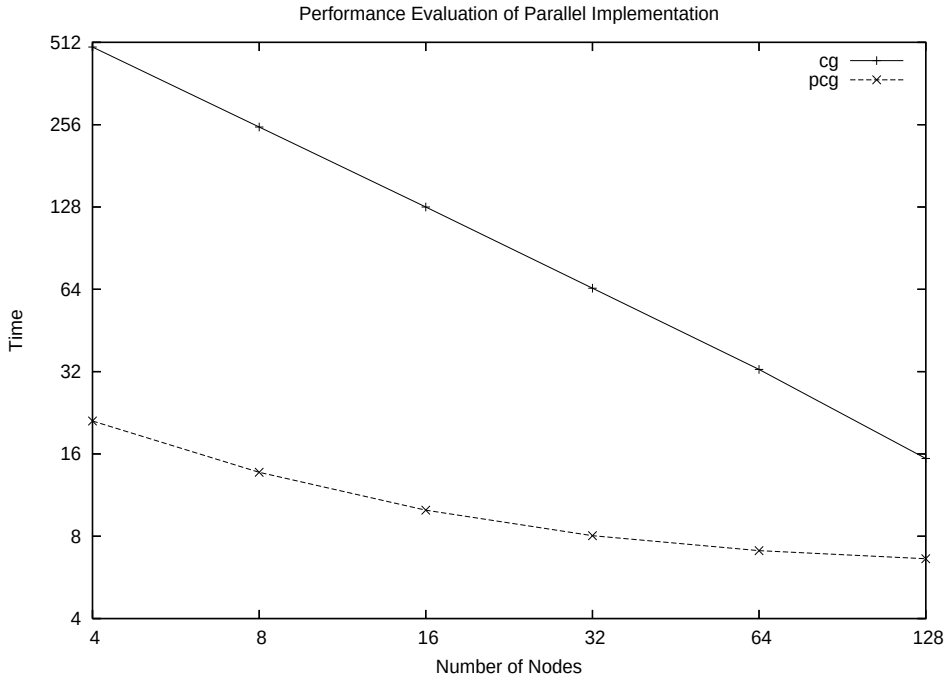


Figure 6.7: Performance evaluation of parallel implementation on Jaguar: Time

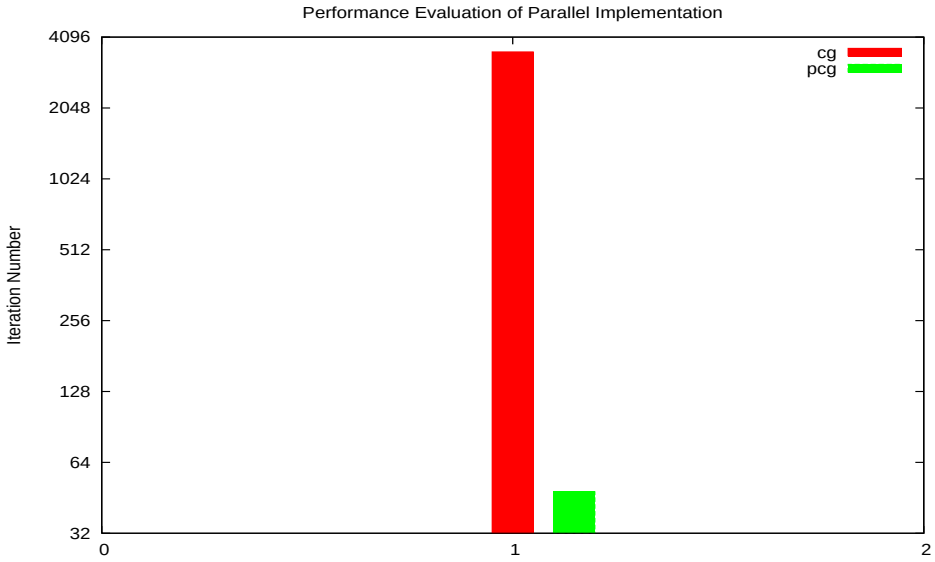


Figure 6.8: Performance evaluation of parallel implementation on Jaguar: Iteration Numbers

  - Using MPI non-blocking sendrecv operations to overlap communication and computation for dot product and matrix-vector multiplication;
  - Using MPI collective operation.
  - fast inter-node connection.

- The preconditioned conjugate gradient method takes much less time to finish the job compared to the CG method, and it scales sublinearly.
  - As we proved in Chapter 3, the condition number of PCG is a constant. Figure 6.8 shows the number of iterations of PCG is much smaller than CG's, and we see better performance of PCG consequently;
  - We chose a direct solver for the coarse mesh correction of the preconditioner. For our parallel implementation, the reason for not choosing the CG method is that the linear system of coarse mesh correction is a small linear system for which it is not beneficial to use a parallel CG method. Since each subdomain computes its own coarse mesh correction, it is actually a sequential part of the whole process. By Amdahl's law, it has a negative impact on scalability.

Chapter 7

Summary and Future Directions

Discontinuous Galerkin FEM has been an active research area for years. Since there are no continuity constraints such as exist in standard Galerkin FEM, DGFEM has great advantages such as high-order accuracy on unstructured meshes, local *hp*-refinement, weak imposition of boundary conditions and local conservation. The drawback is that it requires one to solve for a larger number of unknowns than continuous Galerkin FEM. This research tackles the performance issue from both theoretical and computational fields and has achieved satisfactory results. We notice that the performance of computation has been improved so much that marking and refinement of adaptive FEM are becoming more time consuming. Hence, better memory management and implementation are needed for that part, and we are convinced there is much room for improvements. As new computer architectures are emerging such as multi-core, many-core, GPU, FPU, etc., we believe that DGFEM and our proposed *RDR* format CGFEM, as well as the mixed scheme DG-CGFEM we propose in this research, can do very well on these architectures. Precise embedding and projection operators will be needed for the non-conforming adaptive meshes in the mixed scheme however. In conclusion, this research has provided an opportunity to explore the mathematical theory and to exercise in a spectrum of computational fields such as linear algebra (dense and sparse), code optimization (auto-tuning and SSE assembly coding), multi-threading, MPI, algorithm design, etc. As a result, software consisting of 60000 lines of code has been produced.

Bibliography

Bibliography

[1] R. A. Adams. *Sobolev Spaces*. Academic Press, New York, 1975.

[2] Randy Allen and Ken Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufmann Publishers, 2002.

[3] D. Arnold, F. Brezzi, B. Cockburn, and D. Marini. Discontinuous Galerkin methods for elliptic problems. In B. Cockburn, G.E. karniadakis, and C.-W. Shu, editors, *Proceedings of the International Symposium on the discontinuous Galerkin method*, volume 11, pages 89–101. Springer lecture notes in Computational Science and Engineering, 2000.

[4] G. A. Baker, W. N. Jureidini, and O. A. Karakashian. Piecewise solenoidal vector fields and the Stokes problems. *SIAM J. Numer. Anal.*, 27:1466–1485, 1990.

[5] Utpal Banerjee. A Theory of Loop Permutations. In *Selected Papers of the Second Workshop on Languages and Compilers for Parallel Computing*, pages 54–74. Pitman Publishing, 1990.

[6] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[7] Jeff Bilmes, Krste Asanovic, Chee-Whye Chin, and James Demmel. Optimizing Matrix Multiply Using PHiPAC: A Portable, High-Performance, ANSI C Coding Methodology. In *International Conference on Supercomputing*, pages 340–347, 1997.

[8] S. Brenner and R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer-Verlag, New York, 1994.

[9] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland, Amsterdam, 1978.

[10] R. Courant. Variational methods for the solutions of equilibrium and vibrations. *Bull. Amer. Math. Soc.*, pages 1–23, 1943.

[11] L. C. Cowsar, E. J. Dean, R. Glowinski, P. Le Tallec, C. H. Li, J. Periaux, and M. F. Wheeler. Decomposition principles and their applications in scientific computing. In J. Dongarra, K. Kennedy, P. Messina, D. Sorensen, and R. Voigt, editors, *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, pages 213–237, 1992.

[12] Lawrence C. Cowsar, Alan Weiser, and Mary F. Wheeler. Parallel multigrid and domain decomposition algorithms for elliptic equations. In David E. Keyes et al., editors, *Proceedings of the Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 376–385, 1992.

[13] Lawrence C. Cowsar and Mary F. Wheeler. Parallel domain decomposition method for mixed finite elements for elliptic partial differential equations. In Roland Glowinski et al., editors, *Proceedings of the Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, 1991.

[14] Jim Demmel, Jack Dongarra, Victor Eijkhout, Erika Fuentes, Antoine Petitet, Rich Vuduc, Clint Whaley, and Katherine Yelick. Self adapting linear algebra algorithms and software. *Proceedings of the IEEE*, 93(2), 2005. special issue on ”Program Generation, Optimization, and Adaptation”.

[15] Willy Dorfler. A convergent adaptive algorithm for poisson’s equation. *SIAM Journal on Numerical Analysis*, 33(3):1106–1124, Jun. 1996.

[16] M. Dryja and O.B. Widlund. Towards a unified theory of domain decomposition algorithms for elliptic problems. In T. Chan etc., editor, *Proceedings of Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 3–21, Philadelphia, 1990. SIAM.

[17] X. Feng and O. A. Karakashian. Analysis of two-level overlapping additive schwarz preconditioners for a discontinuous galerkin method. In *Proceedings of Thirteenth International Conference of Domain Decomposition Methods*. DDM.org Press, 2001.

[18] X. Feng and M. T. Rahman. An additive average Schwarz method for the plate bending problem. Technical Report 185, Dept. of Informatics, University of Bergen, Norway, Feb. 2000.

[19] Eun-Jin Im, Katherine Yelick, and Richard Vuduc. Sparsity: Optimization framework for sparse matrix kernels. *Int. J. High Perform. Comput. Appl.*, 18(1):135–158, 2004.

[20] O. A. Karakashian and F. Pascal. A posteriori error estimates for a discontinuous galerkin approximation of second-order elliptic problems. *SIAM J. Numer. Anal.*, 41:2374–2399, 2003.

[21] G. Karypis and V. Kumar. Metis users manual: Unstrctured graph partitioning and sparse matrix ordering system. Technical report, ”1995”.

[22] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[23] C. Lasser and A. Toselli. An overlapping domain decomposition preconditioner for a class of discontinuous Galerkin approximations of advection-diffusion problems. Technical Report 2000-12, Seminar für Angewandte Mathenatik, ETH, Zürich, 2000.

[24] Kathryn S. McKinley, Steve Carr, and Chau-Wen Tseng. Improving Data Locality with Loop Transformations. *ACM Trans. Program. Lang. Syst.*, 18(4):424–453, 1996.

[25] Hans Meuer, Erich Strohmaier, Jack Dongarra, and Horst Simon. TOP500 Supercomputing Sites. Technical report.

[26] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, New York, 1999.

[27] W.H. Reed and T.R. Hill. Triangular mesh methods for the neutron transport equation. Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.

[28] P. E. Bjørstad, M. Dryja, and E. Vainikko. Additive schwarz methods without subdomain overlap and with new coarse spaces. In R. Glowinski, J. Périaux, Z-C. Shi, and O. Widlund, editors, *Domain Decomposition Methods in Sciences and Engineering*, pages 141–157. Wiley & Sons, New York, 1997.

[29] T. Rusten, P. S. Vassilevski, and R. Winther. Interior penalty preconditioners for mixed finite element approximations of elliptic problems. *Math. Comp.*, 65:447–466, 1996.

[30] Michael A. Saum. Adaptive discontinuous galerkin finite element methods for second and fourth order elliptic partial differential equations. Technical report, "University of Tennessee, Knoxville, TN, USA", "August, 2006".

[31] Robert Schreiber and Jack Dongarra. Automatic Blocking of Nested Loops. Technical Report CS-90-108, Knoxville, TN 37996, USA, 1990.

[32] Jonathan Richard Shewchuk. Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator. Technical report.

[33] B. E. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, 1996.

[34] Richard Wilson Vuduc. Automatic performance tuning of sparse matrix kernels. Technical report, "University of California, Berkeley, Berkeley, CA, USA", "December, 2003".

[35] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. In *SC '98: Proceedings of the Proceedings of the IEEE/ACM SC98 Conference*, page 38. IEEE Computer Society, 1998.

[36] J. Xu. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34:581–613, 1992.

[37] Qing Yi, Ken Kennedy, Haihang You, Keith Seymour, and Jack Dongarra. Automatic Blocking of QR and LU Factorizations for Locality. In *2nd ACM SIGPLAN Workshop on Memory System Performance (MSP 2004)*, 2004.

[38] Kamen Yotov, Xiaoming Li, Gang Ren, Michael Cibulskis, Gerald DeJong, Maria Garzaran, David Padua, Keshav Pingali, Paul Stodghill, and Peng Wu. A Comparison of Empirical and Model-driven Optimization. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, pages 63–76. ACM Press, 2003.

Appendix

Appendix A

Affine Transformation

In 2D case, an element is a triangle. Figure A.1 shows a reference triangle $\hat{K}$ and an arbitrary triangle K. In practice the values of polynomial $P_n(x)$ and their first-order derivatives are pre-computed on the reference triangle. To calculate the stiffness matrix (Diri block) for an triangle, an Affine Transformation from an individual triangle to a reference triangle is defined as following:

The Affine transformation **F** from reference triangle $\hat{\mathbf{K}}$ to triangle **K** is:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{A.1}$$

The Affine transformation $\mathbf{F^{-1}}$ from reference triangle **K** to triangle $\hat{\mathbf{K}}$ is:

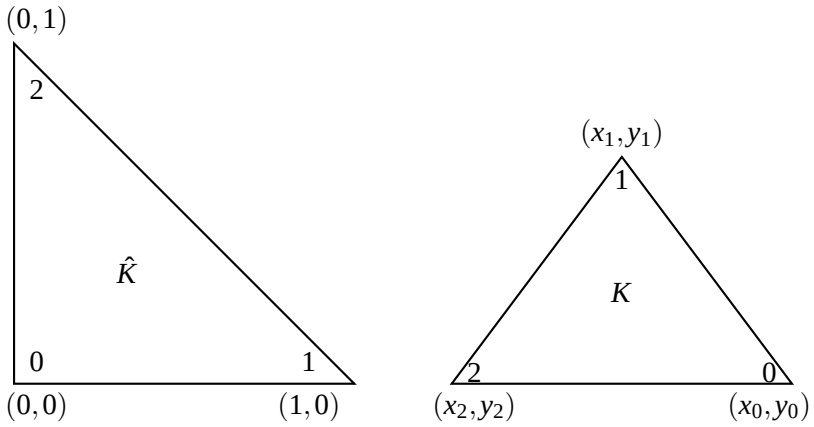
$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \frac{1}{2|K|} \begin{bmatrix} y_2 - y_0 & -(x_2 - x_0) \\ -(y_1 - y_0) & x_1 - x_0 \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} \tag{A.2}$$

or

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \frac{1}{2|K|} \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} \tag{A.3}$$

where $|\mathbf{K}| =$ **area of the triangle**.

Then base function derivative can be represented by reference triangle's base functions' deriva-



(a) $\hat{K}$ (b) K

Figure A.1: Affine transformation between triangle K and reference triangle $\hat{K}$

tives:

$$\nabla\phi = \frac{1}{2|K|}\begin{bmatrix} c_{11} & c_{21} \\ c_{12} & c_{22} \end{bmatrix}\hat{\nabla}\hat{\phi} \tag{A.4}$$

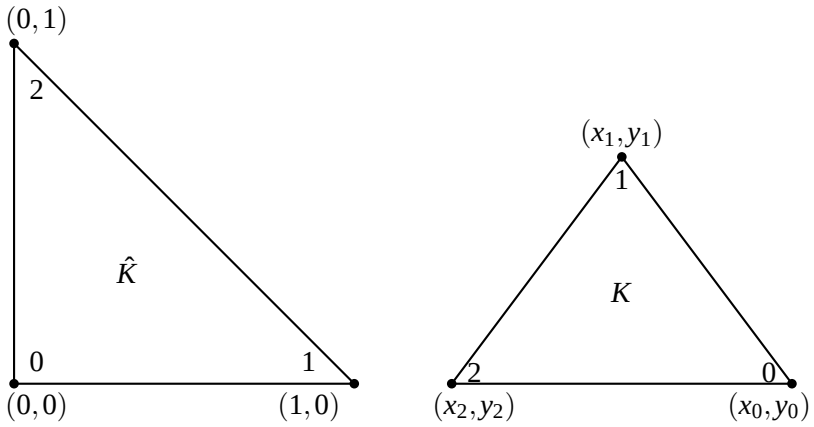
$$a_{ji} = a_{ij} = \int_K \nabla\phi_i \cdot \nabla\phi_j dxdy \tag{A.5}$$

$$a_{ji} = a_{ij} = \frac{1}{2|K|}\int_{\hat{K}} (C\hat{\nabla}\hat{\phi}_l)^T \cdot (C\hat{\nabla}\hat{\phi}_m) d\hat{x}d\hat{y} \tag{A.6}$$

$$b_j = \int_K f(x,y)\phi_j(x,y)dxdy \tag{A.7}$$

$$b_j = 2|K|\int_{\hat{K}} \hat{f}(\hat{x},\hat{y})\hat{\phi}_m(\hat{x},\hat{y})d\hat{x}d\hat{y} \tag{A.8}$$

where $1 \leq i, j \leq N$, are global indices of DOFs on the triangle, and $0 \leq l, m \leq 2$ are local indices.



(a) $\hat{K}$ (b) K

Figure A.2: Vertices of 1*st* order basis functions on a triangle K and reference triangle $\hat{K}$

A.1 First Order Basis Functions

1. Vertices on the reference triangle:

$$\hat{z}_0 = (0,0) \quad \hat{z}_1 = (1,0) \quad \hat{z}_2 = (0,1)$$

   Vertices on a triangle:

$$z_0 = (x_0,y_0) \quad z_1 = (x_1,y_1) \quad z_2 = (x_2,y_2)$$

2. Basis functions are:

$$\hat{\phi}_0(\hat{x},\hat{y}) = 1-\hat{x}-\hat{y}$$
$$\hat{\phi}_1(\hat{x},\hat{y}) = \hat{x}$$
$$\hat{\phi}_2(\hat{x},\hat{y}) = \hat{y}$$

3. 1^{st} order derivative basis functions are:

$$\hat{\nabla}\hat{\phi}_0(\hat{x},\hat{y}) = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

$$\hat{\nabla}\hat{\phi}_1(\hat{x},\hat{y}) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\hat{\nabla}\hat{\phi}_2(\hat{x},\hat{y}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

A.2 Second Order Basis Functions

1. Vertices on the reference triangle:

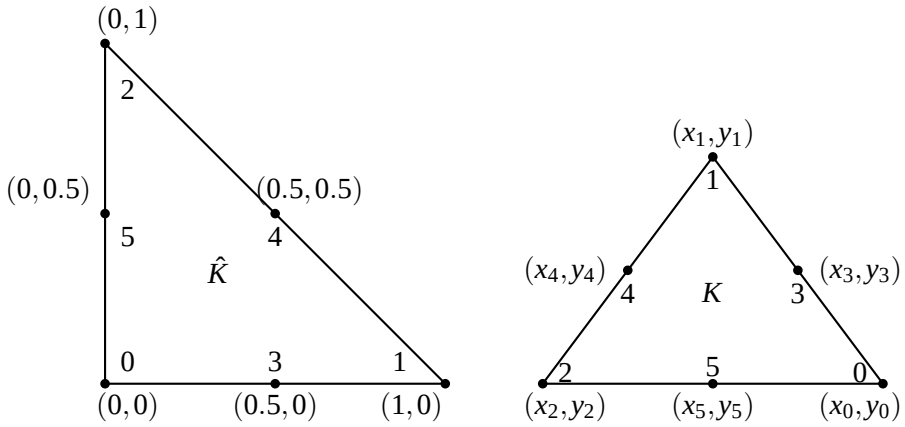
$$z_1 = (0,0) \quad z_2 = (1,0) \quad z_3 = (0,1)$$

$$z_4 = (0.5,0) \quad z_5 = (0.5,0.5) \quad z_6 = (0,0.5)$$

Vertices on a triangle:

$$z_0 = (x_0,y_0) \quad z_1 = (x_1,y_1) \quad z_2 = (x_2,y_2)$$

$$z_3 = (x_3,y_3) \quad z_4 = (x_4,y_4) \quad z_5 = (x_5,y_5)$$



(a) $\hat{K}$ (b) K

Figure A.3: Vertices of 2*nd* order basis functions on a triangle K and reference triangle $\hat{K}$

2. Basis functions are:

$$\hat{\phi}_1(\hat{x},\hat{y}) = 2\hat{x}^2 + 4\hat{x}\hat{y} + 2\hat{y}^2 - 3\hat{x} - 3\hat{y} + 1$$

$$\hat{\phi}_2(\hat{x},\hat{y}) = 2\hat{x}^2 - \hat{x}$$

$$\hat{\phi}_3(\hat{x},\hat{y}) = 2\hat{y}^2 - \hat{y}$$

$$\hat{\phi}_4(\hat{x},\hat{y}) = -4\hat{x}^2 - 4\hat{x}\hat{y} + 4\hat{x}$$

$$\hat{\phi}_5(\hat{x},\hat{y}) = 4\hat{x}\hat{y}$$

$$\hat{\phi}_6(\hat{x},\hat{y}) = -4\hat{x}\hat{y} - 4\hat{y}^2 + 4\hat{y}$$

3. 1^{st} order derivative basis functions are:

$$\hat{\nabla}\hat{\phi}_0(\hat{x},\hat{y}) = \begin{bmatrix} 4\hat{x}+4\hat{y}-3 \\ 4\hat{x}+4\hat{y}-3 \end{bmatrix}$$

$$\hat{\nabla}\hat{\phi}_1(\hat{x},\hat{y}) = \begin{bmatrix} 4\hat{x}-1 \\ 0 \end{bmatrix}$$

$$\hat{\nabla}\hat{\phi}_2(\hat{x},\hat{y}) = \begin{bmatrix} 0 \\ 4\hat{y}-1 \end{bmatrix}$$

$$\hat{\nabla}\hat{\phi}_3(\hat{x},\hat{y}) = \begin{bmatrix} -8\hat{x}-4\hat{y}+4 \\ -4\hat{x} \end{bmatrix}$$

$$\hat{\nabla}\hat{\phi}_4(\hat{x},\hat{y}) = \begin{bmatrix} 4\hat{y} \\ 4\hat{x} \end{bmatrix}$$

$$\hat{\nabla}\hat{\phi}_5(\hat{x},\hat{y}) = \begin{bmatrix} -4\hat{y} \\ -4\hat{x}-8\hat{y}+4 \end{bmatrix}$$

As we have shown above, on each triangle, we have 3 DOFs for 1st order polynomials, 6 for 2nd order, and for 3rd order and 4th order polynomials, we have 10 and 16 DOFs respectively.

Vita

Haihang You was born in Hefei, China on October 28, 1971. He completed his high school in 1990, and after that he joined Beijing Normal University, Beijing, China for pursuing his undergraduate degree. He obtained his Bachelor of Science in Physics in 1994. In 1998, he traveled to United States to pursue graduate studies. He obtained his Master's degree in Computer Science at University of Tennessee, Knoxville in 2001. After graduation, He joined Innovative Computing Laboratory(ICL) as full time research staff. Later on he started the Doctorate program in Computer Science, his advisors are Prof. Jack Dongarra at Computer Science department and Prof. Ohannes Karakashian at Math department.