Spectral Element Solutions for the Navier-Stokes Equations on High-Performance Distributed Memory Parallel Processors

Paul F. Fischer

CRPC-TR90082
December, 1990

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Spectral Element Solutions for the Navier-Stokes Equations on High-Performance Distributed Memory Parallel Processors

Paul F. Fischer
Center for Research on Parallel Computation
California Institute of Technology
Pasadena, CA 91125

Abstract

We present a parallel spectral element method for solution of the unsteady incompressible Navier-Stokes equations in general three-dimensional geometries. The approach combines high-order spatial discretizations with iterative solution techniques in a way which exploits with high efficiency currently available medium-grained distributed-memory parallel computers. Emphasis is placed on the development of algorithm constructs which allow for solution of physically relevant problems; we specifically address the problem of parallel solution in domains of general topology. Measured performance analysis on the Intel vector hypercubes and example Navier-Stokes calculations demonstrate that parallel processing can now be considered an effective fluid mechanics analysis tool.

Introduction

To address many problems in fluid mechanics, it is necessary to solve the fully viscous, incompressible, Navier-Stokes equations governing fluid motion at low Mach numbers:

$$\begin{aligned} \frac{\partial \vec{u}}{\partial t} + \vec{u}\cdot\nabla\vec{u} &= -\nabla p + \frac{1}{Re}\nabla^2\vec{u} - \vec{f} \quad in\ \Omega \\ \nabla.\vec{u} &= 0 \quad in\ \Omega \\ \vec{u} &= \vec{u}_{\partial\Omega} \quad on\ \partial\Omega, \end{aligned} \tag{1}$$

where $Re = UL/\nu$ is the Reynolds number based on a characteristic velocity, length, and viscosity. Examples include stability analysis, transitional and fully separated flows, heat transfer applications where boundary layer resolution is important, and direct simulation of turbulence in which convective effects determine the large scale motion while viscous effects dominate the smaller scales. In such instances, high-order techniques have advantages because of their good boundary layer resolution capabilities, their ability to accurately resolve a broad solution spectrum, and their minimal numerical dispersion and dissipation properties [1,2,3].

Despite the ever decreasing clock-cycle time of conventional scalar computers, the ability to solve very large fluid flow problems is still strongly dependent upon the development

and implementation of algorithms which exploit specialized vector and parallel architectures offering both increased computational speed and economy. As a result, there is a need for inter-disciplinary study combining fluid mechanics, numerical analysis, and computer science, in order to optimize performance in the large parameter space spanning discretizations, solvers, and computer architectures. To this end, we present an analysis of a spectral element method for the Navier-Stokes equations implemented on a high-performance, distributed memory, parallel processor. Our target class of problems is general domain, two- and three-dimensional incompressible Navier-Stokes equations at low to moderate Reynolds numbers. The goal is a parallel implementation which retains the full generality and spectral accuracy of its serial processor counterpart, while simultaneously exploiting the economic computing potential offered by currently available parallel processors [4].

The spectral element method [5,6] is a generalized variational scheme which exploits the rapid convergence rates of spectral methods while retaining the geometric flexibility of the finite element techniques. It is based upon a macro- (spectral) element discretization in which degrees-of-freedom internal to elements are coefficients of high-order expansion functions, and for which C^0 continuity is imposed across element boundaries. With an appropriate choice of interpolants and quadrature formulae, it can be shown that the error for problems having smooth solutions will decrease exponentially fast as the order of the expansion, N, is increased [6].

Traditionally, the arguments against the use of high-order discretizations are that they yield high-bandwidth, non-sparse, linear operators and hence require much more computational effort than their low-order counterparts, and that they do not provide sufficient geometric flexibility for problems of engineering interest. However, with increasing computer performance, it is becoming feasible to address three-dimensional Navier-Stokes calculations. In such cases, it proves economical to employ iterative solvers for both low- and high-order methods. While the high-order operators are still full, it has been known for some time that the tensor-product operators associated with spectral methods can be factored into a series of sparse operators - resulting in operation counts which are competitive with low-order methods [7]. In addition, at higher Reynolds numbers, high-order methods become more attractive because the correct, physical, dissipation can be obtained with fewer grid points in each spatial direction; the implication for three-dimensional calculations is a significantly reduced number of equations. As regards geometric flexibility, high-order spectral element and p-type finite element methods are capable of handling moderately complex geometries by virtue of iso-parametric mappings, and are an appropriate approach to discretization for problems in which the length scales of the solution are much smaller than those of the associated geometry, as is typically the case with fluid mechanics problems.

Finally, we note that heterogeneous discretizations such as spectral element methods, or heterogeneous work decompositions such as iterative methods based on classical domain decomposition techniques (e.g., [8]), are appropriate algorithms for modern architectures featuring a memory hierarchy. It is clear that advances in processor speeds are currently outpacing conventional memory access rates and, although new processor technology such as RISC architectures offer the potential of very high execution rates, the performance actually obtained will be largely dependent upon having sufficient memory bandwidth and/or sufficiently dense computations to keep the processors active. Several avenues can be pursued to exploit the performance of these fast CPU's. One is to use distributed memory parallel architectures which not only increase performance through parallel *data processing*, but have the significant advantage of providing parallel *data access*. Another is to employ algorithms relying on dense operations, such as matrix-matrix multiplication, for which the operation count is significantly higher than the required number of data accesses. (This has been the motivation behind the development of the Level 3 BLAS routines, a basic linear algebra package based upon block matrix techniques [9].) It is somewhat paradoxical that, on the one hand, from a parallel view point it is preferrable to have data which is completely independent, while on the other hand, from a register or cache standpoint, it is desirable to have data which is completely interrelated so that it is used several times per access. As such, heterogeneous work decompositions are of further interest.

We turn now to a brief outline of the spectral element discretizations and solvers, followed by a description of the parallel implementation and the associated computational complexity analysis. We conclude with measured performance on the Intel iPSC-VX hypercube parallel processors, and present two representative Navier-Stokes calculations which illustrate that parallel computing can indeed be considered a useful fluid mechanics analysis tool.

Spectral Element Discretizations

Our numerical methods for the Navier-Stokes equations are premised upon a 'layered' approach, in which the discretizations and solvers are constructed on the basis of a hierarchy of nested operators proceeding from the highest to the lowest derivatives. For incompressible viscous flow equations the linear self-adjoint elliptic Laplace operator represents the 'kernel' of our Navier-Stokes algorithm insofar that it involves the highest spatial derivatives. This operator governs the continuity requirements, conditioning and stability of the system. The fully discretized Navier-Stokes equations are typically solved at each time step by performing a series of elliptic solves and preconditioning steps.

As the parallel implementation of the spectral element method is intrinsically tied to the

discretization, we briefly outline the basis of the discretization by considering the solution of a two-dimensional Poisson equation on the domain shown in Fig. 1.

$$\begin{aligned} -\nabla^2 u &= f \quad in\ \Omega, \\ u &= 0 \quad on\ \partial\Omega. \end{aligned} \tag{2}$$

Equation (2) can be equivalently expressed as: Find $u \in \mathcal{H}_0^1(\Omega)$ such that

$$\int_\Omega \nabla\phi \nabla u \, d\Omega = \int_\Omega \phi f \, d\Omega \quad \forall \phi \in \mathcal{H}_0^1(\Omega), \tag{3}$$

where the space $\mathcal{H}_0^1$ is the space of all functions which are zero on the boundary and have a square integrable first derivative. The variational form has the significant advantage that it reduces the required level of continuity on the solution from C^1 to C^0, which in turn has implications as regards parallel communication.

Discretization of the variational statement proceeds by restricting the admissible solutions and trial functions in (3) to a finite-dimensional subspace, X_h, of the infinite-dimensional space, $\mathcal{H}_0^1$. For the spectral element method we choose the space X_h to be:

$$X_h(\Omega) = \{\phi|_{\Omega^k} \in \mathbf{P}_N(\Omega^k)\} \cap \mathcal{H}_0^1(\Omega) \tag{4}$$

where $\mathbf{P}_N(\Omega^k)$ is the space of all polynomials of degree $\leq N$ in each spatial direction on element k. The spectral element method is thus characterized by the discretization pair $h = (K, N)$, where K is the number of subdomains (elements) and N is the order of the polynomial approximations. For reasons of efficiency (tensor products, [7]) the subdomains are taken to be quadrilaterals in $\mathbf{R}^2$ and hexahedra in $\mathbf{R}^3$. The spectral element discretization of (2) thus corresponds to numerical quadrature of the variational form (3) restricted to the subspace X_h: Find $u_h \in X_h(\Omega)$ such that:

$$\int_\Omega \nabla\phi_h \nabla u_h \, d\Omega = \int_\Omega \phi_h f_h \, d\Omega \qquad \forall \phi_h \in X_h(\Omega), \tag{5}$$

where f_h is the interpolant of f in the space X_h.

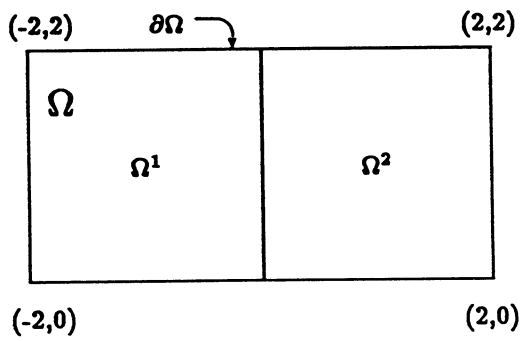


Figure 1: Computational domain Ω consisting of $K = 2$ subdomains.

While (5) is a statement of the type of solution which we seek, it does not indicate the form in which our solution will be represented, i.e., the choice of basis functions to be used for the polynomials in X_h. The Legendre spectral element method employs a tensor product form of Lagrangian interpolants based on a local, elemental, mapping of $\mathbf{x} \in \Omega^k \rightarrow \mathbf{r} = (r,s) \in [-1,1]^2$. We consider for illustration the case where the elemental decomposition consists of the union of squares with sides of length 2. Within each element, u_h has the form:

$$u_h(x,y)|_{\Omega^k} = \sum_{p=0}^{N}\sum_{q=0}^{N} u_{pq}^k h_p(r) h_q(s) \quad , \tag{6}$$

where $u_{pq}^k = u^k(r_p, s_q)$ are the unknown grid values of the approximate solution in element k. The interpolants, $h_i(\xi)$, satisfy:

$$\begin{aligned} h_i(\xi) &\in \mathbf{P}_N[-1,1] \\ h_i(\xi_j) &= \delta_{ij} \quad . \end{aligned} \tag{7}$$

The grid points, ξ_j, are chosen to be the Gauss-Lobatto-Legendre quadrature points [3,10].

With the explicit representation of functions given by (6), it is straightforward to evaluate the discrete variational form (5). Integration is performed using Gauss-Lobatto quadrature:

$$\int_\Omega g(x,y)d\Omega \rightarrow \sum_{k=1}^{K}\left\{\sum_{i=0}^{N}\sum_{j=0}^{N} g_{ij}^k \rho_{ij}\right\} \quad , \tag{8}$$

where ρ_{ij} is the quadrature weight associated with the point $\mathbf{r}_{ij}$. The derivatives at the quadrature points are computed as:

$$\begin{aligned} \left.\frac{\partial u}{\partial x}\right|_{\mathbf{x}_{ij}^k} &\rightarrow \left.\frac{\partial u^k}{\partial r}\right|_{\mathbf{r}_{ij}} = (D_{ip}\, u_{pj}^k), \\ D_{ij} &\equiv \left.\frac{dh_j(r)}{dr}\right|_{r=r_i} \quad , \end{aligned} \tag{9}$$

where, for notational convenience in this and the following equations, we use (.) to imply summation over the repeated index within the parentheses. The variational statement (5) therefore takes the form:

$$\sum_{k=1}^{K}\left\{\sum_{i=0}^{N}\sum_{j=0}^{N}\rho_{ij}\left[(D_{ip}\,\phi_{pj}^k)(D_{iq}\,u_{qj}^k) + (D_{jp}\,\phi_{ip}^k)(D_{jq}\,u_{iq}^k) - \phi_{ij}^k f_{ij}^k\right] - \right\} = 0 \quad , \tag{10}$$

where it remains to specify what are the admissible values of ϕ_{ij}^k.

Since Eq. (10) holds for arbitrary $\phi_h \in X_h$, the requisite discrete system of equations can be generated by setting $\phi_{ij}^k = \delta_{ii'}\delta_{jj'}\delta_{kk'}$, for all (i',j',k') corresponding to unique points in the domain interior. Note that the outermost summation in equation (10) implies that in the case where ϕ_{ij}^k has a physical counterpart in another element, k', (i.e. ϕ_{ij}^k lies on the interface between elements k and k'), the contributions to the integral (sum) from the adjacent elements must be added together. We refer to this operation as direct stiffness summation and denote it by $\sum_{\Omega^k}'$. The final system to be solved for u is therefore:

$$\sum_{\Omega^k}' \left\{ \sum_{p=0}^{N} D_{pi}(D_{pq}u_{qj}^k)\rho_{pj} + \sum_{p=0}^{N} D_{pj}(D_{pq}u_{iq}^k)\rho_{ip} \right\} = \sum_{\Omega^k}' f_{ij}^k \rho_{ij}. \qquad (11)$$

We solve equation (11) using iterative techniques, and therefore never explicitly construct the associated (elemental or global) matrices. Nonetheless, for notational convenience, we express (11) as:

$$\mathbf{A}\,\mathbf{u} = \mathbf{B}\,\mathbf{f} \ , \qquad (12)$$

where $\mathbf{A}$ is the global stiffness matrix and $\mathbf{B}$ is the (diagonal) mass matrix.

Iterative Solution Procedures

Due to memory, operation count, and parallelization considerations, iterative methods are used to solve the system (12). Such methods are dependent upon repeated evaluation of matrix-vector products of the form $\mathbf{r} = \mathbf{A}\mathbf{u}$, where $\mathbf{u}$ and $\mathbf{r}$ are intermediate vectors associated with successive iterations. For spectral element problems in higher space dimensions, d, the linear operators have large bandwidth and, if formed explicitly, are non-sparse with $O(KN^{2d})$ entries. The subsequent operation count and memory requirements can be significantly reduced if the matrix-vector product, $\mathbf{A}\mathbf{u}$, is evaluated element by element, using a factored form in which the discrete derivatives associated with the gradient operators in (11) are applied in a sequential fashion.

A typical term in the elemental matrix-vector product $A^k u^k$ for the case $d = 2$ is:

$$\sum_{p=0}^{N} \rho_{pj}\, D_{pi}(D_{pq}u_{qj}^k) \qquad \forall i,j \in \{0,...,N\}^2 \qquad (13)$$

Derivatives of the form (13) can be evaluated very efficiently via machine coded matrix-matrix product subroutines which require only $O(KN^d)$ memory references for the $O(KN^{d+1})$ operations. The derivative calculation is the single most computationally intensive step in the algorithm; as a result, a five-fold reduction in Navier-Stokes solution time is obtained when standard vectorized FORTRAN code is substituted with library matrix-matrix product routines on the Cray-X/MP. The residual evaluation is completed via direct stiffness summation wherein intermediate residual values at nodes shared by multiple elements are

summed and redistributed to the elemental data structures. The factored evaluation of Au requires only $O(KN^d)$ storage and $O(KN^{d+1})$ operations for general isoparametric discretizations.

To solve (12), we use Jacobi- (diagonal) preconditioned conjugate gradient iteration [7]. The preconditioned A system has condition $O(K_1^2N^2)$ implying an iteration count of $O(K_1N)$, where K_1 is the number of elements in a single spatial direction. The majority of the computational effort is associated with evaluation of Au, all other terms have an operation count of $O(KN^d)$ or less. In addition, two operations require communication of information between elements, namely, direct stiffness summation and inner-product evaluations of the form $\mathbf{r}^t\mathbf{r}$. While these steps require only $O(KN^{d-1})$ and $O(KN^d)$ operations, respectively, they represent the leading order *communication* terms in the parallel implementation discussed below.

Parallel Implementation

The spectral element discretizations, bases, and iterative solvers of the previous sections are constructed so as to admit a native, geometry-based, parallelism in which each spectral element (or group of spectral elements) is mapped to a separate processor/memory, with the individual processor/memory units being linked by a relatively sparse communications network. This conceptual architecture is naturally suited to the spectral element discretization in that it provides for tight, structured coupling within the dense elemental constructs, while simultaneously maintaining generality and concurrency at the level of the unstructured macro-element skeleton. The locally-structured/globally-unstructured spectral element parallel paradigm is closely related to the concept of domain-decomposition by substructured finite elements, and many of our results are generic to both computational models.

Our methods are implemented in an essentially machine-independent fashion. First, we construct a spectral element code in a standard high-level language in which each spectral element is treated as a "virtual parallel processor". In particular, each spectral element is treated as a separate entity, and all data structures and operations are defined and evaluated at the elemental level. The data and code are descended to M processors, each operating asynchronously. During the solution phase, the only procedures which require communication are, by construction, direct stiffness summation and vector reduction, which are relegated to special subroutines to effect data transfer. Processor synchronization is imposed at each iteration by these communication steps.

We comment that the element-processor partition is currently computed on a host processor during the problem start up, implying that, in that phase, a single processor has

access to a description of the entire problem. In an effort to eliminate the need to support two source codes, one on the node processors and one on the host, we are adapting an approach in which the nodes compute the element-processor mapping. This is decidedly more difficult when one envisions running on thousands of processors. If one is to fit the entire problem description on a singe processor, the amount of memory on at least one processor must grow in proportion to the number of processors in the system. Alternatively, an algorithm which works with segmented memory access must be developed. We are following the latter approach, but point out that this difficulty would be easily overcome in an environment which supports some shared memory or, at least, shared *read* access.

The most complex operation in our current parallel Navier-Stokes algorithm is the residual calculation, $\mathbf{r} = \mathbf{A}\mathbf{u}$, because of the communication required for direct stiffness summation. We illustrate its implementation in Fig. 2 in which a four element configuration in $\mathbf{R}^2$ is mapped to four processor/memory units. The elemental contributions to the

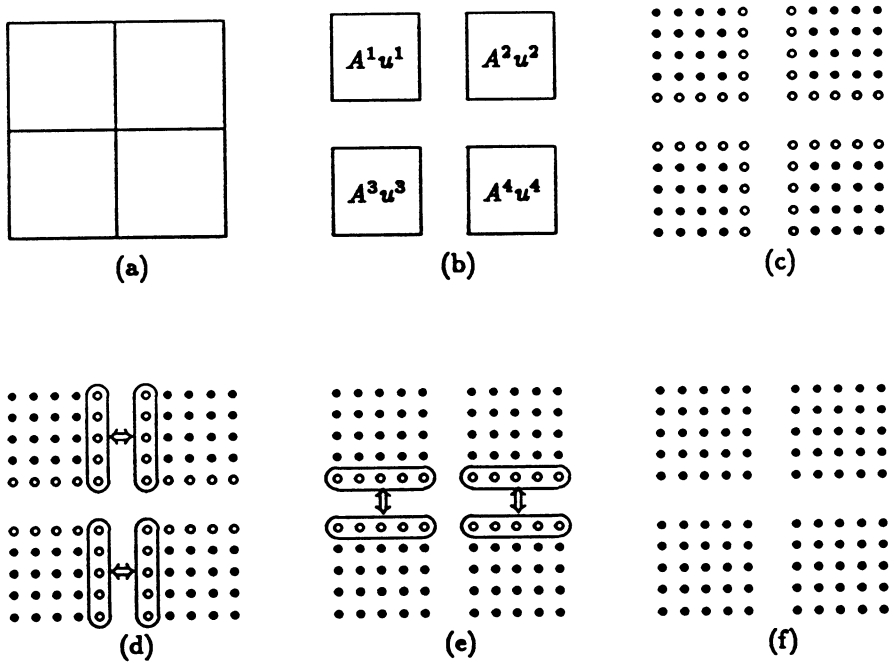


Figure 2: Computation of residual vector for regular geometry in $\mathbf{R}^2$: (a) four element mesh; (b) simultaneous (parallel) computation of incomplete residual, $\tilde{r}^k = A^k\tilde{u}^k$; (c) nodal content of $\tilde{r}^k$ is denoted by circles - solid circles indicate correct residuals, open circles indicate values requiring contributions from neighboring elements; (d) and (e) bi-directional exchange and sum sequence, ψ, which effects the completed residual at all nodes including the corner nodes; (f) completed residual, $\tilde{r}^* = \sum' A^k\tilde{u}^k$.

matrix-vector product $\mathbf{Au}$ are computed concurrently, and a d-directional exchange and sum sequence is used to complete the residual at the shared nodes. The exchanges denoted by the double arrows in Figs. 2d and e imply that a copy of the circled nodal content is sent to the adjacent processor which, upon receipt, is added to the resident values. Since each processor acts according to the same algorithm it follows that the shared edge values will be identical after direct stiffness summation. Note that for the frequently occuring regular geometry of Fig. 2a, the value at the common vertex is correctly updated by the ordered exchange sequence of Figs. 2d and e. However, for many topologies (e.g., one involving three or five elements sharing a single vertex) it is not possible to find a simple exchange sequence which will update shared vertices correctly. In order to retain the full generality of the spectral element method and to avoid the use of costly short data transfers, we have augmented the direct stiffness summation routine with a gather-scatter operation in which nodes not correctly addressed by the standard exchange sequence of Fig. 2 are mapped to copies of a global descriptor which are then summed via standard inter-processor vector reduction techniques.

The conjugate gradient algorithm requires evaluation of two inner-products during each iteration. In addition, the CFL condition will need to be evaluated at each time step. Such evaluations are typical of vector reduction operations in which information which is distributed over M independent memories must be condensed into a single result. Efficient evaluation of inner-products proceeds by computing local sums first, yielding distinct values on each processor. The M values are then summed via a $\log_2 M$ binary spanning tree which can be readily constructed for many network topologies, in particular for hypercube architectures [11]. When computing the global inner-product, multiple representations of vertices on the element interfaces are accounted for by multiplying each term by an inverse multiplicity corresponding to the number of elements which share that given vertex. The multiplicity can be computed once and stored during problem initiation.

Computational Complexity

We analyse the M-parallel performance of the spectral element conjugate gradient algorithm. Because of the heterogeneous nature of distributed memory parallel processing, it is necessary to account for the difference between local and non-local operations by introduction of two distinct time scales, δ - the (clock-cycle) time to execute a single floating point operation, and $\Delta(n)$ - the time per word required to transfer n words of data from one processor to another. The ratio Δ/δ is denoted $\sigma(n)$; $\sigma(n)$ is assumed to be a decreasing function of n, with $\sigma(1)$ appreciably greater than $\sigma(\infty)$ due to message startup overhead. Note that on vector systems, it is in fact an over simplification to give a single value for δ, as

the achieved execution rate will depend greatly upon the scalar-vector mix of the particular implementation. However, for purposes of parallel performance analysis, we take δ to be an average time per floating point operation.

Having already identified the leading order computation and communication terms, the expected solution time of our elliptic problem on M processors is readily expressed as:

$$\tau_{sol}(M) = \delta N_\epsilon^A \left\{ \frac{c_1 K N^{d+1}}{M} + c_2 \sigma(N^{d-1}) \tilde{K} N^{d-1} + c_3 \sigma(1) \log(M) \right\} , \qquad (14)$$

Where N_ϵ^A is the number of A iterations required to reach a specified tolerance, ϵ, and $\tilde{K} \leq 2dK/M$ is the maximum number of element edges per processor for which inter-processor communication is required in order to complete the direct stiffness summation. The c_1 term represents the leading order operation count per processor (generally, the matrix-matrix product operations), and is the only term present for the case of a serial processor implementation. The c_2 term accounts for the communication cost associated with the direct stiffness data exchange, while the c_3 term accounts for the communication for vector reduction operations.

The estimate (14) is based upon the assumptions that the computational load (c_1 term) is balanced and that direct stiffness summation is M-independent. The first assumption only holds in the case where K is a multiple of M. The second assumption is justified to the extent that disjoint element-element/processor-processor pairs can exchange data simultaneously, at a rate which is independent of the location of the processors in the network. Because direct stiffness summation is inherently local (though not intra-element), it will not be strongly dependent upon the system size. Note finally that the c_3 term associated with vector reductions grows as $O(\log_2 M)$ and is strongly M-dependent due to the global nature of the operation.

It is clear from (14) that for a fixed problem (K, N, d) there will be a minimal obtainable solution time for a number of processors, $M_{opt} = c_1 K N^{d+1} \ln 2 / c_3 \sigma(1)$, due to the presence of the $\log_2 M$ term. In addition, for a fixed number of processors, increasing the size of the problem (K, N) will result in solution times which approach the theoretical minimum, $\tau_{sol}(M) \rightarrow \tau_{sol}(1)/M$, due to the concurrent nature of the iterative algorithms.

It is of interest to analyse the case where K and M grow simultaneously in fixed proportion, in anticipation of the availability of larger numbers of processors and the consequent ability to solve larger problems. We consider the variation in parallel efficiency, $\eta \equiv \tau_{sol}(1)/M\tau_{sol}(M)$, which is given to leading order as:

$$\eta\Big|_{K/M} \simeq \eta_0 - \frac{c_3}{c_1} \frac{M}{K N^{d+1}} \sigma(1) \log(M) \qquad (15)$$

where η_0 is the efficiency obtained if one accounts only for direct stiffness communication costs. From (15) it is clear that for sufficiently large values of N, d, and K/M, the logarithmic degradation in efficiency will be small. Such performance has been observed empirically in our experience with the iPSC/2-VX. A typical three-dimensional flow calculation with $N = 10$ and $K/M = 2$ runs at roughly 75 percent efficiency and 10 to 11 MFLOPS on four processors. Similar performance is obtained for larger systems in which $K/M = 2$: 16 processors yield 44 MFLOPS, 32 yield 80 MFLOPS, and 64 yield 160 MFLOPS.

Measured Performance Analysis

We have implemented our algorithms on the Intel iPSC/1-VX and iPSC/2-VX hypercubes which are typical of the class of architectures for which the parallel spectral element method is well suited. The iPSC is a distributed memory, message passing, parallel processor consisting of $M = 2^D$ independent processor/memories, or nodes, arranged on a D-dimensional hypercube communication network. The iPSC/2-VX is an upgraded version of Intel's original iPSC hypercube which incorporates improved "worm-hole" message routing, allowing data to be transferred between non-nearest-neighbor processors with minimal degradation in data transfer rate, as well as a twenty-fold reduction in message transfer times, resulting in $\Delta(1) \simeq 300\mu$sec and $\Delta(\infty) \simeq 1.4\mu$sec. The nodes are based upon Intel 80386/80387 processor/coprocessors with a floating point execution rate of roughly 0.1 MFLOPS, coupled with attached vector processors which achieve 3-4 MFLOPS on standard vector operations and 10-12 MFLOPS on matrix-matrix products. Each node has 4 Mbytes of memory with an additional 1 Mbyte of fast vector memory. The programmer is responsible for vector calls and explicit transfer of data between the standard and vector memory.

We now analyze the spectral element/Intel iPSC-VX algorithm/ architecture coupling based on the complexity estimates of the previous section. We measure the time required for 250 A iterations for a model system consisting of a periodic chain of three-dimensional elements arranged in a $1\times1\times K$ array. The problem parameters are (N=10;K=M,2M), M=1,2,4,...,32. A "gray-code" mapping ensures that only nearest neighbor communication is required during direct stiffness summation, although this is, in fact, not critical.

In order to calculate speedup on the basis of this limited dataset we use the analysis of the previous sections to motivate a functional form for τ,

$$f_\tau(K,M) = aK/M + (b + c\log_2 M)\cdot(1-\delta_{1M}) \quad , \tag{16}$$

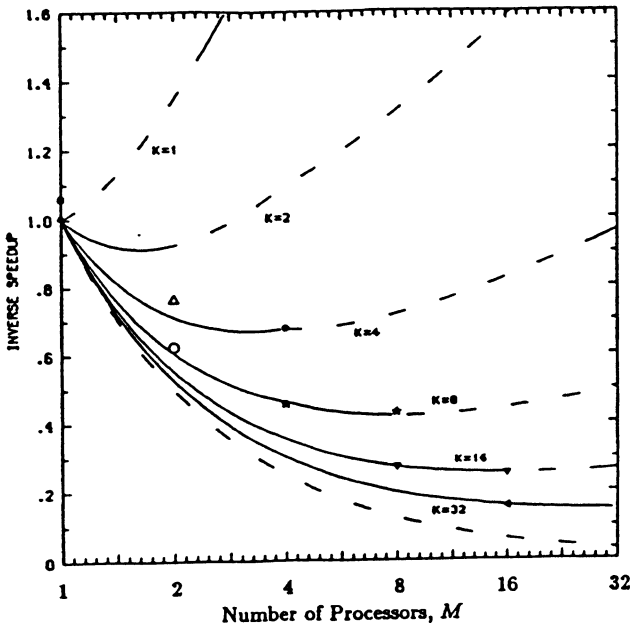
where a, b, and c are constants assumed independent of K and M. We then fit these constants (via least squares) to the total time data, finding for the iPSC/1-VX: a = 9.2 sec, b=3.1 sec, and c = 6.2 sec; and for the iPSC/2-VX: a = 8.1 sec, b = 0.46 sec, and c=0.32

sec. The results of the analysis are shown in Fig. 3 in which we plot $f_\tau(K,M)/f_\tau(K,1)$ versus K for the iPSC/1-VX and iPSC/2-VX. The dashed portion of the upper curves is the unreachable operating regime where $M > K$. The lower dashed curve is the asymptotic limit given by $1/M$. The communication effects are clearly evidenced by the presence of minimum solution times at $M_{opt} \simeq K$ in the iPSC/1 results of Fig. 3a, whereas $M_{opt} \gg K$ for the iPSC/2.

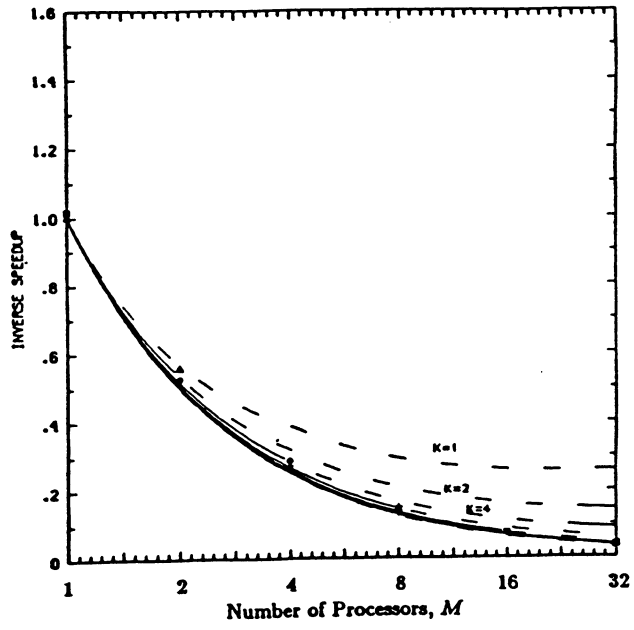
The functional form (16) can be used to predict the number of processors required to achieve a *sustained* 1 GFLOPS performance for the parallel spectral element method. We find that for the current iPSC/2-VX configuration and $K/M = 1$, 507 processors are required. If δ or Δ is halved, only 318 or 430 processors are required, respectively. For $K/M = 6$, the required number of processors is reduced by roughly 25 percent in all cases.

Given the care that is required in developing parallel algorithms, it is necessary that there be demonstrable advantages which justify departure from traditional programming strategies. To this end, we have performed extensive timing and operation count analysis of our parallel implementation and found that parallel processing indeed offers both speed and economy. Sustained execution rates of 40, 80, and 160 MFLOPS have been achieved on 16, 32, and 64 processors, respectively, for large three-dimensional Stokes and Navier-Stokes problems on the iPSC/2-VX. We present as an example timing results for the solution of an 80,000 degree-of-freedom steady Stokes problem. We plot in Fig. 4 resource efficiency, e, (MFLOPS/$) versus execution rate (MFLOPS) for the solution of this problem on several modern computers. The execution rate is derived by dividing the time required to solve the problem into the time required to solve the problem on a uVAX-II and multiplying by an application independent MFLOPS rating of 0.1 MFLOPS for the uVAX-II. The resource efficiency is derived by dividing the execution rate by the quoted manufacturer's list price. Performance on the 16-node Intel vector hypercube (44 MFLOPs) to that on a single node of a (dynamic RAM based) Cray-2 (66 MFLOPs) clearly shows that parallel machines can achieve serial supercomputer speeds at a fraction of the cost. We note that it is significant that the i/o performance of the Intel machines is much worse than that of the Cray-2; the i/o time for the Intel vector machines was roughly equal to the solution time, while for the Cray-2 it was virtually negligible. However, we have not included i/o times in this performance analysis because we are generally interested in time dependent calculations for which the i/o time is amortized over many time steps.

To illustrate the importance of the MFLOPS$-e$ framework, we have also included the data point for the K=32, M=16 problem on the nonvector iPSC/1 and iPSC/2; although the parallel efficiency on the nonvector machines is close to unity ($\eta > 0.99$), the nonvector machines are obviously uninteresting compared to their vector counterparts. This is due to the fact that the nonvector machines achieve high efficiency due to a decrease in σ brought



(a)



(b)

Figure 3: Measured and computed speed-up on the iPSC-VX for 250 A iterations for $(N = 10; K = M, 2M)$, and $M = 1, 2, 4, 8, 16, 32$. (a) iPSC/1-VX, (b) iPSC/2-VX.

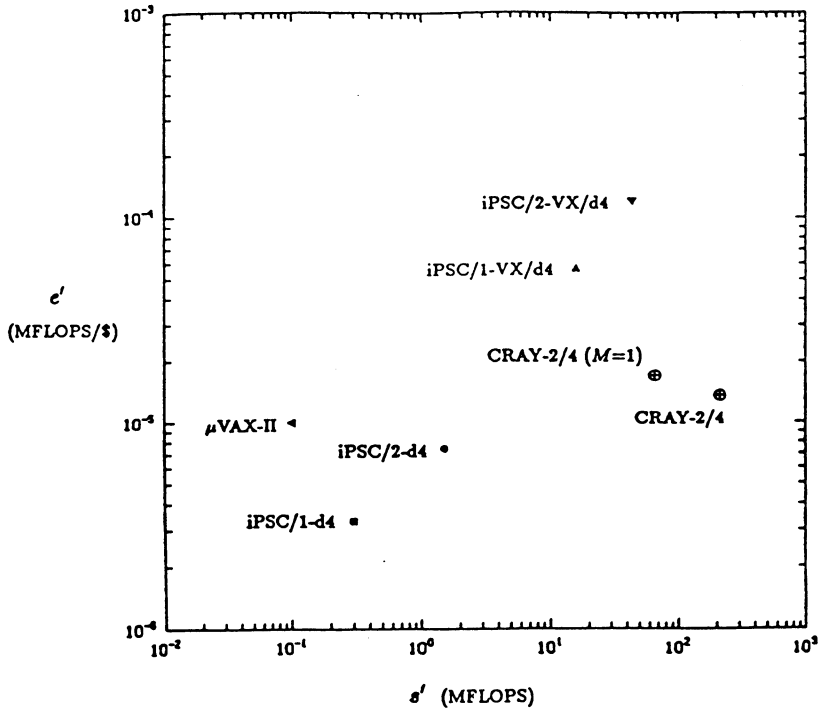


Figure 4: Measured computational resource efficiency, e, for spectral element solution to an 80,000 degree-of-freedom Stokes problem

about by increases in δ, not decreases in Δ. It is apparent from the nonvector exercise that vectorization internal to the nodes is important to performance; the nested parallel/vector hierarchy of the spectral element discretization is ideally suited for the task.

Results

The parallel spectral element method has been used in a number of two- and three-dimensional flow calculations. We present two recent results below.

Our first calculation is that of flow around a low aspect-ratio cylinder having height $H = 1/2$ and diameter $D = 1$. The computational domain extends to $z = 1.75$, and fluid is allowed to pass over the top of the cylinder. Baker [12] presents extensive flow visualization results for this configuration with $Re = UD/\nu = 4370$ and $D/\delta^* = 21.3$, where δ^* is the boundary layer displacement thickness at the position of the cylinder when the cylinder is not present. For the present calculations, $Re = 3000$ and $D/\delta^* = 21$ at the inlet plane. The boundary conditions are: $\vec{u} = 0$ at $z = 0$ and on the cylinder which is centered at $(x = 0, y = 0)$; $u = 1-(1-z/D)^{21}$ at $x = -2$; periodic boundary conditions at $y = \pm 2.25$; $\frac{\partial \vec{u}}{\partial z}$, $\vec{u} \cdot \hat{e}_z = 0$ at $z = 1.75$; $\frac{\partial \vec{u}}{\partial x} = 0$ at $x = 3.25$. The discretization consisted of 168 elements, with $N = 7$. The problem was run on a 16 node iPSC/2-VX and required roughly 70 hours of wall clock time. Visualization results are presented in Fig. 5. The chaotic flow seen above the cylinder is due in part to lack of resolution in this area and to the singular nature of the

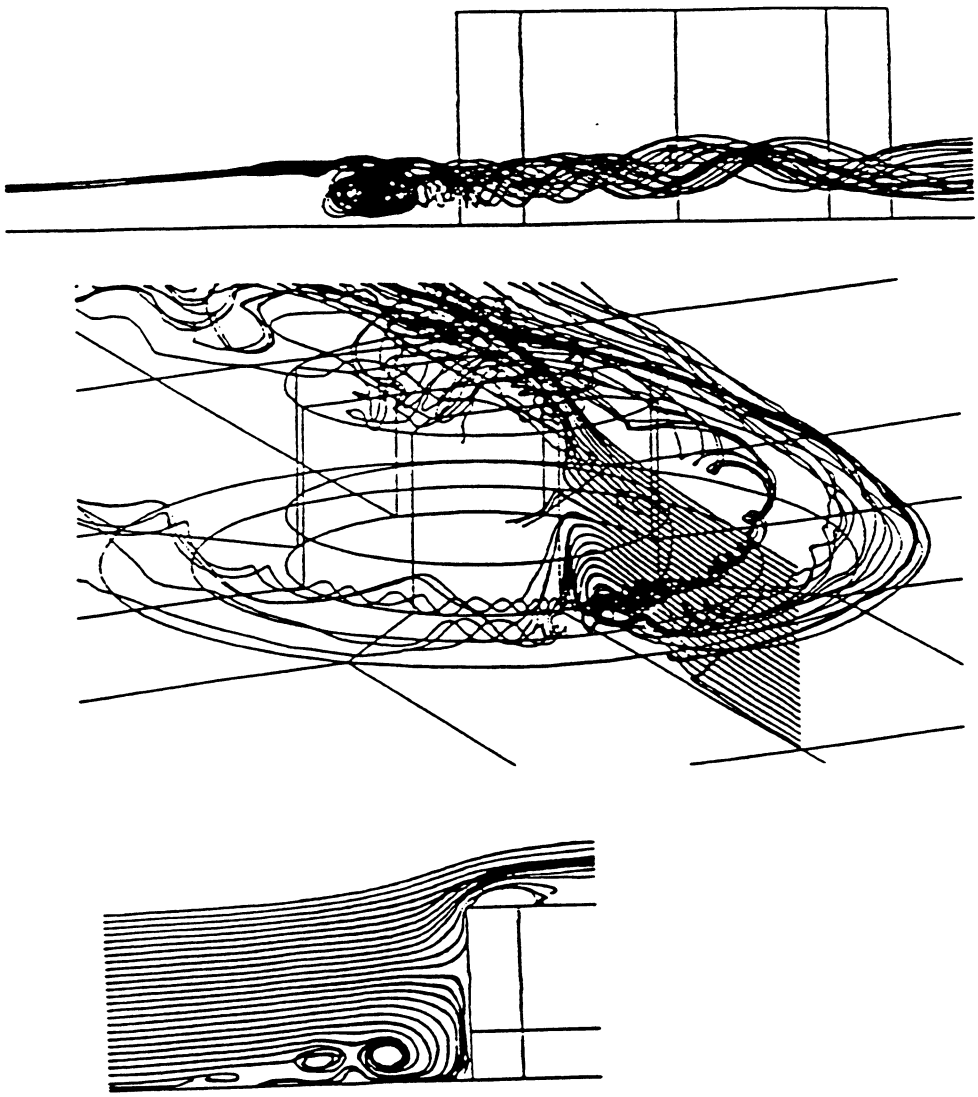


Figure 5: Horseshoe vortex at $Re_D = 3000$, $\delta^* = D/21$, and $H/D = 1/2$. Tight vortex core is seen at the cylinder base in the top figure. Multiple vortex formation is evident in the lower figures. Six vortex pattern at this Reynolds number is similar to the results of Sutton [12].

geometry which results in separation at the sharp edge. The results are in good qualitative agreement with the results of Baker and Sutton [12], in particular as regards the vortex pattern upstream of the cylinder.

In our second example, we consider flow in a container with a rotating lid. The problem is exactly that studied by Lugt and Abboud [13] for the case $Re = \Omega^2 R/\nu = 1492$, $H/R = 2$, with the exception that the full three-dimensional, time dependent Navier-Stokes equations are solved in anticipation of investigating three-dimensional modes. The spectral element decomposition ($K = 48, N = 10, d = 3$) is shown in Fig. 6a, with meridional plane streamlines in Fig. 6b, and axial velocity along the centerline in Fig. 6c. The results are in excellent agreement with Lugt and Abboud who use a 51×51 finite difference, axisymmetric discretization. The fact that the spectral element configuration requires fewer points in the meridional plane (20×40) is clearly advantageous in $\mathbf{R}^3$.

Acknowledgements

The author wishes to acknowledge the significant contributions to this work made by Anthony Patera of M.I.T., Einar Rønquist of Nektonics, Inc., and David Scott of Intel Scientific Computers. This work was supported while the author was at M.I.T. by the ONR and DARPA under contracts N00014-85-K-0208, N00014-87-K-0439, and N00014-88-K-0188, by NSF under Grants DMC-8704357 and ASC-8806925, and by Intel Scientific Computers. Research in the CRPC was provided by the NSF under Cooperative Agreement No. CCR-8809615.

References

[1] D. Gottlieb and S. Orszag, *Numerical Analysis of Spectral Methods* SIAM-CBMS, Philadelphia, 1977.

[2] P. Moin and J. Kim, "On the Numerical Solution of Time-Dependent Viscous Incompressible Fluid Flows Involving Solid Boundaries," *J. Comput. Phys.*, **35**,1980, p.381.

[3] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, 1987.

[4] P.F. Fischer and A.T. Patera, "Parallel Spectral Element Solution of the Stokes Problem," *J. Comput. Phys.*, to appear.

[5] A.T. Patera, "A spectral element method for fluid dynamics; Laminar flow in a channel expansion," *J. Comput. Phys.*, **54**, 1984, p.468.

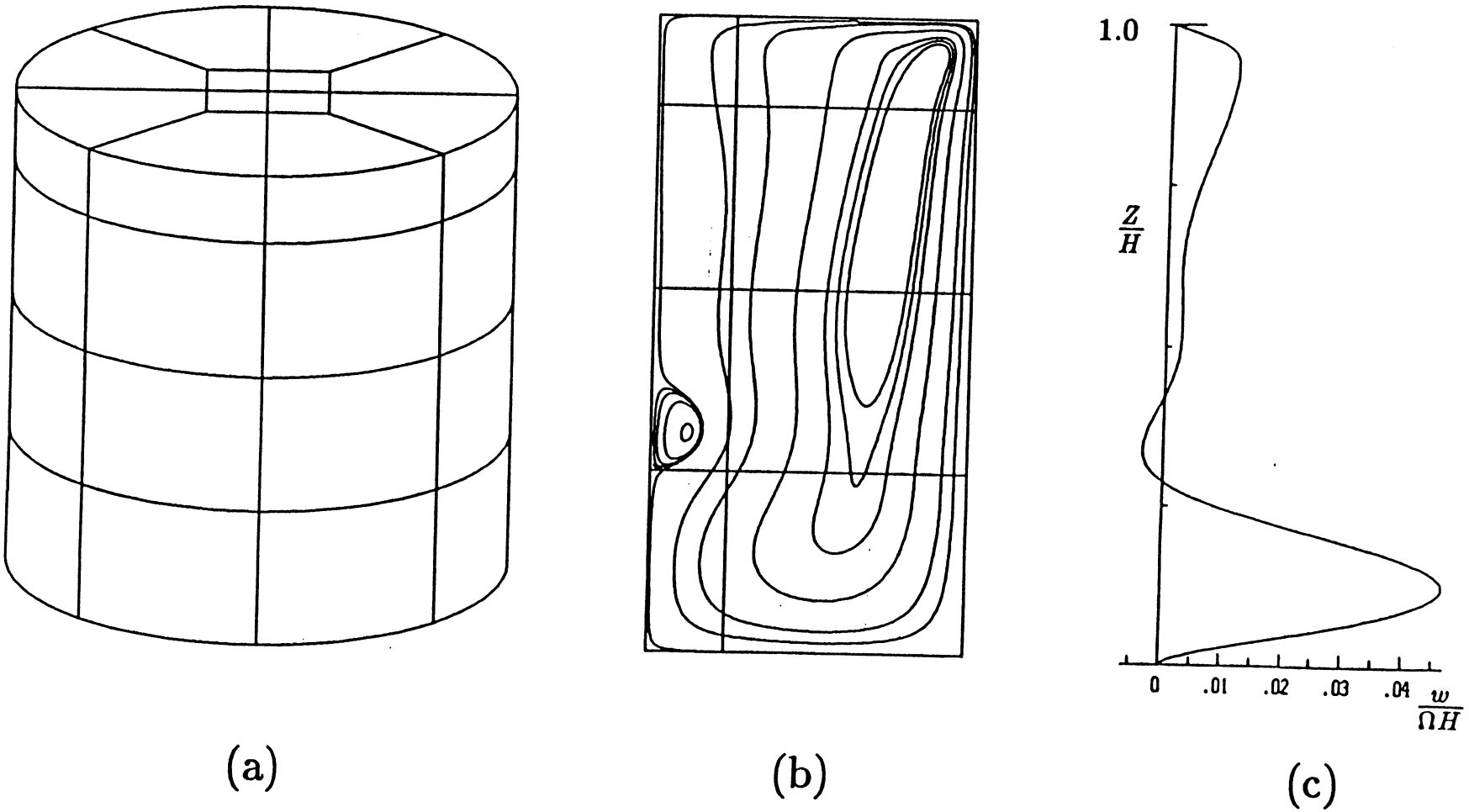


Figure 6: Flow in a container with a rotating lid for $Re = \Omega^2 R/\nu = 1492$: (a) computational domain ($K = 48, N = 10, d = 3$), (b) meridional streamlines, and (c) axial centerline velocity.

[6] Y. Maday and A.T. Patera, "Spectral element methods for the Navier-Stokes equations," in *State of the Art Surveys in Computational Mechanics* (Edited by A.K. Noor), ASME, New York, 1989.

[7] S.A. Orszag, "Spectral Methods for Problems in Complex Geometries," *J. Comput. Phys.*, **37**, 1980, p. 70.

[8] D.E. Keyes and W.D. Gropp, in *Proceedings of the Second International Conference on Domain Decomposition Methods for Partial Differential Equations, Los Angeles* SIAM, Philadelphia, 1988

[9] Christian H. Bischof and Jack J Dongarra, "A Linear Algebra Library for High-Performance Computers," in *Parallel Supercomputing: Methods, Algorithms and Applications* , G.F. Carey, ed., Wiley (1989), pp. 45-56.

[10] A.H. Stroud, and D. Secrest, *Gaussian Quadrature Formulas*, Prentice Hall, 1966.

[11] Y. Saad and M.H. Schultz, *Topological Properties of Hypercubes*, Research Report YALEU/DCS/RR-389, Yale University, New Haven, 1985.

[12] C.J. Baker, "The laminar horseshoe vortex," *J. Fluid Mech.*, **95**, 1979, p. 347.

[13] H.J. Lugt and M. Abboud, "Axisymmetric vortex breakdown with and without temperature effects in a container with a rotating lid," *J. Fluid Mech.*, **179**, 1987, pp. 179-200.