CRPC Parallel Computing Handbook

Jack Dongarra, Ian Foster, Geoffrey Fox, Ken Kennedy, Linda Torczon, and Andy White

Editors

Early Draft

Preface

This book is aimed at students and practitioners of technical computing who need to understand both the promise and practice of high performance and parallel computing. It can be used as a resource by both computer science and application researchers. The book and its associated Web site can be used in computational science and parallel computing education and training. The principal goal of this book is to make it easy for those entering the field of parallel computing with a good background in applications or computational science to understand the technologies available and how to apply them.

The book is aimed at the users of high performance systems whose architectures span the range of small desktop SMP's and PC clusters to high-end supercomputers costing $100 M or more. The book focuses on software technologies, along with the large-scale applications enabled by them. In each area, the text contains a general discussion of the state of the field followed by detailed descriptions of key technologies or methods. In some cases, such as MPI for message passing, this is the dominant approach. In others, such as the discussion of problem solving environments, the authors choose systems representing key concepts in an emerging area.

The book is organized into five sections. In the first section, the field is summarized with an emphasis on motivating applications and the external forces, from the Internet to the HPCC Presidential Initiatives, that have driven the field. This is followed by an overview of the current status of hardware architectures. The next two sections describe applications and software technologies in detail. The final section discusses futures from both a technology and application perspective. There is a related Web site with a set of community resources, CRPC papers, and links to other sites of interest.

The application section is designed to help new users learn if and how high performance techniques can be applied in their area. It consists of an overview of the process by which one identifies appropriate software and algorithms and of the issues involved in implementation. Some twenty vignettes of parallel systems in different areas, which briefly describe successful approaches to use, illustrate these general comments. These examples have been chosen to cover a broad range of both scientific areas and numerical approaches. This overview material is complemented by in-depth studies in areas including computational fluid dynamics, environmental engineering, astrophysical particle simulations, and computational engineering. The applications are cross-referenced to the software technologies section.

The software technologies section will discuss the progress made on a variety of software technologies, including message passing libraries, parallel I/O and parallel file systems, run-time libraries for parallel computing, languages like HPF and HPC++, problem solving environments, high-level programming systems, performance analysis and tuning tools, load balancing technologies, grid generation technologies, and numerical systems and libraries. The goal of this section is to provide a survey of progress with hints to the user that will help in selecting the right technology for use in a given application.

The final section of the book is a discussion of important future problems for the high performance science and engineering community, including distributed computing in a grid environment.

Contents

Chapter 1

Parallel Computing in CFD

Ron Henderson, Dan Meiron, Ravi Samtaney, & Herb Keller (Geoffrey Fox, editor)

Target Length: 20 pages

1.1 Introduction and Overview

The basic equations of fluid mechanics are presented. A brief overview is provided of some of the common physical regimes described by these equations (compressible vs.incompressible flow) and the associated dimensionless parameters associated with these physical regimes (Reynolds number, Mach number). The need to utilize high performance computation to solve these equations in many cases of interest is motivated via some example applications.

The particular computational difficulties associated with incompressible viscous CFD are described For complex geometries which are of practical interest, special attention is paid to the application of the spectral element method and its parallel implementation.

A brief overview is presented of approaches to the numerical simulation of compressible CFD. It is argued that the need to resolve fine scale features such as shock waves makes the use of adaptive mesh refinement essential especially in three dimensions. The difficulty of establishing load balancing and scalability for such calculations is discussed. The chapter concludes with a brief discussion of some future computational challenges for CFD and an assessment of the computational resources required to overcome these challenges.

1.1.1 Basic Equations of Fluid Dynamics

The motion of a fluid is governed by the principles of classical mechanics and thermodynamics, namely, conservation of mass, momentum, and energy. The most general statement of these principles is carried out in integral form in a stationary frame of reference leading to the following conservation equations:

$$\frac{d}{dt}\int_V \rho dV + \int_\Sigma \rho \boldsymbol{u}\cdot\boldsymbol{n} d\Sigma = 0, \tag{1a}$$

$$\frac{d}{dt}\int_V \rho \boldsymbol{u} dV + \int_\Sigma [(\boldsymbol{n}\cdot\boldsymbol{u})\rho\boldsymbol{u} - \boldsymbol{n}\boldsymbol{\sigma}] d\Sigma = \int_V \mathbf{f}_e dV \tag{1b}$$

$$\frac{d}{dt}\int_V \rho E dV + \int_\Sigma \boldsymbol{n}\cdot[\rho E\boldsymbol{u} - \boldsymbol{\sigma}\boldsymbol{u} + \mathbf{q}] d\Sigma = \int_V \mathbf{f}_e\cdot\boldsymbol{u} dV \tag{1c}$$

Here, t is time, ρ is density, $\boldsymbol{u}$ is the velocity of a material fluid particle in this frame of reference, E is the total specific energy

$$E = e + \frac{1}{2}\boldsymbol{u}\cdot\boldsymbol{u} \tag{2}$$

where e is the specific internal energy, $\boldsymbol{\sigma}$ is the stress tensor, $\mathbf{q}$ is the heat flux, $\mathbf{f}_e$ is the external force per unit volume and $\boldsymbol{n}$ is the unit outward normal to the surface Σ enclosing the fluid volume V. We are ignoring other sinks and sources of energy such as those arising from chemical reactions or other phenomena.

The solutions of equations (1a,1b,1c) need not be continuous functions of space and it is for this reason that the equations are written in integral form. However if the flow density, velocity and energy are sufficiently smooth then these equations can be transformed into an equivalent set of partial differential equations through the use of the divergence theorem:

$$\partial_t\rho + \nabla\cdot(\rho\boldsymbol{u}) = 0 \tag{3a}$$

$$\partial_t(\rho\boldsymbol{u}) + \nabla\cdot(\rho\boldsymbol{u}\boldsymbol{u} - \boldsymbol{\sigma}) = \mathbf{f}_e \tag{3b}$$

$$\partial_t(\rho E) + \nabla\cdot(\rho E\boldsymbol{u} - \boldsymbol{\sigma}\boldsymbol{u} + \mathbf{q}) = \mathbf{f}_e\cdot\boldsymbol{u} \tag{3c}$$

The basic dependent variables are the density, velocity, and energy of the flow. Constitutive relations for the stress tensor $\boldsymbol{\sigma}$ and for the heat flux vector $\mathbf{q}$ must be added to these equations in order to form a closed system. A Navier-Stokes fluid is defined by the following constitutive relations:

$$\boldsymbol{\sigma} = -p\mathrm{I} + \lambda(\nabla\cdot\boldsymbol{u})\mathrm{I} + \mu\left[(\nabla\boldsymbol{u}) + (\nabla\boldsymbol{u})^T\right] \tag{4}$$

Here p is the pressure, and λ and μ are coefficients of viscosity. The fluid is assumed to obey the Fourier law of heat conduction

$$\mathbf{q} = -k\nabla T \tag{5}$$

where T is the absolute temperature, and k is the thermal conductivity. Finally since we assume that the fluid is locally in thermodynamic equilibrium we require an equation of state for the fluid which relates for example the entropy of the fluid to the density and internal energy:

$$S = S(\rho, e) \tag{6}$$

where S is the entropy, and from this and the thermodynamic equalities

$$p = -\rho^2 T \left(\frac{\partial S}{\partial \rho}\right)_e , \qquad T^{-1} = \left(\frac{\partial S}{\partial e}\right)_\rho \tag{7}$$

the Navier-Stokes equations become a closed system for the dynamic variables ρ, $\boldsymbol{u}$, and E.

An important special case of these equations is the flow of a perfect gas with constant specific heats C_p, and C_v. For such a gas the equation of state is the well known ideal gas law:

$$p = (\gamma - 1)\rho e, \qquad \gamma = \frac{C_p}{C_v} \tag{8a}$$

$$e = C_v T \tag{8b}$$

1.1.2 Physical Regimes and Dimensionless Variables

The Navier-Stokes equations have been shown to be valid over a wide class of flow regimes. A useful approach to distinguishing the key regimes is to scale the physical variables and to rewrite the equations in dimensionless form. To do this we scale all quantities relative to a reference length L, a reference velocity V^*, a reference density ρ^* and reference values of the coefficients of the viscosity μ abd thermal conductivity, k. All other characterisitic quantities can be derived from these basic ones although some understanding of the various balances of terms in the equations is required to achieve meaningful results. We choose L/V^* to scale time t, $\rho^* V^{*2}$ to scale the stress $\boldsymbol{\sigma}$ and so forth. In this dimensionless form the equations remain essentially unchanged but the consTitutive laws reappear in a scaled form:

$$\boldsymbol{\sigma} = -p\mathrm{I} + \frac{1}{Re}\left[\lambda(\nabla \cdot \boldsymbol{u})\mathrm{I} + \mu\left[(\nabla \boldsymbol{u}) + (\nabla \boldsymbol{u})^T\right]\right] \tag{9}$$

where Re is the Reynolds number and is given by $Re = V^* L \rho^*/\mu^*$.

The Reynolds number is a measure of the ratio of inertial to viscous forces acting within the fluid. A low Reynolds number signifies flow dominated by viscous effects. A high Reynolds number indicates flows dominated by inertial effects. This would seem to imply that one could ignore the viscous terms for flows at high Reynolds number (for example for flow around an aircraft or car which is typically in the range of $Re = 10^5 - 10^8$. However, this is not quite correct since the viscous terms become important near solid boundaries (such as the wing or body of the airplane). In addition, in a turbulent flow, the viscous terms are active at small length scales and cannot be ignored if one wants to compute how much energy is required for example to keep the flow moving at the characteristic velocity implied by a high Reynolds number.

If we assume the fluid is a perfect gas then it can be shown the heat flux is given by

$$\mathbf{q} = -\frac{\gamma}{RePr} k \nabla e \tag{10}$$

where $Pr = \mu^* C_p / k^*$ is the Prandtl number which measures the relative importance of viscous to thermal diffusion. Finally for a perfect gas with constant specific heats the equation of state becomes

$$e = \frac{T}{\gamma(\gamma - 1)M^2} \tag{11}$$

where $M = V^*/\sqrt{\gamma R T^*}$ is the Mach number, which measures the ratio of the characteristic velocity to the speed of sound of the gas at temperature T^*. It can be shown that provided the velocity of the fluid remains substantially lower than the speed of sound the flow is essentially incompressible meaning that the density does not change as the flow evolves. In this case the equations simplify and the equation of state of the fluid becomes irrelevant.

For flows with velocities comparable or exceeding the local speed of sound it is possible to generate shock waves in the fluid which are essentially thin layers of fluid separating regions in which the flow is locally supersonic from those in which the flow is subsonic. The viscous terms again become very important in these thin shock regions.

1.1.3 The Role of High Performance Computing

Numerical computation of fluid flows and in particular the use of high performance computation plays a critical role in fluid mechanics research for several reasons. First, the equations of motion as described above are nonlinear in character. There exist exact solutions to these equations only for highly simplified geometries and initial conditions. Numerical computation is essential for solving general initial value problems in realistic geometries such as the flow over an automobile or an airplane wing. In addition the number of degrees of freedom required to accurately simulate flows in realistic geometries rises rapidy with Reynolds number and Mach number.

To get a feel for the computational requirements consider the simulation of turbulent flow without boundaries. It can be shown that the number of degrees of freedom required to properly simulate all relevant length scales in the flow (including the dissipation-producing length scales due to viscosity) varies as $Re^{9/4}$. For a moderate Reynolds number of 10^6 this implies a total of 3×10^{13} degrees of freedom per field. Typically this needs to be multiplied by 10-15 to accomodate the required fields. Thus roughly 300 Terawords of memory are required simply to describe the flow. In order to integrate the flow forward in time it is clear that one requires a machine with a speed of several Teraflops. Such architectures are only now becoming available.

Turbulent flow is not the only application driver. Even if the flow is kept smooth and laminar the computation of fluid flow about a solid body such as an airplane or car still requires substantial resources. At the surface of the body the flow staisfied the “no-slip” condition and is constrained to move at the velocity of the body. The flow accomodates to this condition via a thin boundary layer in which the viscous terms are sizable. The thickess of a laminar boundary layer scales as $Re^{-1/2}$. For example the boundary layer on a 20 foot automobile traveling at 55 mph is about 110'th of an inch. Thus again there is a wide range of scales required in order to capture correctly the flow.

An even more severe ratio of length scales occurs for compressible flow with shock waves. The thickness of strong shock waves is only on the order of a few molecular mean free paths for a gas. The mean free path is typically several orders of magnitude smaller than any characterisitc length scale of the mean flow. In fact it is currently impractical to perform computations of compressible flows with shock waves in which viscous effects are resolved across the shock wave except at Mach numbers near 1.

The need to resolve the enormous range of scales in the example above makes the use of computational fluid dynamics essential. At the same time it is currently not possible to perform direct numerical simulations of engineering flows in which all relevant scales are resolved. In all such flows some model of the small scales must be introduced. For turbulent flows we instriduce a turbulence model to perform th dissipation of missing scales. For strongly compressible flows we employ modern artifical viscosities which allow us to capture correctly the large scale effects of the shock wave.

1.2 Incompressible Flows

We begin our discussion by considering Newtonian incompressible fluids with constant density ρ and kinematic viscosity $\nu = \mu/\rho$, the motion of which is governed by the incompressible Navier–Stokes equations:

$$\nabla \cdot \boldsymbol{u} = 0 \quad \text{in } \Omega, \tag{12a}$$

$$\partial_t \boldsymbol{u} = \mathbf{N}(\boldsymbol{u}) - \frac{1}{\rho}\nabla p + \frac{1}{Re}\nabla^2 \boldsymbol{u} \quad \text{in } \Omega, \tag{12b}$$

where $\boldsymbol{u} = (u_1, u_2, u_3)$ is the velocity field, p is the static pressure, $Re \equiv UL/\nu$ is the Reynolds number, and Ω is the computational domain. Without loss of generality we take the numerical value of $\rho = 1$ since this simply sets the scale for p. $\mathbf{N}(\boldsymbol{u})$ represents the nonlinear advection term:

$$\mathbf{N}(\boldsymbol{u}) \quad = \quad -(\boldsymbol{u} \cdot \nabla)\boldsymbol{u}, \tag{13a}$$

$$= \quad -\frac{1}{2}\left[(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \nabla \cdot (\boldsymbol{u}\boldsymbol{u})\right], \tag{13b}$$

$$= \quad -\frac{1}{2}\nabla(\boldsymbol{u} \cdot \boldsymbol{u}) - \boldsymbol{u} \times \nabla \times \boldsymbol{u}. \tag{13c}$$

We refer to these as the *convective* form, *skew-symmetric* form, and *rotational* form, respectively. These three forms for $\mathbf{N}(\boldsymbol{u})$ are mathematically equivalent but behave differently when implemented for a discrete system. As shown by Zang [23], the skew-symmetric form is the most robust; this form is used in all calculations.

The Navier–Stokes equations are coupled through the incompressibility constraint $\nabla \cdot \boldsymbol{u} = 0$ and the nonlinear term $\mathbf{N}(\boldsymbol{u})$. However, the biggest challenge for time-integration comes from the *linear* term:

$$\mathbf{L}(\boldsymbol{u}) \equiv \frac{1}{Re}\nabla^2 \boldsymbol{u}. \tag{14}$$

This term is responsible for the fastest time scales in the system and thus poses the most severe constraint on the maximum allowable time step for numerical integration of the fluid equations. Problems associated with the stiffness of the linear operator are handled by treating this term implicitly, while the nonlinear term is usually integrated with a more direct and easily implemented explicit method.

1.2.1 Semi-discrete formulation

To solve the Navier–Stokes equations, (12b) is integrated over a single time step to obtain:

$$\boldsymbol{u}(t+\Delta t) = \boldsymbol{u}(t) + \int_t^{t+\Delta t} [\mathbf{N}(\boldsymbol{u}) - \frac{1}{\rho}\nabla P + \mathbf{L}(\boldsymbol{u})]\,\mathrm{d}t. \tag{15}$$

Next we introduce a discrete set of times $t_n \equiv n\Delta t$ where the solution is to be evaluated, and define $\boldsymbol{u}^n \equiv \boldsymbol{u}(\boldsymbol{x}, t_n)$ as the semi-discrete approximation to the velocity (discrete in time, continuous in space). For reasons that will be explained in a moment, the pressure integral is replaced with:

$$\nabla\tilde{P} \equiv \frac{1}{\Delta t}\int_{t_n}^{t_{n+1}} \frac{1}{\rho}\nabla P\,\mathrm{d}t. \tag{16}$$

Next we introduce appropriate integration schemes for the linear and nonlinear terms. The simplest implicit/explicit scheme would be first-order Euler time integration:

$$\int_{t_n}^{t_{n+1}} \mathbf{L}(\boldsymbol{u})\,\mathrm{d}t \approx \Delta t\,\mathbf{L}(\boldsymbol{u}^{n+1}); \tag{17}$$

$$\int_{t_n}^{t_{n+1}} \mathbf{N}(\boldsymbol{u})\,\mathrm{d}t \approx \Delta t\,\mathbf{N}(\boldsymbol{u}^{n}). \tag{18}$$

Combining (16)–(18) we get a semi-discrete approximation to the momentum equation:

$$u^{n+1} = u^n + [\mathbf{N}(\boldsymbol{u}^n) - \nabla\tilde{P} + \mathbf{L}(\boldsymbol{u}^{n+1})]\,\Delta t. \tag{19}$$

This system of equations can be solved by further splitting (19) into three substeps as follows:

$$\begin{aligned} \boldsymbol{u}^{(1)} - \boldsymbol{u}^n &= \Delta t\,\mathbf{N}(\boldsymbol{u}^n), && \text{(20a)}\\ \boldsymbol{u}^{(2)} - \boldsymbol{u}^{(1)} &= -\Delta t\,\nabla\tilde{P}, && \text{(20b)}\\ \boldsymbol{u}^{n+1} - \boldsymbol{u}^{(2)} &= \Delta t\,\mathbf{L}(\boldsymbol{u}^{n+1}). && \text{(20c)} \end{aligned}$$

Here $\boldsymbol{u}^{(1)}$ and $\boldsymbol{u}^{(2)}$ are intermediate velocity fields that progressively incorporate the nonlinear terms and the incompressibility constraint. The motivation for the splitting is to decouple the pressure term from the advection and diffusion terms.

The classical splitting scheme proceeds by introducing two assumptions: that $\boldsymbol{u}^{(2)}$ satisfies the divergence free condition ($\nabla \cdot \boldsymbol{u}^{(2)} = 0$), and that $\boldsymbol{u}^{(2)}$ satisfies the correct Dirichlet boundary conditions in the direction normal to the boundary ($\boldsymbol{n} \cdot \boldsymbol{u}^{(2)} = \boldsymbol{n} \cdot \boldsymbol{u}^{n+1}$). Incorporating these assumptions, we can derive a separately solvable elliptic problem for the pressure in the form:

$$\nabla^2 \tilde{P} = \frac{1}{\Delta t}(\nabla \cdot \boldsymbol{u}^{(1)}). \tag{21}$$

The field $\tilde{P}$ is no longer associated with thermodynamic pressure and becomes a dynamic variable that couples the divergence-free condition and the momentum equation. Neumann boundary conditions for $\tilde{P}$ come from (19), which can be simplified to the form:

$$\frac{\partial \tilde{P}}{\partial \boldsymbol{n}} = \boldsymbol{n} \cdot [\mathbf{N}(\boldsymbol{u}^n) - \frac{1}{Re} \nabla \times \nabla \times \boldsymbol{u}^n]. \tag{22}$$

This boundary condition prevents the propagation and accumulation of time differencing errors and ensures that $\tilde{P}$ satisfies the important pressure compatibility condition [13]. Note that the linear term in (22) is derived from $\mathbf{L}(\boldsymbol{u}^n)$ rather than $\mathbf{L}(\boldsymbol{u}^{n+1})$. This type of first-order extrapolation is necessary to keep the pressure equation decoupled from the other substeps. The order of the extrapolation should be consistent with the overall time accuracy.

A single time step using the skew-symmetric form of the nonlinear terms requires the computation of various spatial derivatives to assemble the nonlinear term plus the solution of one Poisson equation for the pressure and up to three Helmholtz equations for the diffusion in each direction. Most of the computational work is associated with solving these linear systems; integration of the nonlinear terms makes only a minor contribution. The techniques outlined below can be applied directly to the solution of the various elliptic subproblems as well as computation of the nonlinear terms.

Higher-order schemes

It is relatively easy to make the integration scheme outlined above more accurate in time, i.e. to increase the time accuracy to $O(\Delta t^J)$. The basic idea is to use higher-order multi-step schemes for the time integration. Time derivatives can be approximated with a backward difference of the form:

$$\partial_t \boldsymbol{u} \approx \Delta t\, (\gamma_0 \boldsymbol{u}^{n+1} - \sum_{q=0}^{J-1} \alpha_q\, \boldsymbol{u}^{n-q}), \tag{23}$$

where $\gamma_0 = \sum \alpha_q$ for consistency. The nonlinear term can be integrated using an Adams–Bashforth method:

$$\int_{t_n}^{t^{n+1}} \mathbf{N}(\boldsymbol{u})\, \mathrm{d}t \approx \Delta t \sum_{q=0}^{J-1} \beta_q\, \mathbf{N}(\boldsymbol{u}^{n-q}), \tag{24}$$

where $\sum \beta_q = 1$. The pressure boundary conditions should be integrated with a scheme of the same order to ensure consistent time accuracy:

$$\frac{\partial \tilde{P}}{\partial \boldsymbol{n}} = \boldsymbol{n} \cdot \sum_{q=0}^{J-1} \beta_q \, [\mathbf{N}(\boldsymbol{u}^{n-q}) - \frac{1}{Re} \nabla \times \nabla \times \boldsymbol{u}^{n-q}]. \tag{25}$$

Combining these various integration schemes produces the following semi-discrete equations:

$$\boldsymbol{u}^{(1)} - \sum_{q=0}^{J-1} \alpha_q \boldsymbol{u}^{n-q} = \Delta t \sum_{q=0}^{J-1} \beta_q \, \mathbf{N}(\boldsymbol{u}^{n-q}), \tag{26a}$$

$$\boldsymbol{u}^{(2)} - \boldsymbol{u}^{(1)} = -\Delta t \, \nabla \tilde{P} \tag{26b}$$

$$\gamma_0 \boldsymbol{u}^{n+1} - \boldsymbol{u}^{(2)} = \Delta t \, \mathbf{L}(\boldsymbol{u}^{n+1}). \tag{26c}$$

1.2.2 Spectral Element Methods

As stated above the key steps in solving the Navier-Stokes equations are the approximation of the various operators (both linear and nonlinear) and the solution of the Possson equation for the pressure. In this section we will lay out a solution to both of these problems that utilizes high order finite element or spectral element methods. The advantage of this approach is that we can addres issues of accuracy as well as complex geometry. Classical formulations of discrete solutions of the Navier-Stokes equations that are obtained via the use of lower order finite difference methods can be recovered using this formulation through the use of low order basis functions and appropriate projection operators.

A 1-D example

It turns out that all the key aspects of the spatial approximation schemes can be described by considering the solution in one space dimension of the Poisson equation.

Suppose we want to find u such that

$$u'' + f = 0 \quad \text{on } \Omega, \tag{27}$$

where Ω is the unit interval $0 \leq x \leq 1$ and f is a given smooth function. At the endpoints we will specify the boundary conditions

$$u(0) = g, \tag{28a}$$

$$u'(1) = h. \tag{28b}$$

This defines the *strong* form, the usual starting point for finite difference and other schemes.

Consider the following alternative formulation of the same problem. We begin with the equation for the residual,

$$R(u) = \int_\Omega w(u'' + f) \, \mathrm{d}x, \tag{29}$$

from which we want to find the unique function u that drives the residual to zero. The search will include all functions satisfying the boundary condition $u(0) = g$; each candidate is called a *trial* solution, and we denote the set of all trial solutions by $\mathcal{S}$. The residual is orthogonalized with respect to a second set of functions $w \in \mathcal{V}$ called test functions or *variations*. Each test function should satisfy $w(0) = 0$. To incorporate the Neumann boundary condition we integrate equation (29) once by parts, finding that $R(u) = 0$ if

$$\int_\Omega w'u' \,\mathrm{d}x = \int_\Omega wf \,\mathrm{d}x + w(1)h. \tag{30}$$

If we identify the symmetric, bilinear forms $a(w,u) = \int_\Omega w'u' \,\mathrm{d}x$ and $(w,f) = \int_\Omega wf \,\mathrm{d}x$, then we can state the *weak* form as follows: find $u \in \mathcal{S}$ such that for every $w \in \mathcal{V}$

$$a(w,u) = (w,f) + w(1)h. \tag{31}$$

Galerkin approximation solves (31) using a finite collection of functions: find $u^h \in \mathcal{S}^h$ such that for every $w^h \in \mathcal{V}^h$

$$a(w^h,u^h) = (w^h,f) + w^h(1)h. \tag{32}$$

This method reduces an *infinite*-dimensional problem to an n-dimensional problem by choosing a set of n basis functions $(\phi_1, \phi_2, \ldots, \phi_n)$ to represent each member of $\mathcal{S}^h$ and $\mathcal{V}^h$. It admits all linear combinations $w^h \in \mathcal{V}^h$ as $w^h = c_1\phi_1 + c_2\phi_2 + \ldots + c_n\phi_n$, where each $\phi_p(0) = 0$. To generate the trial solutions we need one additional function satisfying $\phi_{n+1}(0) = 1$ so that if $u^h \in \mathcal{S}^h$ then

$$u^h = g\phi_{n+1} + \sum_{p=1}^{n} d_p\phi_p. \tag{33}$$

Note that with the exception of ϕ_{n+1}, $\mathcal{S}^h$ and $\mathcal{V}^h$ are composed of the same functions.

Substituting u^h for u and w^h for w, the weak form becomes

$$\sum_{p=1}^{n} c_p G_p = 0, \tag{34}$$

where

$$\begin{aligned} G_p &= \sum_{q=1}^{n} [a(\phi_p,\phi_q)d_q \\ &\quad -(\phi_p,f) - \phi_p(1)h + a(\phi_p,\phi_{n+1})g]\,. \end{aligned} \tag{35}$$

Since this must be true for any choice of the c_p's, we require $G_p \equiv 0$. If we put the coefficients d_p into a vector $\mathbf{d}$, it becomes the matrix problem

$$\mathbf{A}\mathbf{d} = \mathbf{F}, \tag{36}$$

where the matrix entries are given by $A_{pq} = a(\phi_p, \phi_q)$ and the components of the vector $\mathbf{F}$ are $F_p = (\phi_p, f) + \phi_p(1)h - a(\phi_p, \phi_{n+1})g$. The solution is $\mathbf{d} = \mathbf{A}^{-1}\mathbf{F}$. Quite literally, this is a best fit of the approximate solution u^h to the true solution u based on the measure of error given in equation (29).

Basis functions

Galerkin approximation is "optimal" in the sense that it gives the best approximation in the restricted space $\mathcal{S}^h$. If the true solution u lies in the intersection of $\mathcal{S}^h$ and $\mathcal{S}$, then $u^h = u$. But the success of the method lies in the selection of the basis functions. If they are too complicated it will be impossible to generate the matrix problem, too simple and they cannot adequately describe the true solution u. The key is to combine computability and accuracy. Spectral elements accomplish this in the following manner.

First, the domain is partitioned into K non-overlapping subintervals, where each subinterval, or *element*, is given by $\Omega^k = [a^k, b^k]$. On element k we want to introduce a set of local functions that provide accuracy of order N for the solution over that piece of the computational domain. For spectral element methods, the basis functions are invariably polynomials.

Often, the most convenient approach is to form a set of polynomials from the Lagrangian interpolants through a particular set of *nodes*. Recall that the Lagrangian interpolant takes the value one at some node x_i and is zero at all other nodes. The simplest set of nodes would be the equally spaced points $x_i = a^k + (b^k - a^k)\, i/N$. Of course, this turns out to be a terrible choice for a high-order method because the basis is almost linearly dependent, resulting in ill-conditioned algebraic systems. It is not the choice of Lagrangian interpolants but the choice of nodes we define them over, so to fix the problem we just need to choose a "good" set of nodes. The choice of points is crucial to the success and accuracy of the spectral method. In constrast this close connection between the sampling points and the order of the method is not present in finite difference methods.

To standardize the basis, we introduce a parent domain with the coordinates $-1 \le \xi \le 1$, and a coordinate transformation to the elemental nodes as

$$x_i = a^k + \frac{b^k - a^k}{2}(1 + \xi_i). \tag{37}$$

Now we choose the nodes ξ_i to be the solutions of $(1 - \xi^2)\, L_N'(\xi) = 0$, where $L_N(\xi)$ is the Legendre polynomial of degree N. With this special choice, the Lagrangian interpolants can be written down explicitly as

$$\phi_i(\xi) = -\frac{(1 - \xi^2)\, L_N'(\xi)}{N(N+1)\, L_N(\xi_i)\, (\xi - \xi_i)}. \tag{38}$$

These polynomials are called the Gauss–Lobatto Legendre (GLL) interpolants. Figure 1.1 illustrates the mesh and basis functions for a typical element. We will refer to any basis defined this way as a *nodal* basis.

There are several important reasons for choosing this set of polynomials. First, the expansion of any smooth function using the GLL interpolants, $u \approx u^h = \sum d_i \phi_i(x)$, converges exponentially fast, as can be demonstrated by singular Sturm–Liouville theory [9]. Because these are Lagrangian interpolants, the coefficients d_i are simply the nodal values of the approximate solution: $d_i = u^h(x_i)$. Also, there is a set of integration weights ρ_i associated with

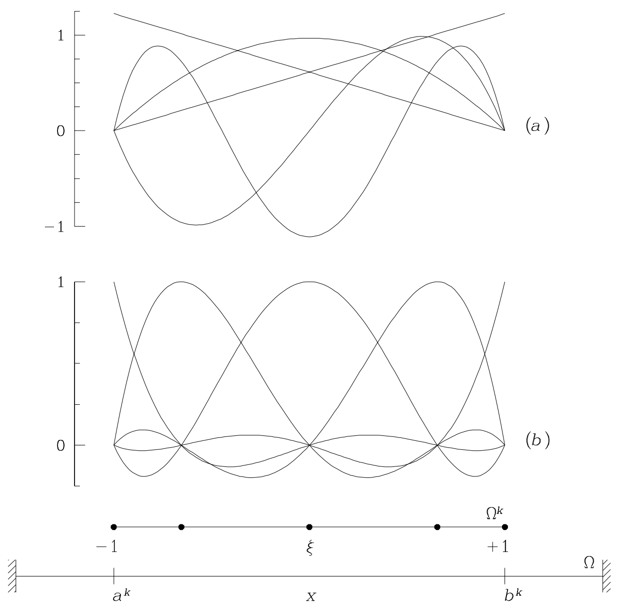

Figure 1.1: One-dimensional spectral element basis functions for an expansion order of $N = 4$, along with a sketch of the local and global coordinate systems: (*a*) modal basis constructed from $P_n^{1,1}(\xi)$; (*b*) Gauss–Lobatto Legendre basis and the set of nodal points that define them as Lagrangian interpolants.

the nodes ξ_i so that the integrals appearing in the weak form can be computed via the GLL quadrature

$$\int_{-1}^{1} f \, \mathrm{d}\xi = \sum_{i=0}^{N} \rho_i f(\xi_i) + \epsilon_N, \tag{39}$$

where the error $\epsilon_N \sim \mathcal{O}(f^{2N}(\zeta))$ for some point $-1 \leq \zeta \leq 1$; as long as the integrand is a polynomial of degree less than $2N$ this quadrature rule is exact [7]. Finally, and perhaps most importantly, the interpolants, quadrature points, and weights can be generated within a computer program by recursive algorithms that are numerically stable through values of $N \sim 100$, eliminating the need to store static tables of quadrature data.

Legendre polynomials are one example of a broad polynomial class called the *generalized Jacobi polynomials*, which we denote as $P_n^{\alpha,\beta}(\xi)$. Legendre polynomials correspond to the parameter values $\alpha = 0, \beta = 0$. Sometimes, especially in higher dimensions and on more complex domains, it is more convenient to work directly with the polynomials rather than an intermediate Lagrangian basis. Jacobi polynomials have the orthogonality property

$$\int_{-1}^{1} (1-\xi)^\alpha (1+\xi)^\beta P_i^{\alpha,\beta}(\xi) P_j^{\alpha,\beta}(\xi) \, \mathrm{d}\xi = \delta_{ij}. \tag{40}$$

We can use Jacobi polynomials directly to represent a function through the expansion $u^h = \sum d_i P_i^{\alpha,\beta}(x)$. The values d_i are the coefficients of the basis functions but they do not correspond to any set of nodal values. In practice, there is a significant advantage if most of the basis functions are orthogonal, so in the one-dimensional case we would use:

$$\begin{array}{rcl} \phi_0(\xi) & = & \frac{1}{2}(1+\xi), \\ \phi_1(\xi) & = & \frac{1}{2}(1-\xi), \\ \phi_i(\xi) & = & \frac{1}{4}(1+\xi)(1-\xi)P_{i-2}^{1,1}(\xi), \quad i \geq 2. \end{array} \tag{41}$$

Figure 1.1 shows the first five basis functions constructed this way. In the nodal basis every function is a polynomial of degree N. In the modal basis there is a *hierarchy* of modes starting with the linear modes, proceeding with the quadratic, the cubic, and so on.

We will refer to spectral elements constructed from a nodal basis as *Lagrange* spectral elements and to those based on a modal basis as *h-p* elements. The latter were first introduced in the early seventies by Szabo [20] who used the integrals of Legendre polynomials as a modal basis, taking $\phi_i(\xi) = \int_{-1}^{\xi} P_{i-1}^{0,0}(s)\,\mathrm{d}s$. However, using the properties of Jacobi polynomials [1] we obtain

$$2n \int_{-1}^{\xi} P_{n-1}^{0,0}(s)\,\mathrm{d}s = (1-\xi)(1+\xi)P_{n-2}^{1,1}(\xi), \tag{42}$$

which is the same as the basis in equation (41) except for the normalization.

The choice of which approach to take is somewhat arbitrary since a nodal basis can always be transformed to an equivalent modal basis and vice versa. The Fast Fourier Transform (FFT) is one familiar example of such a transformation onto the basis $\phi_k(\xi) = \exp(ik\xi)$. Unfortunately, there are no “fast transform” methods for Jacobi polynomials and the transforms require matrix multiplication. However, for the values of N used in practice ($N \leq 16$) this is not a serious drawback. For the remainder of this section we will work with the GLL polynomials, but when we introduce the basis on triangular and tetrahedral subdomains we will switch back to the modal point of view.

Discrete equations

Returning to the problem of solving equation (32), we begin by noting that the integral can be broken into a sum of integrals of each element:

$$a(\phi_p, \phi_q)_\Omega = \sum_{k=1}^{K} a(\phi_p, \phi_q)_{\Omega^k}.$$

Since each basis function is non-zero over a *single* element, the inner product $a(\phi_p, \phi_q)$ is non-zero only if ϕ_p and ϕ_q “belong” to the same element. This makes the global system sparse, and allows us to compute only local matrices. Because of the origin of finite element methods in computational mechanics,

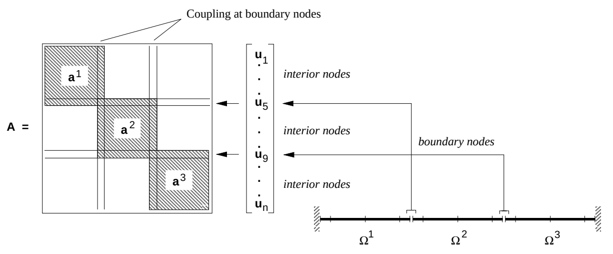

Figure 1.2: Schematic of the direct stiffness summation of local matrices A^k to form the global matrix $\mathbf{A}$.

these matrices are traditionally called:

$$\begin{array}{lcl} \text{“mass”} \quad \mathbf{M}^k_{pq} & = & \int_{\Omega^k} \phi_p \phi_q \, \mathrm{d}x, \\ \text{“stiffness”} \quad \mathbf{A}^k_{pq} & = & \int_{\Omega^k} \phi'_p \phi'_q \, \mathrm{d}x. \end{array}$$

To construct the right-hand side of the matrix system, $f(x)$ is approximated by collocation at the nodal points to produce $f^h(x)$; the mass matrix provides the coefficients necessary to perform the integration. Now the *elemental* matrix system may be written as

$$\mathbf{A}^k \mathbf{v}^k = \mathbf{F}^k \quad (+ \text{ boundary terms}). \tag{43}$$

Just as the integral over the entire domain can be written as a sum of the integral over each element, the global matrices can be computed by summing contributions from the elemental matrices:

$$\mathbf{A} = {\sum_{k=1}^{K}}' \mathbf{A}^k, \quad \mathbf{M} = {\sum_{k=1}^{K}}' \mathbf{M}^k. \tag{44}$$

The symbol $\sum'$ represents “direct stiffness summation,” the procedure diagrammed for the nodal basis in figure 1.2 that maps contributions from the boundary node shared by adjacent elements to the same row of the global matrix $\mathbf{A}$. The global matrix system is

$$\mathbf{A}\,\mathbf{v} = \mathbf{F} \quad (+ \text{ boundary terms}). \tag{45}$$

$\mathbf{A}$ is banded as a result of using local basis functions, with all of its non-zero entries located in the N diagonals above and below the main diagonal. It is also symmetric, due to the symmetry of $a(\cdot,\cdot)$, and positive-definite. Thus $\mathbf{A}$ can be computed, stored, and factored economically and efficiently.

Spectral element discretizations encompass both spectral methods and finite elements. With the proper choice of basis functions and projection methods, finite difference methods can also be included. Standard approximation error estimates for Galerkin methods applied to elliptic problems on quasi-uniform meshes predict that

$$||u - u^h||_1 \leq \text{const.} \times h^{\mu-1} N^{-(k-1)} ||u||_k, \tag{46}$$

where $\mu = \min(k, N+1)$, N is the polynomial degree appearing in the basis functions, and h is a parameter related to the element size [3]. The constant depends on the degree of mesh quasi-uniformity. There are two ways to improve the approximation: make h smaller ($K \to \infty$), or make N and μ larger ($N \to \infty$). The latter results in *exponential* convergence for smooth solutions. If a solution varies rapidly over a small region, any polynomial fit will oscillate rapidly and the best approach is to reduce the element size until the solution is resolved *locally*. A more effective approach is to combine the two convergence procedures, increasing both K and N simultaneously; this dual path of convergence is known as an *h-p* refinement procedure [20]. The flexibility to adapt the mesh to the solution makes spectral element methods quite robust.

Basis functions in d-dimensions

A key to the efficiency of high-order methods in two- and three-dimensional problems is the formation of a basis from the *tensor product* of one-dimensional functions. Among other things, this allows the computation of integrals and derivatives of the basis functions to be simplified through a procedure called *sum factorization* [17]. It also contributes to the sparse structure of matrix systems for multi-dimensional problems.

In this section we describe the procedure for constructing an efficient, high-order basis on two- and three-dimensional domains. To keep the discussion simple, we only consider the standard domains R^d, where d is the problem dimension. Figure 1.3 defines the standard rectangle, R^2. "Standard" here means that the coordinates are normalized to fall in the range -1 to 1. For $d = 3$, the standard domain is a hexahedral element. Isoparametric mappings can always be used to transform more general elements to these standard domains, as illustrated in figure 1.3. On the standard element, we wish to define a polynomial basis, denoted by $\phi_{ij}(\xi_1, \xi_2)$, so that we can represent a function $u^h(\xi_1, \xi_2)$ by the expansion

$$u^h(\xi_1, \xi_2) = \sum_{i=0}^{N} \sum_{j=0}^{N} u_{ij} \phi_{ij}(\xi_1, \xi_2),$$

where u_{ij} is the coefficient of the basis function ϕ_{ij} and $\boldsymbol{\xi} = (\xi_1, \xi_2)$ is the local coordinate within the element.

For quadrilateral (two-dimensional) and hexahedral (three-dimensional) elements, the procedure is straightforward. For example, on the domain $\Omega^k = R^2$, the basis would be

$$\phi_{ij}(\xi_1, \xi_2) = \phi_i(\xi_1)\, \phi_j(\xi_2),$$

where $\phi_i(\xi)$ is the one-dimensional GLL polynomial defined in § **??**. In this case, u_{ij} represents the function value at the node $\boldsymbol{\xi}_{ij}$. The three-dimensional basis on R^3 is exactly analogous to this one.

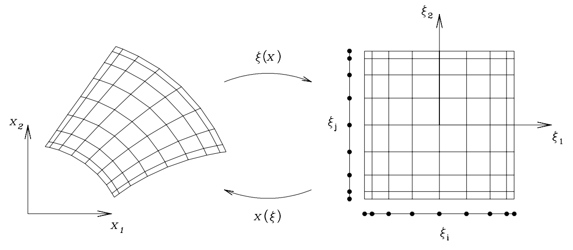

Figure 1.3: Definition of the standard quadrilateral domain R^2. General curvilinear elements can always be mapped back to the standard element as shown.

In the remainder of this Chapter we will use the following simplified notation. Every index (ijk) in the tensor product basis will be mapped to a single number as $p = i + jN + kN^2$, so there is a one-to-one correspondence between $\phi_p(\boldsymbol{\xi})$ and $\phi_{ijk}(\boldsymbol{\xi})$. This hides the tensor product nature of the basis but makes the discrete equations much easier to write down. When necessary, we can "unroll" the p index to take advantage of the tensor product form. This expression for p is valid for quadrilateral elements only; a modified expression should be used with the triangular domains.

1.2.3 Basic operations

Integration

The general form for the evaluation of an integral by Gaussian quadrature with weights $(1-\xi)^\alpha(1+\xi)^\beta$ can be written as

$$\int_{-1}^{1}(1-\xi)^\alpha(1+\xi)^\beta u(\xi)\,\mathrm{d}\xi = \sum_{i=0}^{N}\rho_i^{\alpha,\beta}u(\xi_i^{\alpha,\beta}),$$

where $\xi_i^{\alpha,\beta}$ and $\rho_i^{\alpha,\beta}$ are the quadrature points and weights associated with the Jacobi polynomial $P_N^{\alpha,\beta}(\xi)$. The quadrature rule is exact if $u(\xi)$ is a polynomial of degree $2N+1$ for the Gauss points, $2N$ for the Gauss–Radau points, and $2N-1$ for the Gauss–Lobatto points.

To integrate a function defined over the standard domain R^2, we simply use the tensor product form to reduce the integral to two one-dimensional quadratures. The integral of a general function is written as

$$\int_{R^2} u(\boldsymbol{\xi})\,\mathrm{d}\xi_1\mathrm{d}\xi_2 = \sum_{i=0}^{N}\sum_{j=0}^{N}\rho_i\rho_j u(\boldsymbol{\xi}_{ij}).$$

The extension to integrals over R^3 is straightforward.

Projection

To apply the integration rules described above, we need to evaluate a function at a given set of quadrature points. For the nodal basis this is trivial because the basis coefficients *are* the function values at the quadrature points. For a modal basis we need an efficient way to evaluate the full solution at the quadrature points. This, and the related problem of determining the modal expansion coefficients from a set of nodal values, are both called *projections*.

A projection is the procedure for determining the coefficients u_{ijk} so that $u^h \approx u$ for some given function u. First, recall the general form of the expansion:

$$u(\boldsymbol{\xi}) \approx u^h(\boldsymbol{\xi}) = \sum_p u_p \, \phi_p(\boldsymbol{\xi}).$$

The expansion coefficients are determined by taking the inner-product with the basis functions on both sides of this equation:

$$(u, \phi_p)_{\Omega^k} = (u^h, \phi_p)_{\Omega^k} \quad \forall \phi_p \in \{\phi_{ijk}\}. \tag{47}$$

Solving this system of equations to determine the approximation u^h is straightforward if the basis $\{\phi_{ijk}\}$ is orthogonal. Otherwise, we have to compute u^h by inverting a matrix.

To describe this for the modal basis, we introduce the following notation:

$\mathbf{u}_p$	$=$	vector of $P \sim N^3$ expansion coefficients, $\mathbf{u}_p \leftarrow u_{ijk}$;
$\tilde{\mathbf{u}}_q$	$=$	vector of Q function values at the quadrature points, $\tilde{\mathbf{u}}_q \leftarrow u(\boldsymbol{\xi}_q)$;
$\mathbf{W}_{qq}$	$=$	diagonal matrix of $Q \times Q$ quadrature weights required to integrate a function over Ω^k;
$\mathbf{B}_{qp}$	$=$	rectangular matrix containing the value of the basis functions at the quadrature points (Q quadrature points $\times$ P basis functions).

Now we can write down the algebraic form of the inner-products given in (47). First, the inner product of u with the basis functions:

$$(u, \phi_p)_{\Omega^k} \to \mathbf{B}^T \mathbf{W} \, \tilde{\mathbf{u}}.$$

Second, the inner product of u^h with the basis functions:

$$(u^h, \phi_p)_{\Omega^k} \to \mathbf{B}^T \mathbf{W} \mathbf{B} \, \mathbf{u}.$$

The approximation $u^h \approx u$ is determined by matching these two inner products for every basis function:

$$\mathbf{B}^T \mathbf{W} \, \tilde{\mathbf{u}} = \mathbf{B}^T \mathbf{W} \mathbf{B} \, \mathbf{u}. \tag{48}$$

This is the fully discrete form of (47). Note that the epression on the right-hand-side defines the mass matrix $(\phi_i, \phi_j)_{\Omega^k} \rightarrow \mathbf{B}^T\mathbf{W}\mathbf{B}$, or simply $\mathbf{M} = \mathbf{B}^T\mathbf{W}\mathbf{B}$.

Now we can define the discrete projection operator as

$$\mathbf{u} = \mathcal{P}(\tilde{\mathbf{u}}) \equiv [\mathbf{B}^T\mathbf{W}\mathbf{B}]^{-1}\mathbf{B}^T\mathbf{W}\tilde{\mathbf{u}}.$$

This is also called the *forward transform* of a function from physical space (nodal values) to transform space (modal coefficients). The discrete *inverse transform* is simply the evaluation of the modal basis at a given set of points:

$$\tilde{\mathbf{u}} = \mathcal{P}^{-1}(\mathbf{u}) \equiv \mathbf{B}\mathbf{u}.$$

Finally we note that in the GLL nodal basis, $\mathbf{M}$ is a *diagonal* matrix. This follows directly from the discrete orthogonality of the basis functions and the fact that $\phi_p(\boldsymbol{\xi}_q) = \delta_{pq}$, where $\boldsymbol{\xi}_q$ are the GLL quadrature points. A diagonal mass matrix is a tremendous simplification since multiplication by $\mathbf{M}^{-1}$ is trivial.

Differentiation

Since the basis is formed from continuous functions, in principle derivatives can be evaluated by simply differentiating the basis functions:

$$\frac{\partial u^h}{\partial \xi_1} = \sum_{ijk} u_{ijk} \frac{\partial \phi_i}{\partial \xi_1}(\xi_1)\phi_j(\xi_2)\phi_k(\xi_3).$$

In practice we only need the derivatives at certain points, namely the quadrature points. Therefore, the solution is first transformed onto an equivalent Lagrangian interpolant basis defined over the quadrature points. We introduce the one-dimensional Lagrangian derivative matrix

$$\mathbf{D}_{ip} \equiv \left.\frac{\mathrm{d}\phi_p}{\mathrm{d}\xi}\right|_{\xi_i}.$$

Rather than $\mathcal{O}(N^3)$ terms, the Lagrangian interpolant basis reduces the summation to an equivalent one-dimensional operation. The coefficient of the derivative, u'_{ijk}, is then given by

$$u'_{ijk} = \sum_{p=0}^{N} \mathbf{D}_{ip} u_{pjk}.$$

Since only $\mathcal{O}(N)$ operations are required per point, it takes $\mathcal{O}(N^3)$ operations to compute all derivatives in R^2, and $\mathcal{O}(N^4)$ operations to compute all derivatives in R^3. In the modal basis, calculation of derivatives is preceded by an inverse transform (to nodal values) and followed by a forward transform (to modal coefficients), therefore increasing the computational cost.

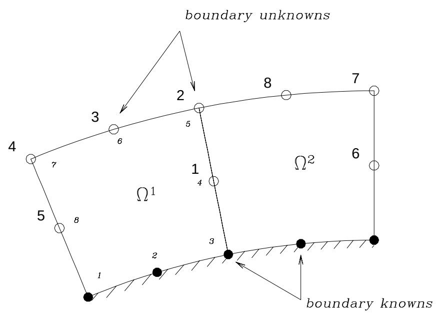

Figure 1.4: Local and global numbering for a simple domain composed of two quadrilateral elements of order $N = 2$. Points along the boundary do not constitute global "degrees of freedom" and are not assigned indices in the global index set.

1.2.4 Global matrix operations

Conforming

One of the basic principles for maintaining the sparse structure in the global matrix systems is to enforce only the minimum continuity between elements. For all of the problems we consider here, the global basis is required to be C^0 continuous, i.e. only function values and not derivatives are required to be globally continuous. For discretizations with both Lagrangian and *h-p* basis functions, this is accomplished by choosing a unique set of global "degrees of freedom" that define the approximation space.

Global continuity in the Lagrangian basis is straightforward. Since the basis functions are defined as the Lagrangian interpolant through the elemental nodes, we only have to use the same set of nodes along the edge of adjacent elements. As long as the elements are conforming (each edge matches up exactly to one other edge) and of equal order (same number of nodes along each edge), C^0 continuity is guaranteed. Figure 1.4 shows a possible global numbering scheme for a simple quadrilateral mesh.

Continuity in the modal basis is more involved because we have to match up all modes. Depending on the orientation chosen for the triangular elements, local modes may be a positive or negative image of the corresponding global mode. This extra bit of information must be tracked as part of the implementation, and we describe it as one use of the mapping matrix $\mathbf{Z}^k$.

Nonconforming

An important extension to the original spectral element method was the introduction of nonconforming elements by Bernardi *et al.* [6]. Here we give only a sketch of the how the method is used to patch together a nonconforming mesh; for a full description of the method, including efficient solution techniques and numerous examples, see the references [2,6,10,11,15].

The main idea is to use a *constrained approximation*. For a geometrically and functionally nonconforming set of elements, we cannot guarantee global C^0 continuity of the basis. Therefore, we make the basis as continuous as possible by minimizing the difference in function values across each nonconforming interface. We do this by enforcing the following weighted residual equation:

$$\int_\Gamma (u - v)\psi \,\mathrm{d}s = 0 \quad \forall \psi \in P_{N-2}(\Gamma). \tag{49}$$

The residual is the difference in two functions u and v that we would like to be continuous, and ψ is the weight used to perform the minimization. The algebraic form of this equation is

$$\mathbf{u} = \mathbf{Z}\,\mathbf{v},$$

where $\mathbf{u}$ and $\mathbf{v}$ are the coefficients of whatever basis we choose to represent u and v, and the entries of $\mathbf{Z}$ are determined by evaluating the residual equation using numerical quadrature. We say the values of $\mathbf{v}$ are free and the values of $\mathbf{u}$ are constrained to match them such that equation (49) is satisfied.

To use this as a computational tool, we choose v to be the solution along the edge of some element, and u to be the solution along the edge of an adjacent nonconforming element. Equation (49) is used to construct u from v, thereby eliminating u as an "unknown" in the mesh. Since v contributes to the global degrees of freedom in the problem, this is one type of the "combining" described in § 1.2.5. There is an additional consistency error associated with the nonconforming discretization because the approximation space is no longer a proper subset of the solution space—it admits discontinuous solutions. As bad as this sounds, the consistency error is of the same order as other components of the approximation error, and if implemented properly the method always converges to a continuous solution if one exists.

Nonconforming elements allow quadrilateral meshes to be refined locally, without the conforming restriction propagating refinement across the mesh. It is not as important for triangular and tetrahedral elements where algorithms such as Rivara refinement [18] can be used to perform local refinement and maintain consistency in the mesh. We will give several examples that make use of nonconforming quadrilateral elements in the following sections.

1.2.5 Data structures

Here we describe the data structures and basic operations required to implement the most common procedures in spectral element methods. We cover representation of the global system, how to transfer global data to local (element) data, direct stiffness summation, and finally the procedures for integration and differentiation of solutions defined on geometrically complex two- and three-dimensional elements.

Implementation

First we start with the representation of the solution within a computer program. In this section we give several examples as pseudo-code fragments that follow basic *C* and *C++* syntax. This is not meant to be an in-depth presentation, but simply an illustration of the most important ideas and the basic approach.

In spectral element methods, as in finite element methods, global data is stored as a flat, unstructured array. The basic data structure used to relate the mesh to entries in this array is a table that identifies the global node number of a local node within each element. Since we are interested in both nodal and modal descriptions, we replace "node" with the more general concept of a "degree of freedom" in the global solution. The table of indices can be stored as a two-dimensional array of integers:

$$\texttt{map[k][i]} \;=\; \begin{array}{l}\text{global index of local datum } i\\ \text{in element } k.\end{array}$$

Local data can be stored in any convenient, regular format. In our first version, we will assume the number of degrees of freedom in the mesh (`ndof`) and the number of degrees of freedom associated with each element (`edof`) are constant. To perform some global operation, for example to evaluate a function $v = F(u)$, we insert a layer of indirection between the unstructured global data and the structured local data. The following is a template for any such computation:

```
for (i=0; i < ndof; i++)      // Initialize v
  v[i] = 0.;
for (k=0; k < nel; k++) {     // Loop over elements
  for (i=0; i < edof; i++)    // Copy global data
    uk[i] = u[ map[k][i] ];   // -- gather
    compute (uk, vk);         // Compute v=F(u) locally
  for (i=0; i < edof; i++)    // Accumulate the result
    v[ map[k][i] ] += vk[i];  // -- scatter
}
```

Depending on the specific operation, the final result may need to be corrected in some way: rescaled with the global mass matrix, averaged based on the data multiplicity, or some similar global operation. The last loop corresponds to direct stiffness summation, and in our matrix notation we would write this same operation as:

$$\mathbf{v} = {\sum_{k=1}^{K}}{}' \mathbf{v}^k = {\sum_{k=1}^{K}}{}' F(\mathbf{u}^k) = F(\mathbf{u}). \tag{50}$$

To make this data structure suitable for both hierarchical bases and nonconforming elements (to be developed in § 1.2.4), we introduce two generalizations.

First, we allow the number of degrees of freedom in each element to be different by replacing the constant `edof` with the array `edof[k]`. Second, we allow each local degree of freedom to depend on an arbitrary combination of the global degrees of freedom. To implement this we need to introduce two new arrays:

`idof[k][i]`	=	number of global dependencies for local datum i in element k,
`combine[k][i]`	=	array of coefficients for combining global data to get local data.

And finally, we need to add a new dimension to our index table:

`map[k][i][j]`	=	global index of the jth dependency of local datum i.

In effect, we are introducing a set of coefficient matrices Z^k that define a general transformation between global and local degrees of freedom. Using this approach, the global initialization, loop over the elements, and function call for the local computation shown above stay the same, but the procedure for constructing the local data is re-written as follows:

```
for (i=0; i < edof[k]; i++)   // Initialize
  uk[i] = 0.;
for (i=0; i < edof[k]; i++) { // Combine
   real *Z  = combine[k][i];
  for (j=0; j < idof[k][i]; j++)
    uk[i] += Z[j] * u[ map[k][i][j] ];
}
```

Likewise, the accumulation of results uses a similar method for combining local contributions to the global degrees of freedom:

```
for (i=0; i < edof[k]; i++) { // Combine
  real *Z  = combine[k][i];
  for (j=0; j < idof[k][i]; j++)
    v[ map[k][i][j] ] += Z[j] * vk[i];
}
```

We also introduce a new matrix notation for this more general approach. Since the local data is $Z^k\mathbf{u}$, and the local contribution to the global system is $[Z^k]^T\mathbf{v}^k$, the equivalent procedure for assembling the global system is written as:

$$\mathbf{v} = {\sum_{k=1}^{K}}'[Z^k]^T\mathbf{v}^k = {\sum_{k=1}^{K}}'[Z^k]^T F(Z^k\mathbf{u}) = F(\mathbf{u}) \tag{51}$$

Compare this to equation (50) above, and note that the only change is how we transform *between* the local and global systems. The actual computations at both the local and global level are the same.

In the remaining sections we will describe computations in terms of either the local or global system, omitting the actual "assembly" required to go between them. Equation (51) is always implied as the method for recovering local solutions and assembling global ones. This simplifies what would otherwise become a confusing barrage of notation. Along the way we will give more specific information about how the coefficients for the mapping matrix Z^k are chosen. This is a very flexible scheme for storing the global solution and reconstructing the local one. The additional storage and computational overhead is simply the price we pay for new capabilities: variable order of the local basis functions and arbitrary connectivity in the mesh. However, these are the key ingredients for adaptive *h*-*p* refinement techniques!

Improvements

Although the scheme outlined above is complete, it is not an efficient way to implement *h*-*p* methods: too much of the addressing is done by indirection. One of the computational advantages of high-order elements is the natural partitioning of data into sets that can be operated on as a group. For example, local degrees of freedom are normally partitioned into several groups: vertices, edges, faces, and interior data. Data associated with any of these groups can be operated on as a single entity. For example, all the points on the interior of an element can be identified with the element number and moved around or computed on as a single unit. High-order elements provide better data locality than low-order elements because computations always involve large amounts of data that can be grouped together in memory.

The type of full indirection outlined above is only necessary for the degrees of freedom associated with the surface of an element. These data make up the loosely-coupled components of the global system. This sparse global system forms the "skeleton" of the discretization and shares many characteristics with low-order finite elements. For example, the numbering system stored in the index table can be optimized to reduce its algebraic bandwidth using the same techniques applied in finite element methods (see §1.2.6). Unfortunately, more sophisticated data structures than can be described here are required to incorporate these simplifications.

1.2.6 Solution techniques

In this section we will describe efficient iterative and direct methods for inverting the large algebraic systems that result from nonconforming spectral element discretizations. Iterative methods are more appropriate for steady-state calculations or calculations involving variable properties, such as a changing time step or a Helmholtz equation with a variable coefficient. For direct methods the issue is one of memory management — storing $\mathbf{A}$ as efficiently as possible without sacrificing the performance needed for fast back-substitution. The development of fast direct and well-preconditioned iterative solvers represents a major advance towards the application of nonconforming spectral element methods to the simulation of turbulent flows on unstructured meshes.

Conjugate gradient iteration

Conjugate gradient methods [5] have been particularly successful with spectral elements because the tensor-product form and local structure allows the global Helmholtz inner product to be evaluated using only elemental matrices. To solve the system $\mathbf{Au} = \mathbf{F}$ by the method of conjugate gradients we use the following algorithm:

```
$k = 0$; $u_0 = 0$; $r_0 = \mathbf{F}$;
while $r_k \neq 0$
    Solve $\mathbf{M}q_k = r_k$ ; $k = k + 1$
    if $k = 1$
        $p_1 = q_0$
    else
        $\beta_k = r_{k-1}^T q_{k-1} / r_{k-2}^T q_{k-2}$
        $p_k = q_{k-1} + \beta_k p_{k-1}$
    end
    $\alpha_k = r_{k-1}^T q_{k-1} / p_k^T \mathbf{A} p_k$
    $r_k = r_{k-1} - \alpha_k \mathbf{A} p_k$
    $u_k = u_{k-1} + \alpha_k p_k$
end
$\mathbf{u} = u_k$
```

where k is the iteration number, r_k is the residual, and p_k is the current search direction. The matrix $\mathbf{M}$ is a preconditioner used to improve the convergence rate of the method and is discussed in detail next.

Selection of a good preconditioner is critical for rapid convergence; the preconditioner must be spectrally close to the full stiffness matrix yet easy to invert. Popular preconditioners for spectral methods include incomplete Cholesky factorization and low-order (finite element, finite difference) approximations [8, 17]. Unfortunately, these preconditioners can be as complicated to construct for an unstructured mesh as the full stiffness matrix $\mathbf{A}$. Next we present three preconditioners which are simple to build and apply even when the mesh is unstructured.

In conjugate gradient methods the number of iterations required to reach a given error level scales as $\sqrt{\kappa_A}$. This is only an estimate, since the actual convergence rate is determined by the *distribution* of eigenvalues — if all of $\mathbf{A}$'s eigenvalues are clustered together, convergence is much faster. To assess the effectiveness of a given preconditioner we begin by looking at the condition number of $\mathbf{M}^{-1}\mathbf{A}$.

Each of the following methods is based on selecting a subset of entries from the full stiffness matrix. The first two preconditioners are diagonal matrices given by

$$M_{ii} = A_{ii} \qquad \text{“diagonal”}, \tag{52}$$

$$M_{ii} = \sum_{j=0}^{n_{\rm dof}} |A_{ij}| \qquad \text{“row-sum”}, \tag{53}$$

where $n_{\rm dof} = \text{rank}(\mathbf{A})$; the diagonal (52) is sometimes called a point Jacobi preconditioner. Both are direct estimates of the spectrum of $\mathbf{A}$, and have the advantage of minimal storage and work. The third preconditioner is a block-diagonal matrix:

$$M_{ij} = \begin{cases} |A_{ij}| & \text{if } i \leq n_{\rm bof},\ j = i \\ 0 & \text{if } i \leq n_{\rm bof},\ j \neq i \\ A_{ij} & \text{otherwise} \end{cases} \tag{54}$$

where $n_{\rm bof}$ is the number of mortar nodes in the mesh. The structure of this matrix assumes that $\mathbf{A}$ is arranged in the static condensation format described in §1.2.6. Applying this preconditioner amounts to storing and inverting the isolated blocks of $\mathbf{A}$ associated with the degrees of freedom on the interior of each element, while applying a simple diagonal matrix to the mortar nodes.

We conclude this section by giving the memory requirements and computational complexity for a preconditioned conjugate gradient (PCG) solver. Since the elemental Helmholtz operator can be evaluated using only the one-dimensional Lagrangian derivative matrix, the required memory is simply storage for the nodal values and geometric factors:

$$S_I = s_1 K N^2. \tag{55}$$

As mentioned above, the dominant numerical operations are vector-vector and matrix-vector products, although derivative calculations are folded into a more efficient matrix-matrix multiplication. The operation count for the entire solver is

$$C_I = J^{\epsilon} \left[c_1 K N^3 + c_2 K N^2 + c_3 K N \right], \tag{56}$$

where $J^{\epsilon} \propto \sqrt{KN^3}$ is the number of iterations required to reach a given error level ϵ. Our numerical results (Tables **??** and **??**) show that with these preconditioners J^{ϵ} is still proportional to KN^3, but the constant is reduced. The block matrix operations required to compute the elemental inner products provide good data locality and can be coded efficiently on both vector processors and RISC microprocessors.

Static condensation

The static condensation algorithm is a method for reducing the complexity of the stiffness matrices arising in finite element and spectral element methods. Static condensation is particularly attractive for unstructured spectral element methods because of the natural division of equations into those for boundaries (mortars) and element interiors. To apply this method to the discrete Helmholtz equation, we begin by writing partitioning the stiffness matrix into boundary and interior points:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^k \begin{bmatrix} \mathbf{u}_b \\ \mathbf{u}_i \end{bmatrix}^k = \begin{bmatrix} \mathbf{F}_b \\ \mathbf{F}_i \end{bmatrix}^k, \tag{57}$$

where A_{11} is the boundary matrix, $A_{12} = [A_{21}]^T$ is the coupling matrix, and A_{22} is the interior matrix. This system can be factored into one for the boundary (mortar) nodes and one for the interior nodes, so that on Ω^k:

$$[A_{11} - A_{21}A_{22}^{-1}A_{12}]\,\mathbf{u}_b = \mathbf{F}_b - [A_{21}A_{22}^{-1}]\mathbf{F}_i, \tag{58a}$$

$$A_{22}\,\mathbf{u}_i = \mathbf{F}_i - A_{21}\mathbf{u}_b. \tag{58b}$$

During a pre-processing phase, the global boundary matrix is assembled by summing the elemental matrices,

$$\mathbf{A}_{11} = \sum_{k=1}^{K}{}'[A_{11} - A_{21}A_{22}^{-1}A_{12}], \tag{59}$$

and prepared for the solution phase by computing its LU factorization. Equation (59) may also be recognized as the Schur complement of A_{22} in A. As part of this phase we also compute and store for each element the inverse of the interior matrix $[A_{22}^{-1}]$ and its product with the coupling matrix $[A_{21}A_{22}^{-1}]$. The system is solved by setting up the modified right-hand side of the global boundary equations, solving the boundary equations using back-substitution, and then computing the solution on the interior of each element using direct matrix multiplication. Because the coupling between elements is only C^0, the element interiors are independent of each other and on a multiprocessor system this final stage can be solved concurrently.

Figure 1.5 illustrates the structure of a typical spectral element stiffness matrix factored using this approach. To reduce computational time and memory requirements for the boundary phase of the direct solver, we wish to find an optimal form of the discrete system corresponding to a minimum bandwidth for the matrix $\mathbf{A}_{11}$. This is complicated by the irregular connectivity generated by the using of nonconforming elements. One approach to bandwidth optimization is to think of the problem in terms of finding an optimal path through the mesh that visits "nearest neighbors." During each of the K stages of the optimization, an estimate is made of the new bandwidth that results from adding one of the unnumbered elements to the current path. The element corresponding to the largest increase is chosen for numbering, resulting in what is essentially a Greedy algorithm. This basic concept is illustrated in figure 1.6. The reduction in bandwidth translates to direct savings in memory and quadratic savings in computational cost. Note that standard methods of bandwidth reduction used for finite elements, e.g. the Reverse Cuthill-McKee algorithm, can also be used, although they only need be applied to the boundary system.

The search for an optimal numbering system can be accomplished during preprocessing, so the extra work has no impact on the simulation cost and can

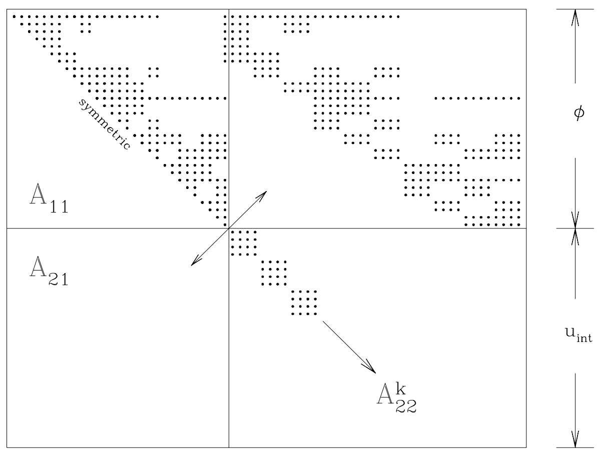

Figure 1.5: Static condensation form of the spectral element stiffness matrix. The vector $\phi = \mathbf{u}_b$ represents the boundary (mortar) solution, while $\mathbf{u}_i$ represents the interior solution.

result in significant savings. For computers where memory is a limitation, this procedure can determine whether an in-core solution is even possible. Other simple memory optimizations include storage of only a single copy of the interior and coupling matrices for each element with the same geometry, and evaluation of the force vector $\mathbf{F}$ using tensor product summation instead of matrix operations. By carefully organizing matrix usage, the overall memory requirement scales as

$$S_D = \frac{1}{2} s_1 K^2 N^2 + s_2 K N^3 + s_3 K N^4. \tag{60}$$

As mentioned in the introduction to this section, the direct solver is advantageous only when the cost of factoring this stiffness matrix can be spread over a large number of solutions. Therefore, we consider only the cost of a back-substitution using the factored stiffness matrix, for which the operation count scales as

$$C_D = c_1 K^{3/2} N^2 + c_2 K N^4 + c_3 K N. \tag{61}$$

For a well-conditioned, diagonally-dominant system this method usually results in at least a factor of two savings versus an iterative solver. For a system that is not diagonally-dominant, like the Navier–Stokes pressure equation, it can be faster by a full order of magnitude.

1.2.7 Adaptive Mesh Refinement

In this section we look at the implementation of a high-order adaptive code based on the nonconforming spectral element method developed in § **??**. In practice this method is used with high-order polynomials ($p \approx 4$ to 16) and

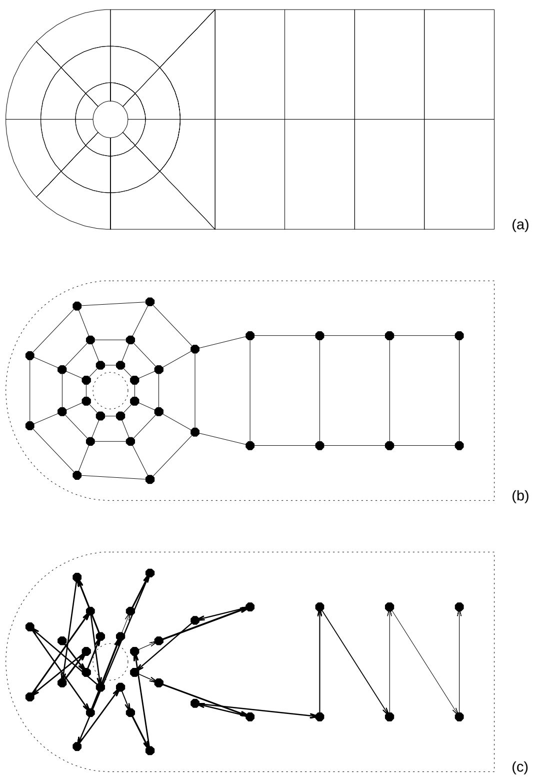

Figure 1.6: Bandwidth optimization for a spectral element mesh: (a) computational domain, (b) connectivity graph and (c) an optimal path for numbering the boundary nodes in the mesh. Line thickness demonstrates the change in global bandwidth with each step.

a mesh of elements that is generated adaptively by h-refinement. We will not attempt to refine both the elements and the basis functions simultaneously as experience indicates that uniformly high p and adaptive mesh refinement leads to an efficient solution for a wide variety of problems.

The formulation based on mortar elements [6] allows completely arbitrary assembly of nonconforming elements. However, our goal is to develop automatic procedures for generating an appropriate mesh and this calls for some compromises. To simplify the encoding of the mesh we will require the refinement to propagate down a quadtree (two-dimensional geometries) or octtree (three-dimensional geometries). A basic description of the mesh generation procedure is provided in §1.2.7. This is found to be a suitable restriction for problems with smooth solutions and leads to a significant reduction in the complexity of the data structure needed to represent the many levels in the refined grid. For

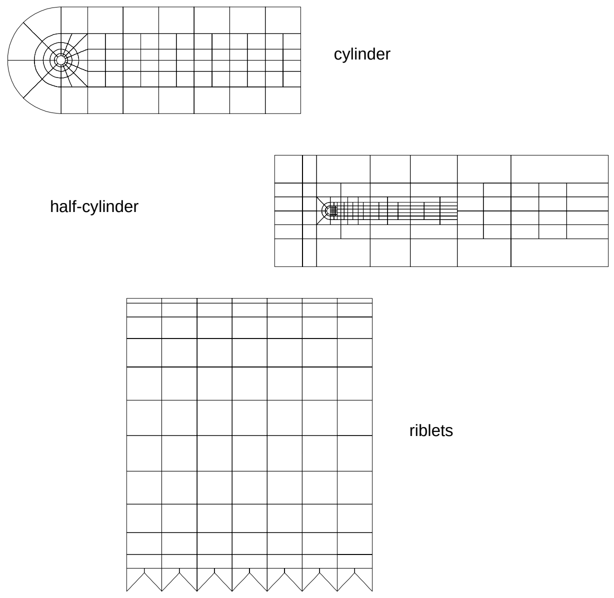

Figure 1.7: Nonconforming meshes used to test the bandwidth optimization.

complex geometries the mesh may incorporate multiple trees at the coarse level.

To give a more specific introduction to the goals of developing an adaptive spectral element method, figure 1.8 shows a sample calculation for the impulsively started flow past a bluff plate. In this simulation the solution field is generated by integrating the incompressible Navier–Stokes equations from an initial state of zero motion. The characteristic scales in the problem are the free-stream speed u_∞, the plate diameter d, and the kinematic viscosity of the fluid ν. The Reynolds number, defined as $Re \equiv u_\infty d/\nu$, is set to the value $Re = 1000$. The lower part of the figure shows the global domain used to represent the flow around the plate. A symmetry condition is imposed along the centerline so that only one half of the flow field needs to be computed. The upper part of the figure is an enlargement of the near wake region. It shows both the vorticity of the developing flow at an early time and the adaptively generated mesh. Each element is an 8×8 point subdomain ($p = 7$) of the global solution. A large number of separate 'trees' are needed at the coarse level to correctly model the beveled geometry of the finite-thickness plate. The initial stage of mesh generation is done by hand to provide the correct starting geometry. Once the problem is handed to the flow solver the additional adaptivity in the mesh is based on a maximum allowable approximation error in the vorticity field.

Because the algorithms for time integration in problems like the one illustrated in figure 1.8 are generally semi-implicit, the computational issues that arise are somewhat different when compared to other methods that incorporate

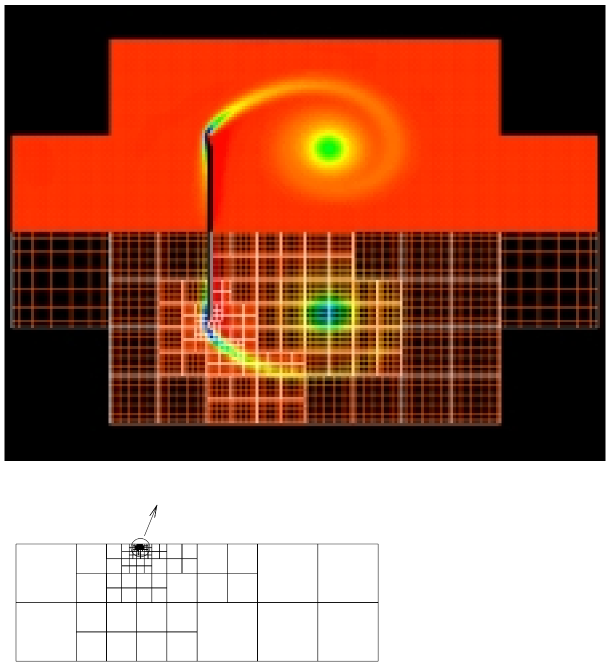

Figure 1.8: Simulation of the impulsively started flow past a bluff plate at $Re = 1000$ using an adaptive spectral element method: (*top*) close-up of the mesh and vorticity of the flow a short time after the impulsive start; (*bottom*) global computational domain.

adaptive meshes. We are interested primarily in studying *incompressible* flows governed by the Navier–Stokes and Euler equations. Because of the elliptic nature of the governing equations (due in part to the incompressibility constraint), local time-stepping is not usually an option. Therefore, solving the elliptic boundary-value problems that arise in these systems is a particular challenge. Even for two-dimensional flows the resolution needed to maintain sufficiently high accuracy can lead to very large systems of equations, and computational efficiency is an important issue. In the past this meant algorithms that could be *vectorized*, while today it means algorithms that can be *parallelized*. There is a close relationship between spectral elements and finite elements, so when it comes to parallel computing many of the same problems (e.g. load balancing) arise, and similar solutions apply. §1.2.8 addresses the implementation of this method for parallel computers with a programming model based on a weakly coherent shared memory which is synchronized via message passing.

Just as important as overall computational performance are the algorithms used for driving adaptive refinement. Ideally such an algorithm would take as input an error estimate and produce as output a new discrete model or mesh that reduces the error. The basic problems are the lack of an error estimate for nonlinear systems and the unlimited ways in which such an algorithm could improve the discrete model. The latter problem is addressed by restricting 'improvements' to propagating refinement down the tree as described in §1.2.7. The former problem is addressed with a pseudo-heuristic error estimate based on the local polynomial spectrum as described in §1.2.7. Depending on the nonlinearity in the partial differential equations being solved, parts of the spectrum will give an accurate approximation to the true solution and parts will be polluted. We estimate the order of magnitude of the local error by examining the decay along the tail of the local polynomial spectrum. In a general sense, this heuristic flags locations in the mesh where the polynomial basis fails to provide a good description of the solution. For simple problems (linear, one-dimensional) this can be formally related to the true difference between the exact solution and the approximate solution, i.e. the approximation error. For more interesting problems it is shown to be a robust guide for driving adaptivity. The heuristic is easy to compute but is only accurate as an error estimate in computations with sufficiently high p, meaning that the local polynomial coefficients should decay like $|a_n| \sim \exp(-\sigma n)$ for $p = n \gg 1$. This is generally not true near singular points (e.g. corners) and these locations are automatically flagged for refinement. The method based on local spectra is compared to simpler heuristics such as refining in regions with strong gradients and the two are shown to lead to quite different results. In general the local spectrum works well and is a good match to the overall computational strategy.

Framework

In this section we restrict our attention to two-dimensional problems. Most of the difficulties arise in two dimensions and there are no fundamental barriers (other than computing power) in extending the method to three dimensions. To begin, let D be some region of space that has been partitioned into K subdomains which we denote $D^{(k)}$. We consider two related problems:

1. Given a discretization tolerance ϵ, generate a spatial discretization $D = \{D^{(k)}\}$ that allows the tolerance to be met;
2. Given a spatial discretization $D = \{D^{(k)}\}$, generate a finite-dimensional approximation $u^h \approx u$. The function u may be given explicitly or implicitly, i.e. as the solution of a boundary-value problem.

Our approach to problem (1) is to create a hierarchy of grids by forming a quadtree partition of D. This provides the computational domain for problem (2) where we apply a nonconforming spectral element method to approximate u^h.

Mesh generation

The mesh generation problem is somewhat simpler, so we describe that first. A quadtree is a partition of two-dimensional space into squares. Each square is a *node* of the tree. It has up to four daughters, obtained by bisecting the square

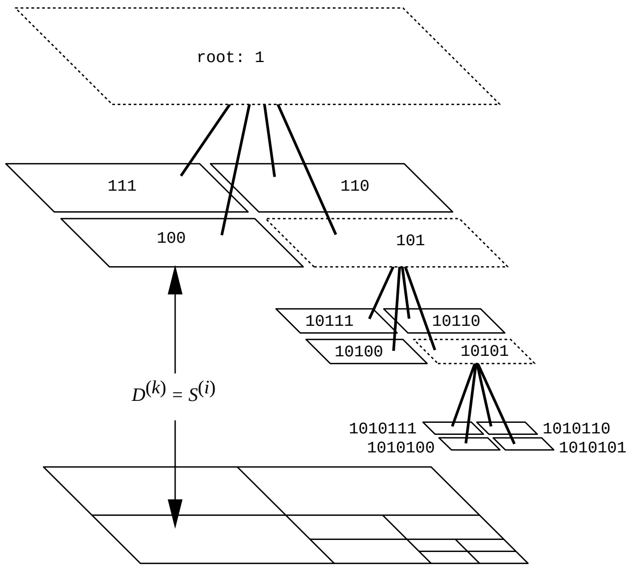

Figure 1.9: A four-level quadtree mesh, expanded to show the elements that make up each level. Each leaf node $S^{(i)}$ has a unique integer key shown in binary. Daughter keys are generated from a parent's key by a two-bit left shift, followed by a binary *or* in the range 00–11. The active elements $D^{(k)}$ that make up the current discretization are shown with a solid outline.

along each dimension. Each node in a quadtree has geometrical properties (spatial coordinates, size) and topological properties (parents, daughters, siblings). Geometrical properties of daughter nodes are inherited from parents, and thus the geometrical properties of the entire tree are determined by the root node.

To represent the topological aspects of the tree we use an idea originally developed for gravitational N-body problems [19]. Every possible square $S^{(i)}$ is assigned a unique integer *key*. The root of the tree is $S^{(1)}$ with key 1. The daughters of any node are obtained by a left-shift of two bits of the parent's key, followed by a binary *or* in the range 00–11 (binary) to distinguish each sibling. A node's parent is obtained by a two bit right-shift of its own key. Since the set of keys installed in the tree at any time is obviously much smaller than the set of all possible keys, a hash table is used for storage and lookup.

From the complete set of nodes in the tree we choose a certain subset $D^{(k)} \subseteq S^{(i)}$ to form the *active* elements of the computational domain. Figure 1.9 shows a four-level quadtree with thirteen nodes and $K = 10$ active elements. Active elements in the figure are shown with a solid outline while inactive elements are shown with a dashed outline. Inactive elements are retained so that they are available for coarsening the mesh, if necessary. The only requirement enforced on the topology of the mesh is that active elements that share a boundary segment live at most one refinement level apart, limiting adjacent elements to a two-to-one refinement ratio. This imposes a certain smoothness on the change in resolution in the mesh that is appropriate for the class of smooth functions we wish to represent.

Refinement criteria

The adaptive mesh generation described above and high-order domain decomposition methods described in §?? are coupled through the refinement criteria used to drive adaptivity. Here we consider three types of refinement criteria.

The first is by far the simplest: refine everywhere that solution gradients are large. We can enforce this idea by requiring

$$\| \nabla u^{(k)} \| \le \epsilon \; \| u^h \|_1 \tag{62}$$

everywhere in the mesh, where $\| \cdot \|$ is the L_2 norm, $\| \cdot \|_1$ is the H^1 norm, and ϵ is the discretization tolerance. This is a common refinement criteria in cases where there is simply no alternative measure of solution errors.

The second type takes direct advantage of the high-order polynomial basis. Consider the expansion of a given smooth function u over the domain $D = [-1,1]^2$ in terms of Legendre polynomials:

$$u(x,y) = \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} a_{n,m} \, P_n(x) P_m(y). \tag{63}$$

The expansion coefficients are given by

$$a_{n,m} = \frac{1}{c_n c_m} \int_{-1}^{1} \int_{-1}^{1} u \, P_n P_m \, |J| \, \mathrm{d}x \, \mathrm{d}y, \tag{64}$$

where the normalization constant is $c_i = (2i+1)/2$. We have included the Jacobian $|J|$ to include the effects of element size and other geometric transformations, e.g. curvilinear boundaries. There is nothing magical about Legendre polynomials—they are simply a convenient orthogonal basis for projecting the approximation onto. Since our approximate solution $u^h \approx u$ is formed essentially by truncating this expansion at some finite order p, we can form an estimate of the approximation error $\| u - u^h \|$ by examining the tail of the spectrum.

To do so we first average over polynomials in x and y to produce an equivalent one-dimensional spectrum:

$$\bar{a}_p = |a_{p,p}| + \sum_{i=0}^{p-1} |a_{i,p}| + |a_{p,i}|. \tag{65}$$

Next we replace the discrete spectrum $\bar{a}_p$ with an approximation to a decaying exponential:

$$\tilde{a}(n) = \text{const.} \times \exp(-\alpha n). \tag{66}$$

The function $\tilde{a}(n)$ is a least squares best fit to the last four points in the spectrum $\bar{a}_p$. Our refinement criteria becomes

$$\left(\tilde{a}(p)^2 + \int_{p+1}^{\infty} \tilde{a}(n)^2 \, \mathrm{d}n \right)^{1/2} \le \epsilon \; \| u^h \| \, . \tag{67}$$

The only practical complication here is making sure the decay rate $\alpha > 0$ so that the integral converges. Otherwise, the estimate is ignored and the element is flagged for immediate refinement. This method is analyzed in [16] where it is shown to be an effective refinement criteria for driving h-p refinement.

The third refinement criteria is similar. Since the main contribution to (67) comes from the coefficients of order p, we can simply sum along the tail of the spectrum. For an accurate representation of u we require the spectrum to satisfy the discretization tolerance:

$$|a_{p,p}| + \sum_{i=0}^{p-1} |a_{i,p}| + |a_{p,i}| \leq \epsilon \parallel u^h \parallel . \tag{68}$$

This method is somewhat simpler to apply and, as we will see, produces almost identical results.

To use these polynomial spectrum criteria with our spectral element method (based on GLL polynomials) we first perform a Legendre transform of the local solution $u^{(k)} \to a_{n,m}$ and then use (67) or (68) to decide if the element should be refined. Although we keep p fixed, the error is reduced because we approximate u over a smaller region $D^{(k)}$.

1.2.8 Implementation for parallel architectures

We end this section with a few additional notes on implementation. The algorithms described above have been implemented using a combination of *C* for the computational modules and *C++* for high-level data types like `Element` $\equiv D^{(k)}$ and `Field` $\equiv u^h$ that make up the discretization. The logic and control structure needed for most of the code are the same as in any algorithm for finite element methods. The most complex problem is maintaining the connectivity of the mesh dynamically, and the approach taken here is worth mentioning.

The geometry and topology of the mesh are closely connected. Figure 1.10 shows the three geometric elements of the discretization: vertices, edges, and interiors. Obviously interior points are completely local to an element and play no role in the global system. All connectivity in the mesh is through the edges and vertices. Because of the method used to construct the grid these geometric elements are interlocking. The midpoint of each nonconforming edge aligns with the shared vertex of its two adjacent elements. As discussed below, this feature is used to simplify the procedure for setting up the mesh topology.

Figure 1.10 shows one other side effect of the mesh generation. Internal curvilinear boundaries are automatically propagated down the various levels of the refinement tree because of the isoparametric representation of the geometry. In the same way that a solution field is projected onto a new set of elements, the polynomial representation of the geometry can also be projected to a finer grid. On the other hand, external boundaries like the B-spline segment shown as the lower boundary in the figure are explicitly re-evaluated to keep the representation as accurate as possible.

How does one represent the topology of this kind of mesh? One solution is to use pointers. This immediately runs into the problem of interpreting pointers to

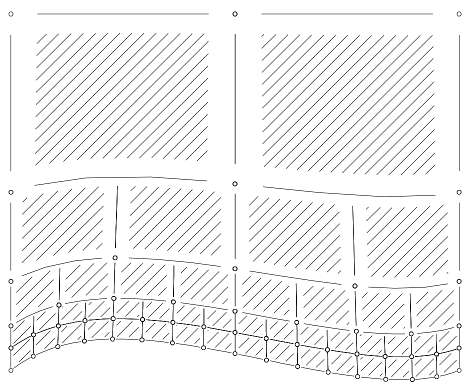

Figure 1.10: The logical structure of a spectral element mesh can be divided into three geometric parts: (∘) vertices, (—) edges, and (*shaded*) interiors. Edges and vertices define the connectivity in the mesh.

objects on remote processors if the computation is running in parallel. Instead we use the concept of a *voxel database* (VDB) of geometric positions in the mesh [22]. A VDB may be thought of as register of position–subscript pairs. To each position stored in the VDB we assign a unique integer subscript so that data may be associated with points in space by using the subscript as an index into an array.

The basic idea is illustrated in figure 1.11. The number of times a position is registered is its *multiplicity.* Data objects that share positions also share memory by virtue of a common subscript. In essence the VDB provides a natural map of the mesh geometry onto the computer's memory. This basic paradigm can be used to implement many types of finite element or finite volume methods [22].

To establish the connectivity of a mesh like the one depicted in figure 1.10 we build two separate VDBs: one for the vertices and one for the midpoints of the edges. Every vertex with multiplicity one that does not lie along an external boundary is *virtual* and not part of the true mesh degrees of freedom. Every edge with multiplicity one that does not lie along an external boundary is *nonconforming.* For each nonconforming edge we make a second query to the VDB using the endpoints. If there is a match then the edge is also *virtual* and we store the subscript of the adjacent edge. Otherwise it is simply flagged as an internal nonconforming boundary segment.

The shared memory represented by a VDB is extended across processor boundaries by passing around a list of local positions and comparing against those registered remotely. A communications link is established for each common position. The shared memory at each point is weakly coherent and must be synchronized by explicit message passing. For example, elements on separate processors with a common boundary segment share data along an edge. Each

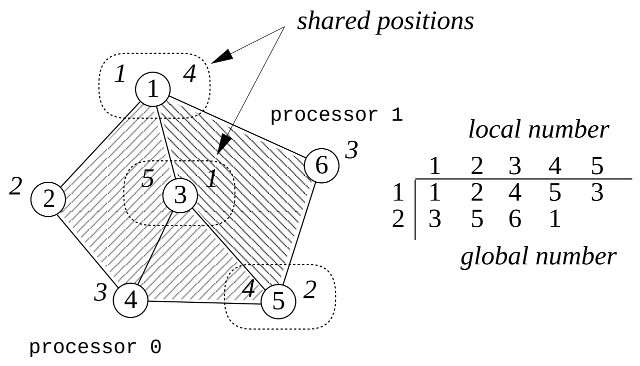

Figure 1.11: Connectivity and communications are established by building a *voxel database* (VDB) of positions. A VDB maps each position to a unique index or subscript. It also tracks points shared by multiple processors to provide a loosely synchronous shared memory. Points that share memory are those at the same geometric position.

processor may update its edge values independently and then call a synchronization routine that combines local and remote values to produce a globally consistent data set. For further details see [22].

There is very little overhead for the adaptive versus non-adaptive data structure: just one integer (the node key) per element. Likewise, an iterative solver for sparse systems incurs no performance penalty just because the underlying mesh is adaptive. When approached in the right way the conversion to a solution adaptive code is almost trivial. To a large degree this is because of the unstructured nature of the spectral element method we built upon.

1.2.9 An Example - the cylinder wake

Understanding the fluid flow around a straight circular cylinder is one of the most fundamental problems in fluid mechanics. It's a model for flow around bridges, buildings, and many other non-aerodynamic objects. Recent work, both experimental and computational, has revealed some exciting new information about the nature of this flow including intricate three-dimensional structures that emerge just prior to the onset of turbulence in the wake.

The system considered is an infinitely long cylinder placed perpendicular to an otherwise uniform open flow. The sole parameter for this system in then the Reynolds number: $Re \equiv U_\infty d/\nu$, where U_∞ is the free-stream velocity and d is the cylinder diameter. First we describe some of the physically important behavior in this flow, and then come back to details of how it can be simulated. It helps to begin with a 'road-map' for the sequence of bifurcations that take the flow from simple to more complex states. There are two useful quantities to form such a guide to understanding: the non-dimensional shedding frequency and the mean drag coefficient C_D. Both shedding frequency and drag show distinct changes at the various bifurcation points of the wake and can be used

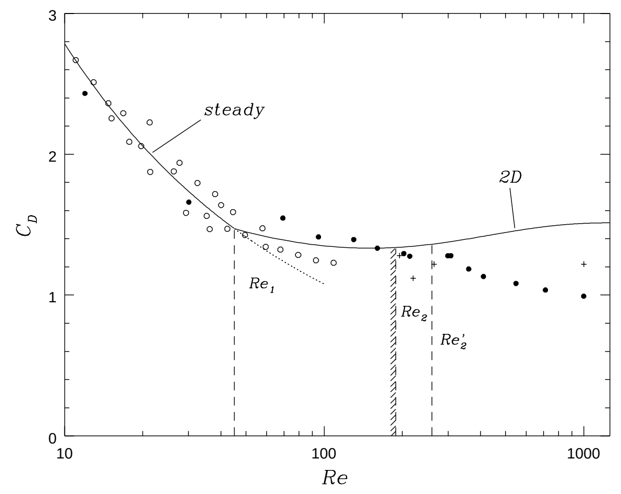

Figure 1.12: Drag coefficient as a function of Reynolds number for the flow past a circular cylinder. Experiments: (∘,•), Wieselsberger [21]; 3D simulations: +, Henderson [12]. The solid line is a curve fit to two-dimensional simulation data for Re up to 1000 [11].

as a guide to interpreting changes in the wake structure and dynamics as a function of Reynolds number.

In non-dimensional form the shedding frequency is referred to as the Strouhal number. It is defined as $St \equiv f\,d/u_\infty$, where f is the peak oscillation frequency of the wake. At low Reynolds number the flow is steady ($St = 0$) and symmetric about the centerline of the wake. At $Re_1 \simeq 47$ the steady flow becomes unstable and bifurcates to a two-dimensional, time-periodic flow. Note that each point along the two-dimensional curve represents a perfectly time-periodic flow and there is no evidence of further two-dimensional instabilities for Reynolds numbers up to $Re \approx 1000$. At $Re_2 \simeq 190$ the two-dimensional wake becomes absolutely unstable to long-wavelength spanwise perturbations and bifurcates to a three-dimensional flow (mode A). Experiments and computations indicate a further instability at $Re_2' \simeq 260$ marked by the appearance of fine scale streamwise vortices.

Figure 1.12 shows the drag curve for flow past a circular cylinder for Reynolds number up to 1000. In the computations the spanwise-averaged fluid force $\mathbf{F}(t)$ is computed by integrating the shear stress and pressure over the surface of the cylinder. The x-component of $\mathbf{F}$ is the drag, the y-component is the lift. Because C_D is determined from an average over the surface of the cylinder, it is much less sensitive to changes in the character of the wake at low Reynolds number than single-point measurements like the shedding frequency. The ‘textbook’ version of the drag curve is generally plotted on a log-log scale where the only discernible feature is the drag crisis at $Re = O(10^5)$. The flat response of C_D to changes in Reynolds number is compounded by the fact that experimental

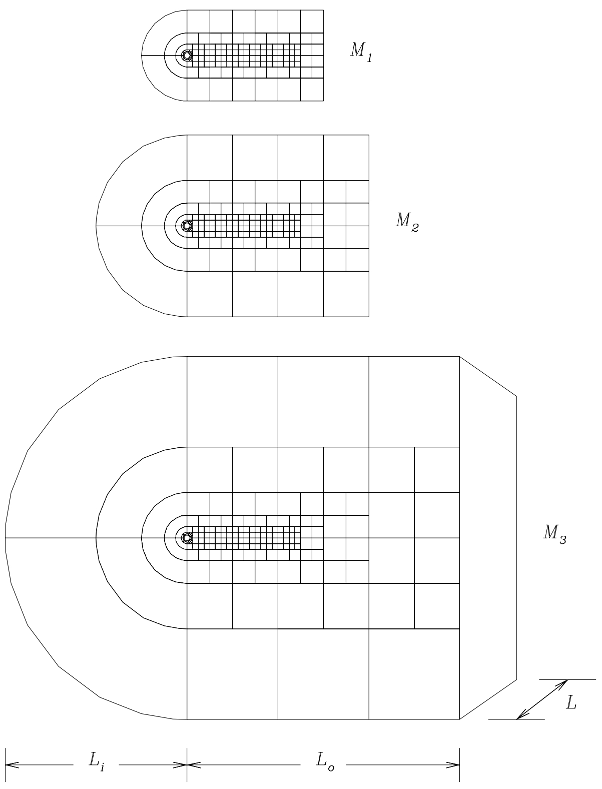

Figure 1.13: Computational domains used for simulating the flow past a circular cylinder. Each domain is a subset of the largest. The parameters L_o and L_i determine the cross-sectional size, and L determines the spanwise dimension.

drag measurements are extremely difficult to make at low Reynolds number, and subtle details of the drag curve are lost in the experimental scatter. The decrease in magnitude of C_D in the steady regime can be fitted to a power-law curve and also makes a sharp but continuous transition at Re_1. Henderson [11] gives the form and coefficients for the steady and unsteady drag curves.

This problem is extremely challenging because it combines several features that are difficult to handle numerically: unsteady separation, thin boundary layers, outflow boundary conditions, and the need for a large computational domain to simulate an open flow. If the computational domain is too small the simulation suffers from blockage. This can have a significant impact on quantities like the shedding frequency, generally producing higher frequencies in the the simulations than are observed in experiments [14]. If resolution near the cylinder is sacrificed for the sake of a larger computational domain then the physically important flow dynamics may not be computed accurately.

Figure 1.13 shows a sequence of computational domains used to simulate both 2D and 3D wakes using nonconforming quadrilateral elements [12]. Boundary conditions are imposed as follows. Along the left, upper, and lower boundaries we use free-stream conditions: $(u_1, u_2, u_3) = (1, 0, 0)$. At the surface of the cylinder the velocity is equal to zero (no-slip). Along the right boundary we use a standard outflow boundary condition for velocity and pressure:

$$p = 0, \quad \partial_x u_i = 0.$$

Along all other boundaries the pressure satisfies (22).

These domains use large elements away from the cylinder and outside the wake where the flow is smooth. Local mesh refinement is used to resolve the boundary layer, near wake, and wake regions downstream of the cylinder. In this case the refinement is done beforehand and the mesh is static. Clearly from figures **??** and 1.12 the simulations predict values of the shedding frequency and drag that agree extremely well with experimental studies up to the point of 3D transition. Just as important as good agreement with experiments, the simulation results are independent of the grid as shown by a detailed h- and p-refinement study [4].

1.3 Compressible Flow

1.3.1 Governing equations of motion

1.3.2 Numerical methods for hyperbolic conservation laws

1.3.3 Details of the numerical method

1.3.4 Application - the Richtmyer-Meshkov instability

1.3.5 Adaptive mesh refinement

1.4 Conclusion

Bibliography

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions.* Dover, 1970.

[2] G. Anagnostou. *Nonconforming Sliding Spectral Element Methods for the Unsteady Incompressible Navier–Stokes Equations.* PhD thesis, M.I.T., February 1991.

[3] I. Babuska and M. Suri. The p and h–p versions of the finite element method, basic principles and properties. *SIAM Review*, 36(4):578–632, 1994.

[4] D. Barkley and R. D. Henderson. Three-dimensional Floquet stability analysis of the wake of a circular cylinder. *J. Fluid Mech.*, 319, 1996.

[5] R. Barrett et al. *Templates* for the solution of linear systems: Building blocks for iterative methods. Text available at `http://www.netlib.org`, 1994.

[6] C. Bernardi, Y. Maday, and A. T. Patera. A new nonconforming approach to domain decomposition: the mortar element method. In H. Brezis and J. L. Lions, editors, *Nonlinear Partial Differential Equations and Their Applications.* Pitman and Wiley, 1992.

[7] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration.* Academic Press, Inc., 1984.

[8] M. O. Deville and E. H. Mund. Chebyshev pseudospectral solution of second order elliptic equations with finite element preconditioning. *J. Comput. Phys.*, 60:517–533, 1985.

[9] D. Gottlieb and S. A. Orszag. *Numerical Analysis of Spectral Methods: Theory and Applications.* SIAM, Philadelphia, 1977.

[10] R. D. Henderson. *Unstructured Spectral Element Methods: Parallel Algorithms and Simulations.* PhD thesis, Princeton University, June 1994.

[11] R. D. Henderson. Details of the drag curve near the onset of vortex shedding. *Phys. Fluids*, 7(9), September 1995.

[12] R. D. Henderson. Nonlinear dynamics and pattern formation in turbulent wake transition. *J. Fluid Mech.*, 352:65–112, 1997.

[13] G. E. Karniadakis, M. Israeli, and S. A. Orszag. High-order splitting methods for the incompressible Navier–Stokes equations. *J. Comput. Phys.*, 97(2):414, 1991.

[14] G. E. Karniadakis and G. S. Triantafyllou. Three-dimensional dynamics and transition to turbulence in the wake of bluff objects. *J. Fluid Mech.*, 238:1, 1992.

[15] C. Mavriplis. *Nonconforming Discretizations and* a Posteriori *Error Estimates for Adaptive Spectral Element Techniques.* PhD thesis, M.I.T., February 1989.

[16] C. Mavriplis. Adaptive mesh strategies for the spectral element method. *Comput. Methods Appl. Mech. Engrg.*, 116:77–86, 1994.

[17] S. A. Orszag and L. C. Kells. Transition to turbulence in plane Poiseuille flow and plane Couette flow. *J. Fluid Mech.*, 96:159, 1980.

[18] M. C. Rivara. Selective refinement/derefinement algorithms for sequences of nested triangulations. *Int. J. Numer. Methods Eng.*, 28:2889–2906, 1989.

[19] J. K. Salmon, M. S. Warren, and G. S. Winckelmans. Fast parallel tree codes for gravitational and fluid dynamical N-body problems. *Int. J. Super. Appl.*, 8(2), 1994.

[20] B. Szabo and I. Babuska. *Finite Element Analysis.* John Wiley and Sons, 1991.

[21] C. Wieselsberger. Neuere Feststellungen über die Gesetze des Flüssigkeits- und Luftwiderstands. *Phys. Z.*, 22:321–238, 1921.

[22] R. D. Williams. Voxel databases: a paradigm for parallelism with spatial structure. *Concurrency*, 4(8):619–636, 1992.

[23] T. A. Zang. On the rotational and skew-symmetric forms for incompressible flow simulations. *App. Num. Math.*, 7:27–40, 1991.