

## Parallel Spectral Element Solution of the Stokes Problem

PAUL F. FISCHER AND ANTHONY T. PATERA

*Department of Mechanical Engineering, Massachusetts Institute of Technology,  
Cambridge, Massachusetts 02139*

Received: June 15, 1988; revised: December 14, 1989

In this paper we present a high-efficiency medium-grained parallel spectral element method for numerical solution of the Stokes problem in general domains. The method is based upon: naturally concurrent Uzawa and Jacobi-preconditioned conjugate gradient iterative methods; geometry-based data-parallel distribution of work amongst processors; nearest-neighbor sparsity and high-order substructuring for minimum communication; general locally-structured/globally-unstructured parallel constructs; and efficient embedding of vector reduction operations for inner product and norm calculation. An analysis is given for the computational complexity of the algorithm on a "native" medium-grained parallel processor, and the potential communication superiority of high-order discretizations is described. Lastly, the method is implemented on the (fast) Intel vector hypercube, and the performance of this algorithm-architecture coupling is evaluated in a technical and economic framework that reflects the true advantages of parallel solution of partial differential equations. © 1991 Academic Press, Inc.

### INTRODUCTION

The solution of fluid dynamics problems by numerical simulation has advanced rapidly in recent years due to simultaneous improvements in algorithms and computers. However, despite these advances, computational fluid dynamics is still unable to address many problems of fundamental and practical importance due to the large number of degrees-of-freedom required to resolve relatively simple three-dimensional laminar flows, let alone transitional and turbulent flows. In essence, large-scale fluid mechanics calculations are still too costly in terms of human and computational resources to assume the role of "primary means of analysis."

A promising approach to reducing the costly nature of fluid dynamics calculations is to solve problems not on a single (expensive) computer, but rather to distribute the work amongst many less powerful (and less expensive) processors. The potential increase in efficiency due to the economies of parallel processing derive not only from decreases in direct costs, but also from improvements in productivity and creativity brought about by a more local and interactive computing environment. Unfortunately, the availability of parallel processors does not necessarily imply their efficient usage, and care must be taken in developing numerical algorithms that are appropriate for parallel implementation.

The purpose of the present paper is to describe a spectral element algorithm for the Stokes (and, by extension, Navier-Stokes) equations which exploits with high

parallel efficiency the distributed-memory parallel computers currently available. Our work builds extensively on past work on parallel partial differential equation solution in the choice of an iterative solver, as well as in the underlying strategy of load balancing, communication, and topological embeddings. In particular, our schemes are founded on the following well-developed precepts: use of iterative solvers that exploit sparsity and minimize non-concurrent operations, e.g., [1, 2]; geometry-based data-parallel distribution of work amongst processors, e.g., [3–6]; exploitation of nearest-neighbor sparsity and substructuring to minimize communication, e.g., [7, 8]; efficient embedding of vector reduction operations to allow for more general and implicit solution algorithms, e.g., [9, 10].

The methods presented in this paper represent an extension of these well-established ideas in the following ways. First, the spectral element discretizations [11] employed are high-order, leading not only to improved accuracy but also to a more efficient, work-intensive medium-grained parallelism. Second, the discretizations, solvers, and parallel constructs are built upon the general foundation of locally-structured/globally-unstructured representations, thus allowing for efficient implementation in arbitrary geometries. Third, the equations solved are the full equations describing viscous fluid flow, as opposed to (second-order elliptic) subsets of the Stokes problem; all potentially non-concurrent hazards are therefore addressed. Lastly, the methods are implemented on a *fast* vector parallel processor, and thus relative performance measures such as parallel efficiency can be supplemented with meaningful absolute measures such as cost-per-solution.

The outline of the paper is as follows. In Section 1 we briefly describe an economic caricature of computation with the aim of providing a rational framework in which to evaluate parallel performance. In Section 2 we introduce the spectral element discretization for elliptic operators, and indicate the extension of these discretizations to the Stokes and Navier–Stokes equations. In Section 3 we present a representative iterative solution procedure for the spectral element discretization and give serial computational complexity estimates. In Section 4 the “native” medium-grained/distributed-memory spectral element parallelism is presented and theoretical models are given for performance of the solution methods on various architectures. We also present a comparison of the relative medium-grained parallel potential of high-order and low-order variational discretizations. Lastly, in Section 5, the implementation of the methods is described, both in terms of general software constructs and for the particular case of the Intel iPSC vector hypercube. Computational results and performance measures are presented that demonstrate the advantage of spectral element algorithms on modern parallel computers.

## 1. ECONOMIC CARICATURE

Given the complexity of parallel computation as compared to its serial counterpart, it is imperative to verify that there is a sound economic basis for the notion

that parallelism will lead to improved computational "efficiency." To this end, we briefly review an economic caricature of the costs associated with numerical simulation. The particular physical problem of interest is fixed, and the maximum error that can be tolerated in the numerical solution,  $\varepsilon$ , is specified. We then choose an algorithm and architecture/machine with which to solve the problem: the former is characterized by  $W$ , the number of floating point operations required to attain the specified accuracy; the latter is characterized by the usual "fully-utilized" speed rating,  $s$  (in units of MFLOPS), and a purchase cost,  $C$  (in dollars, say). The wall-clock time to perform the calculation is then given by  $\tau = (W/10^6 \eta s)$ , and the direct computer costs are given by  $c = W/(10^6 \eta e t_{\text{dep}})$ . Here  $\eta$  is an algorithm-architecture efficiency parameter;  $e = s/C$  is a measure of the resource efficiency of a computer; and  $t_{\text{dep}}$  is a depreciation time ( $C/t_{\text{dep}}$  then represents a unit-time cost).

Although it is not appropriate in this context to introduce any particular computation cost function, it is clear that an unambiguous condition for reduction in cost (i.e., improvement in performance) is a simultaneous decrease in both the time to compute,  $\tau$ , and the direct computer cost of the solution,  $c$ . From the relationships between  $(\tau, c)$  and  $(W, s, e)$  we conclude that any algorithm-architecture coupling that corresponds to a decrease in  $W$ , an increase in  $s$ , and an increase in efficiency  $e$  constitutes a real increase in performance. There are two different avenues to improving performance. First, a better numerical algorithm can be devised, corresponding to a decrease in operation count,  $W$ , at fixed accuracy; this decreased operation count may be achieved either through improvements in discretization or through improvements in solution method. Second, a "better" computer can be found, in which both the speed,  $s$ , and resource-efficiency,  $e$ , are increased.

To illustrate more clearly the cost reduction due to computer performance, we plot in Fig. 1 the  $(s, e)$  operating points of several modern computers (this data is given in tabular form in Appendix A). It follows from the arguments given above that for a fixed algorithm and a fixed algorithm-architecture coupling,  $\eta$ , a computer  $A$  is better than a computer  $B$  if  $A$  is in the quadrant above and to the right of  $B$  in  $s$ - $e$  space. It is seen from Fig. 1 that supercomputers have made great strides in reducing  $\tau$ , however, they have had little impact as regards  $c$ ; this is consistent with the fact that supercomputers are typically used only where the potential profit is large and the analysis alternatives (e.g., experiment) are expensive.

In order to render the calculation of complex three-dimensional flows quotidian we will require significantly more resource-efficient computers. In fact, these machines now exist; within the past few years computers have emerged which are characterized by a resource-efficiency rating,  $\hat{e}$ , which is a full factor of 10 better than the previous norm,  $\bar{e}$ . This progress has been effected by basic hardware advances, that is, new technology, at the low MFLOPS limit of the  $e = \hat{e}$  curve, followed by parallel architecture advances which extend the performance envelope to the high-MFLOPS limit. In terms of the "quadrant of improvement" there now exist machines that represent clear improvements in performance over current mainframe and supercomputers alike.

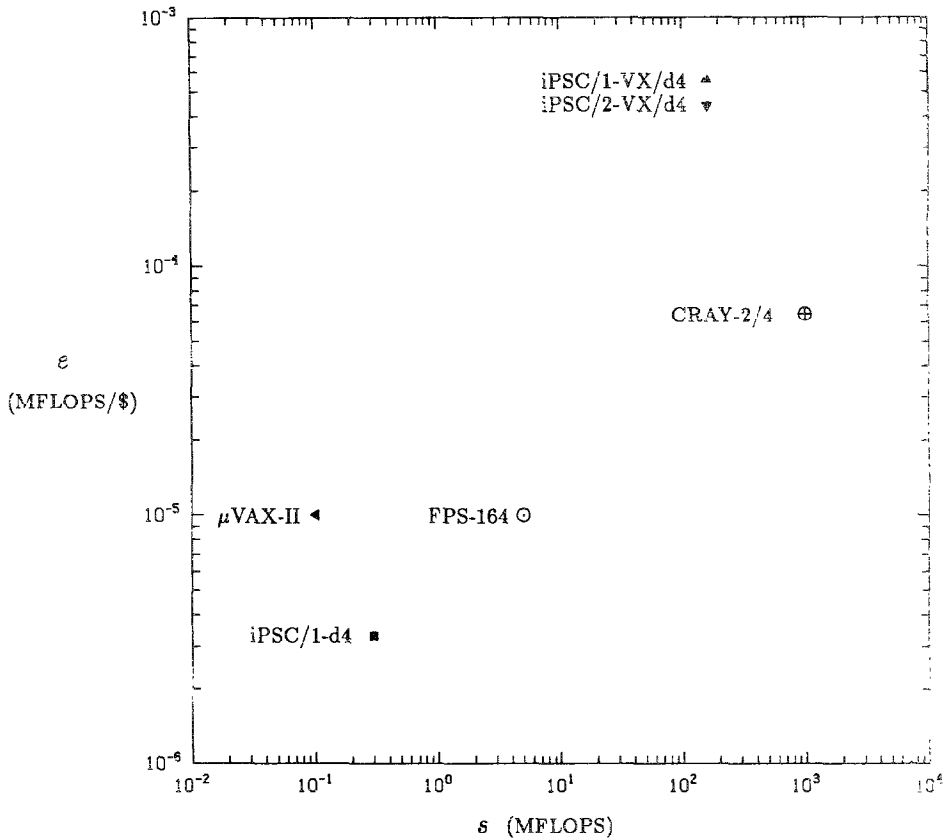


FIG. 1. Operating points (peak theoretical) of several modern computers in  $s-e$  space. Lines of constant  $s$  are lines of constant calculation time,  $\tau$ ; lines of constant  $e$  are lines of constant calculation cost,  $c$ . Cost data is given in Appendix A; in all cases  $M_{\max}$  processors are used.

The fact that computer manufacturers are able to increase the number of processors in a system,  $M_{\max}$ , with only a slightly faster than linear increase in cost is indicative of the fact that most of the new high-MFLOPS  $e = \hat{e}$  machines consist of processors which are largely independent, coupled by a rather sparse (albeit sophisticated) connection/routing network. This distributed-memory paradigm appears critical to maintaining cost effectiveness, in that it eliminates the need for many expensive memory access routes. The burden is thus placed on the algorithm to be sufficiently concurrent and "uncommunicative" to realize this ideal algorithm-independent performance; that is, the numerical algorithms must attain a sufficiently high algorithm-architecture efficiency  $\hat{\eta}$  so as not to erode the savings in  $\tau$  and  $c$  due to increase in  $s$  and  $e$ . Note that if we ignore other architectural issues such as vectorization,  $\hat{\eta}$  reduces to the usual definition of parallel efficiency.

$\eta = S_\tau/M$ , where  $M$  is the number of processors used in a calculation, and  $S_\tau$  is the parallel speedup, defined as  $S_\tau = \tau_{1-\text{proc}}/\tau_{M-\text{proc}}$ .

Having introduced our metrics of success in  $\tau$  and  $c$ , and hence  $s$  and  $e$ , we now turn to the algorithmic issues of determining  $W$  and  $\hat{\eta}$ .

## 2. THE SPECTRAL ELEMENT DISCRETIZATION

### 2.1. One-Dimensional Elliptic Problem

We present the spectral element method in some detail, as our parallel algorithms and constructs are closely coupled to the underlying discretization. The method is quite similar to  $h$ -type finite element substructure procedures [4, 12], as will be described in greater detail in Section 4.3. We begin by considering the simple one-dimensional ‘‘Poisson’’ equation,

$$-u_{xx} = f, \quad x \in A, \tag{1a}$$

with homogeneous Dirichlet boundary conditions

$$u(-1) = u(1) = 0, \tag{1b}$$

where  $A$  is the interval  $x \in ]-1, 1[$ . The basis for our numerical scheme is the variational form associated with (1): Find  $u \in H_0^1(A)$  such that

$$a(u, v) = (f, v), \quad \forall v \in H_0^1(A), \tag{2}$$

where

$$\forall \phi, \psi \in L^2(A), \quad (\phi, \psi) = \int_A \phi(x) \psi(x) dx, \tag{3a}$$

$$\forall \phi, \psi \in H_0^1(A), \quad a(\phi, \psi) = \int_A \phi_x(x) \psi_x(x) dx. \tag{3b}$$

The function spaces  $L^2(A)$  and  $H_0^1(A)$  are defined by  $L^2(A) = \{v \mid \int_A v^2 dx < \infty\}$ , and  $H_0^1(A) = \{v \mid v \in L^2(A), v_x \in L^2(A), v(-1) = v(1) = 0\}$ .

The spectral element method [11, 13] proceeds by specifying the discretization pair  $h = (K, N)$  and breaking up the interval  $A$  into  $K$  (assumed equal) subintervals,

$$\bar{A} = \bigcup_{k=1}^K \bar{A}^k,$$

where  $\bar{A}^k$  is defined by  $a^k \leq x \leq a^k + b$ . We then choose for the approximation of the

solution  $u$  a subspace  $X_h$  of  $H_0^1(A)$  consisting of all piecewise polynomials of degree  $\leq N$ ,

$$X_h = Y_h \cap H_0^1(A), \tag{4a}$$

where

$$Y_h = \{v | v|_{A^k} \in \mathbf{P}_N(A^k)\}. \tag{4b}$$

Here  $\mathbf{P}_N(A^k)$  is the space of functions which are polynomial of degree  $\leq N$  on the interval  $A^k$ .

The spectral element discretization corresponds to numerical quadrature of the variational form (2) restricted to the subspace  $X_h$ . Find  $u_h \in X_h$  such that

$$a_{h, GL}(u_h, v) = (f, v)_{h, GL} \quad \forall v \in X_h, \tag{5}$$

where  $(\cdot, \cdot)_{h, GL}$  and  $a_{h, GL}(\cdot, \cdot)$  refer to Gauss-Lobatto quadrature of the inner products defined in (3a) and (3b), respectively.

$$(\phi, \psi)_{h, GL} = \frac{b}{2} \sum_{k=1}^K \sum_{n=0}^N \rho_n \phi(\xi_n^k) \psi(\xi_n^k), \tag{6a}$$

$$a_{h, GL}(\phi, \psi) = \frac{b}{2} \sum_{k=1}^K \sum_{n=0}^N \rho_n \phi_x(\xi_n^k) \psi_x(\xi_n^k). \tag{6b}$$

Here the  $\xi_n^k = a^k + (\xi_n + 1) b/2$ ,  $0 \leq n \leq N$ ,  $1 \leq k \leq K$ , are the locations of the local nodes  $\{n; k\}$ , and the  $\xi_n, \rho^n$ ,  $0 \leq n \leq N$ , are the Gauss-Lobatto Legendre quadrature points and weights, respectively [14]. It can be shown that the spectral element solution  $u_h$  converges spectrally fast to the exact solution  $u$  as  $N \rightarrow \infty$  for  $K$  fixed, with exponential convergence being obtained for (locally) infinitely smooth data and solution [13].

The last step in the actual implementation of (5) is representation of  $u_h$  by a basis which reflects the sparsity and structure intrinsic to the piecewise-smooth space  $X_h$  in (4); the choice of basis will prove critical in subsequent efficient parallel implementation. We choose an interpolant basis to represent  $w_h \in X_h$ ,

$$w_h|_{A^k} = \sum_{p=0}^N w_p^k h_p(r), \quad x \in A^k \Rightarrow r \in I \tag{7a}$$

$$h_p \in \mathbf{P}_N(I), \quad h_p(\xi_q) = \delta_{pq} \quad \forall p, q \in \{0, \dots, N\}^2, \tag{7b}$$

where  $w_p^k = w_h(\xi_p^k)$  is the value of  $w_h$  at local node  $\{p; k\}$ ,  $\delta_{pq}$  is the Kronecker-delta symbol, and  $I$  is the interval  $]-1, 1[$ , with  $x \in A^k$  and  $r \in I$  related by  $x = a^k + (r + 1) b/2$ . To honor the  $H^1$  requirement and the essential boundary conditions (1b) we further require that

$$w_N^k = w_0^{k+1} \quad \forall k \in \{1, \dots, K-1\} \tag{8a}$$

and

$$w_0^1 = w_N^K = 0, \tag{8b}$$

respectively. Note that for a function  $w_h$  which is in  $Y_h$ , but not  $X_h$ , we use the same representation (7), but no longer require the conditions (8). For the spectral element mesh shown in Fig. 2a the nodal bases for  $X_h$  and  $Y_h$  are depicted diagrammatically in Figs. 2b and c, respectively; one-dimensional diagram conventions are defined in Table I.

We now insert (7)–(8) into (5)–(6) to arrive at the final discrete matrix statement,

$$\Sigma_p'^k \sum_{q=0}^N \hat{A}_{pq}^k u_p^k = \Sigma_p'^k \sum_{q=0}^N \hat{B}_{pq}^k f_q^k, \quad \forall k \in \{1, \dots, K\}, \quad \forall p \in \{0, \dots, N\}, \tag{9}$$

where  $f_q^k = f(\zeta_q^k)$  is the interpolant of the inhomogeneity, and

$$\begin{aligned} \hat{A}_{pq}^k &= \frac{2}{b} \sum_{n=0}^N \rho_n D_{np} D_{nq} & \forall p, q \in \{0, \dots, N\}^2 \\ \hat{B}_{pq}^k &= \frac{b}{2} \rho_p \delta_{pq} & \forall p, q \in \{0, \dots, N\}^2 \\ D_{pq} &= \frac{dh_q}{dr}(\zeta_p) & \forall p, q \in \{0, \dots, N\}^2. \end{aligned} \tag{10}$$

Here  $\Sigma_p'^k$  denotes “direct stiffness” summation, in which contributions from local nodes  $\{p; k\}$  which are physically coincident are summed (enforcing (8a)), and contributions from local nodes  $\{p; q\}$  which correspond to domain boundary points (here  $x = \pm 1$ ) are masked to zero (enforcing (8b)). Direct stiffness summation can be thought of as an operator  $\Sigma': Y_h \rightarrow X_h$ , as described diagrammatically in Fig. 3.

Although in practice parallel spectral element iterative procedures are based

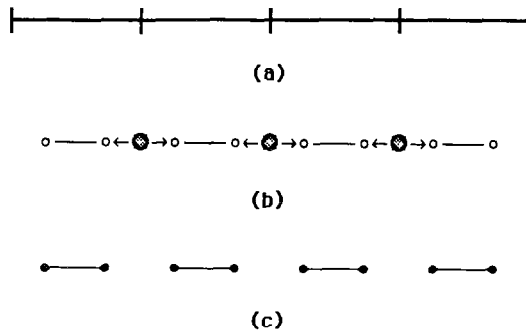


FIG. 2. Spectral element discretization for four elements in  $\mathbf{R}^1$  (a), with the corresponding nodal basis representation of the functional spaces  $X_h$  (b), and  $Y_h$  (c).

TABLE I  
Diagrams in  $\mathbf{R}^1$

$$w_h \Big|_{\Lambda^k} = \sum_{i=0}^N w_i^k h_i(r)$$

Data Type	Nodal Content	Symbol
Vertex	$w_0^k, w_N^k$	• ◦ (local) ● ○ (global)
Edge	$w_i^k, i \in \{1, \dots, N-1\}$	—————

Operations	
Assign Vertex	● → ◦
Summation of Vertices	• → ○ ← •

Note. Open or dashed objects denote destinations. Solid objects denote sources.

entirely on element-level calculations, it is of interest for notational purposes to define a global representation. In particular, we introduce a mapping from local node numbers  $\{p; k\}$  to global node numbers  $\{q\}_G$ , in which all local node numbers which reference the same nodal position map to a unique  $\{q\}_G$  (e.g.,  $\{N; k\} = \{0; k+1\} = \{\cdot\}_G$ ). In terms of this local-to-global mapping we then define a global-numbered condensed representation of a function  $w_h \in X_h$  as  $\underline{w}$ , where  $\underline{w}$  is the vector of values of  $w_h$  at all physically distinct nodes. The discrete equations (9) can be written in global form as

$$A\underline{w} = \underline{g}, \tag{11}$$

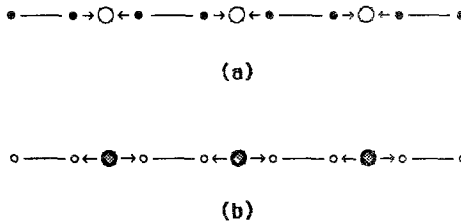


FIG. 3. Direct stiffness summation in  $\mathbf{R}^1$  represented as a mapping of data from  $Y_h$  to  $X_h$ : (a) summation; (b) redistribution.



where

$$\underline{A} = \sum_{pq}^{\prime k} \hat{A}_{pq} \quad \forall k \in \{1, \dots, K\}, \quad \forall p, q \in \{0, \dots, N\}^2, \quad (12a)$$

$$\underline{g} = \sum_p^{\prime k} \sum_{q=0}^N \hat{B}_{pq}^k f_q^k \quad \forall k \in \{1, \dots, K\}, \quad \forall p \in \{0, \dots, N\}. \quad (12b)$$

2.2. Multi-dimensional Elliptic Operators

We next consider the multi-dimensional elliptic equation,

$$-\nabla^2 u = f, \quad \mathbf{x} \in \Omega, \quad (13a)$$

with homogeneous Dirichlet boundary conditions

$$u = 0 \quad \text{on } \partial\Omega, \quad (13b)$$

in some bounded domain  $\Omega$  in  $\mathbf{R}^d$ . The variational form for (13) is given by: Find  $u(\mathbf{x}) \in H_0^1(\Omega)$  such that

$$a(u, v) = (f, v) \quad \forall v \in H_0^1(\Omega), \quad (14)$$

where

$$\forall \phi, \psi \in L^2(\Omega) \quad (\phi, \psi) = \int_{\Omega} \phi(\mathbf{x}) \psi(\mathbf{x}) \, d\mathbf{x}, \quad (15a)$$

$$\forall \phi, \psi \in H_0^1(\Omega) \quad a(\phi, \psi) = \int_{\Omega} \nabla \phi \cdot \nabla \psi \, d\mathbf{x}. \quad (15b)$$

The Sobolev spaces  $L^2(\Omega)$  and  $H_0^1(\Omega)$  are the usual multi-dimensional analogues of  $L^2(A)$  and  $H_0^1(A)$  defined previously.

For illustrative purposes we describe the simple case where the domain  $\Omega$  is a two-dimensional region representable by the union of  $K$  squares  $\Omega^k$  of edge length two,

$$\Omega = \bigcup_{k=1}^K \Omega^k.$$

(Three-dimensional curved-geometry examples are given in Section 5.) The solution  $u(\mathbf{x})$  is approximated by a subspace  $X_h$  of  $H_0^1(\Omega)$  consisting of all piecewise polynomials of degree  $\leq N$ ,

$$X_h = Y_h \cap H_0^1(\Omega), \quad (16a)$$

$$Y_h = \{v \mid v|_{\Omega^k} \in \mathbf{P}_N(\Omega^k)\}, \quad (16b)$$

where now  $\mathbf{P}_N(\Omega^k)$  is the space of all polynomials of degree  $\leq N$  in each spatial direction. The spectral element discretization is then: Find  $u_h \in X_h$  such that

$$a_{h, GL}(u_h, v) = (f, v)_{h, GL} \quad \forall v \in X_h, \quad (17)$$

where  $(\cdot, \cdot)_{h, GL}$  and  $a_{h, GL}(\cdot, \cdot)$  refer to tensor-product Gauss-Lobatto quadrature of the inner products defined in (15a) and (15b), respectively:

$$(\phi, \psi)_{h, GL} = \sum_{k=1}^K \sum_{m, n=0}^N \rho_m \rho_n \phi(\xi_m^k, \xi_n^k) \psi(\xi_m^k, \xi_n^k) \tag{18a}$$

$$a_{h, GL}(\phi, \psi) = \sum_{k=1}^K \sum_{m, n=0}^N \rho_m \rho_n \nabla \phi(\xi_m^k, \xi_n^k) \cdot \nabla \psi(\xi_m^k, \xi_n^k). \tag{18b}$$

The choice of basis in higher space dimensions is even more critical than in one space dimension, as the internal as well as element-boundary test functions directly affect the efficiency of the scheme. We choose a tensor-product interpolant basis to represent  $w_h \in X_h$ ,

$$w_h(x, y)|_{\Omega^k} = \sum_{p=0}^N \sum_{q=0}^N w_{pq}^k h_p(r) h_q(s), \quad \mathbf{x} \in \Omega^k \Rightarrow (r, s) \in I \times I, \tag{19}$$

where  $r$  and  $s$  are the local coordinates corresponding to translations of  $x$  and  $y$ , respectively;  $w_{pq}^k = w_h(\xi_p^k, \xi_q^k)$  is the value of  $w_h$  at local node  $\{p, q; k\}$ ; and the  $h_p(\xi)$  are the one-dimensional Gauss-Lobatto Lagrangian interpolants defined in (7b). Although for a function  $w_h$  in  $Y_h$  (19) is sufficient without further continuity conditions, for a function  $w_h$  in  $X_h$  the representation (19) is not complete until the two-dimensional  $H_0^1$  conditions analogous to (8) have been incorporated. In multi-dimensional problems the diagrammatic representations are much simpler than their indicial embodiments, and we thus forego the latter in favor of the former. For the spectral element mesh shown in Fig. 4a we present in Figs. 4b and c the diagrammatic representations of the bases  $X_h$  and  $Y_h$ , respectively, in terms of the two-dimensional diagrams defined in Table II.

The bases (19) are then inserted into the variational form (17) and inner products (18) to arrive at the final discrete matrix statement of the two-dimensional problem,

$$\begin{aligned} \Sigma'_{ij} \sum_{p=0}^N \sum_{q=0}^N (\hat{A}_{ip}^k \hat{B}_{jq}^k + \hat{B}_{ip}^k \hat{A}_{jq}^k) u_{pq}^k \\ = \Sigma'_{ij} \sum_{p=0}^N \sum_{q=0}^N \hat{B}_{ip}^k \hat{B}_{jq}^k f_{pq}^k \quad \forall k \in \{1, \dots, K\}, \quad \forall ij \in \{0, \dots, N\}^2, \end{aligned} \tag{20}$$

where the  $\hat{A}_{pq}^k, \hat{B}_{pq}^k$  are the one-dimensional operators defined in (10) (with  $b = 2$ ), and the two-dimensional direct stiffness operation  $\Sigma': Y_h \Rightarrow X_h$  is shown diagrammatically in Fig. 5. For notational purposes the system (20) can be reduced to a standard global form via the  $\{p, q; k\} \Rightarrow \{\cdot\}_G$  mapping,

$$\underline{A} \underline{u} = \underline{g}, \tag{21}$$

with  $\underline{A}, \underline{u}$ , and  $\underline{g}$  defined analogously to their one-dimensional counterparts in (11)-(12).

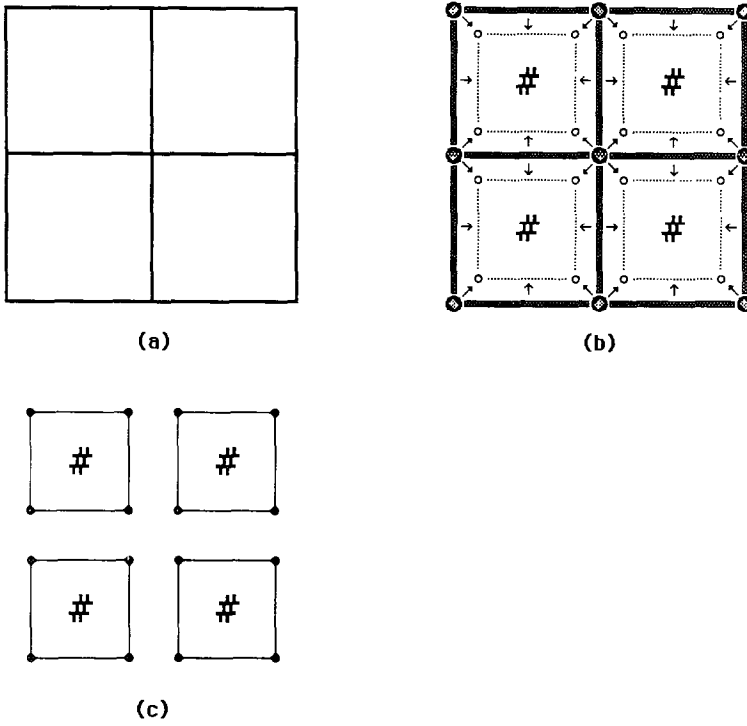


FIG. 4. Spectral element discretization for four elements in  $\mathbf{R}^2$  (a), with the corresponding nodal basis representation of the functional spaces  $X_h$  (b), and  $Y_h$  (c).

2.3. The Steady Stokes Problem

Armed with the detailed descriptions of spectral element projection operators, spaces, and bases described in the last section, we now turn to the discretization of the steady (incompressible) Stokes equations,

$$-\nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } \Omega \tag{22a}$$

$$-\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \tag{22b}$$

$$\mathbf{u} = \mathbf{0} \quad \text{on } \partial\Omega, \tag{22c}$$







where  $\mathbf{u}(\mathbf{x})$  is the velocity,  $p$  is the pressure, and  $\mathbf{f}$  is the force. The equations are appropriately non-dimensionalized such that the viscosity and density of the fluid are unity. The variational form of (22) is well known [15, 16]: Find  $\mathbf{u} \in (H_0^1(\Omega))^d$  and  $p \in L_0^2(\Omega)$  such that

$$a(\mathbf{u}, \mathbf{v}) - b(p, \mathbf{v}) = (\mathbf{f}, \mathbf{v}) \quad \forall \mathbf{v} \in (H_0^1(\Omega))^d \tag{23a}$$






$$-b(q, \mathbf{u}) = 0 \quad \forall q \in L_0^2(\Omega). \tag{23b}$$

TABLE II  
Diagrams in  $\mathbf{R}^2$

$$w_h|_{\Omega^k} = \sum_{i=0}^N \sum_{j=0}^N w_{ij}^k h_i(r) h_j(s)$$

Data Type	Nodal Content	Symbol
Vertex	$w_{00}^k, w_{N0}^k, w_{0N}^k, w_{NN}^k$	 (local)  (global)
]Edge[	$w_{i,j}^k, i = 0, N, j \in \{1, \dots, N-1\}$ $j = 0, N, i \in \{1, \dots, N-1\}$	 (local)  (global)
[Edge]	$w_{i,j}^k, i = 0, N, j \in \{0, \dots, N\}$ $j = 0, N, i \in \{0, \dots, N\}$	 (local) 
Area	$w_{ij}^k, i, j \in \{1, \dots, N-1\}^2$	#

Operations	
Assign Vertex	
Assign ]Edge[	
Summation of Vertices	
Summation of ]Edges[	
Sum & Redistribute [Edges]	

Note. Open or dashed objects denote destinations. Solid objects denote sources, except in the sum and redistribute operation where both objects act as sources and destinations.

All inner products and spaces have been defined previously save  $b(\cdot, \cdot)$ ,

$$\forall \phi \in L^2(\Omega), \mathbf{w} \in (H_0^1(\Omega))^d, \quad b(\phi, \mathbf{w}) = \int_{\Omega} \phi(\mathbf{x}) \operatorname{div} \mathbf{w} \, d\mathbf{x}. \quad (24)$$

(The space  $L_0^2$  is the space of all functions in  $L^2$  with zero mean.)

The spectral element discretization is then: Find  $\mathbf{u}_h \in (X_h)^d$  and  $p \in M_h$  such that

$$a_{h, GL}(\mathbf{u}_h, \mathbf{v}) - b_{h, G}(p_h, \mathbf{v}) = (\mathbf{f}, \mathbf{v})_{h, GL} \quad \forall \mathbf{v} \in (X_h)^d \quad (25a)$$

$$-b_{h, G}(q, \mathbf{u}_h) = 0 \quad \forall q \in M_h, \quad (25b)$$

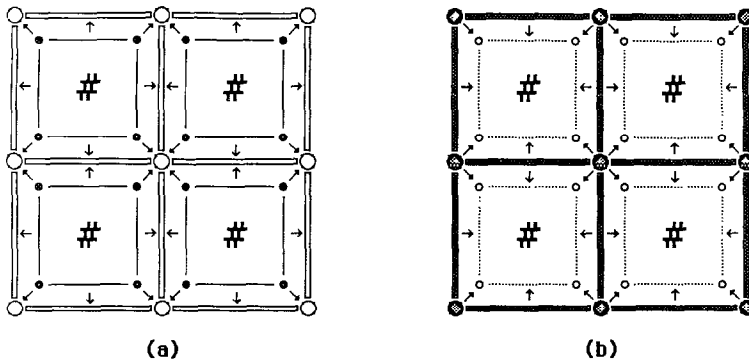


FIG. 5. Direct stiffness summation in  $\mathbf{R}^2$  represented as a mapping of data from  $Y_h$  to  $X_h$ : (a) summation, (b) redistribution.

where  $X_h$ ,  $(\cdot, \cdot)_{h, GL}$ , and  $a_{h, GL}(\cdot, \cdot)$  are defined in (16), (18a), and (18b), respectively. The pressure space is given by

$$M_h = \{q \mid q|_{\Omega^k} \in \mathbf{P}_{N-2}(\Omega^k)\} \cap L^2_0, \tag{26}$$

where the choice of  $\mathbf{P}_{N-2}$  is dictated by the inf-sup condition [13, 17]. The discrete product  $b_{h, G}(\cdot, \cdot)$  is related to the continuous form  $b(\cdot, \cdot)$  in (24) in the same fashion as  $a_{h, GL}(\cdot, \cdot)$  in (6b) is derived from  $a(\cdot, \cdot)$  in (3b), except that for the  $b$  inner product the numerical quadrature is a tensor-product Gauss (not Gauss-Lobatto) formula based on  $N-1$  points [18].

We now express the velocity in the tensor-product basis (19) and the pressure in an analogous tensor-product representation based on one-dimensional Lagrangian interpolants through the Gauss points. Inserting these bases into the variational forms, we arrive at the following global representation of the discrete Stokes system,

$$\underline{A}\mathbf{u} - \underline{D}^T p = \mathbf{g} \tag{27a}$$

$$-\underline{D} \cdot \mathbf{u} = 0, \tag{27b}$$

where  $\underline{A}$  and  $\mathbf{g}$  are the Laplacian operator and inhomogeneity appearing in (21). The matrix  $\underline{D}$  is a gradient operator associated with the  $b$  inner product, which in elemental representation has a tensor-product form similar to that of  $\underline{A}$ . More details of the Stokes discretization (27) can be found in [13, 17], in particular, as regards stability and convergence. It should be noted that (27) is readily extended to the unsteady Navier-Stokes equations if the nonlinear convective terms are treated explicitly or by characteristic methods [19, 20]; the final set of equations closely resembles (27), with the nonlinear terms appearing as an augmented force  $\tilde{\mathbf{g}}$  [13, 21, 22, 23].

### 3. ITERATIVE SOLUTION PROCEDURES

The natural choice of solution algorithm in a parallel environment is an iterative procedure, given that such techniques can be both highly local and concurrent. In this section we describe essential features of iterative spectral element solvers.

#### 3.1. Evaluation of Spectral Element Operators

At the heart of any iterative solver is the evaluation of matrix-vector products such as those that appear in (20). We review here briefly how these products can be efficiently calculated using sum-factorization methods [24]. Considering a representative term in (20), the double sum can be factored as

$$\Sigma_{ij}^{rk} \left\{ \left( \sum_{p=0}^N \hat{A}_{ip}^k \left( \sum_{q=0}^N \hat{B}_{jq}^k u_{pq}^k \right) \right) \right\} \quad \forall k \in \{1, \dots, K\}, \forall ij \in \{0, \dots, N\}^2. \quad (28)$$

It is clear that, for a discretization in  $\mathbf{R}^d$ , each term in parentheses in (28) can be evaluated in  $O(KN^{d+1})$  operations, and that the final direct stiffness summation described by Fig. 5 will require  $O(KN^{d-1})$  operations. It thus follows that the number of clock cycles required to evaluate the left side of (20) on a *single* processor is

$$Z_1^e = c_1 KN^{d+1} + c_2' KN^d + c_3 KN^{d-1}. \quad (29)$$

Here, and in what follows, constants  $c_1, c_2', c_3, \dots$  depend only on spatial dimension, and not on  $K, N$ , or the number of processors in the system. The  $O(KN^d)$  contribution to  $Z_1^e$  is only present in the case of complex geometry or non-separable coefficients. It should also be noted that only  $O(KN^d)$  storage is required to evaluate  $Au$ .

The proper choice of spectral element basis is directly reflected in the “good” computational complexity estimate  $Z_1^e$ . First, the sum-factorization (28) and the operation count (29) applies to general-geometry isoparametric spectral element discretizations of non-separable equations [13], due to the tensor product element-geometry, tensor product spaces (16), tensor product quadratures (18), and tensor product bases (19) described in Section 2. Second, the direct stiffness summation contribution to  $Z_1^e$  is only  $O(KN^{d-1})$ , rather than  $O(KN^{d+1})$ , due to our choice of basis (Fig. 4) in which the number of test functions which are nonzero on the elemental boundary is minimal. Although the fact that the direct stiffness summation work is small does not appear particularly important in the single-processor estimate (29), in the parallel case the direct stiffness contribution will be the leading-order *communication* term.

#### 3.2. Conjugate Gradient Iteration

We next consider simple Jacobi (diagonal)-preconditioned conjugate-gradient iterative solution [25] of the multi-dimensional elliptic equation (21). The conjugate gradient algorithm is given by

$$\begin{aligned}
 \underline{u}^0; \underline{r}^0 &= \underline{g} - \underline{A} \underline{u}^0; & \underline{q}^0 &= \underline{P}^{-1} \underline{r}^0; & \underline{s}^0 &= \underline{q}^0 & (30) \\
 \underline{a}^m &= (\underline{r}^m, \underline{s}^m) / (\underline{q}^m, \underline{A} \underline{q}^m); & \underline{u}^{m+1} &= \underline{u}^m + \underline{a}^m \underline{q}^m \\
 \underline{r}^{m+1} &= \underline{r}^m - \underline{a}^m \underline{A} \underline{q}^m; & \underline{s}^{m+1} &= \underline{P}^{-1} \underline{r}^{m+1} \\
 \underline{b}^m &= (\underline{r}^{m+1}, \underline{s}^{m+1}) / (\underline{r}^m, \underline{s}^m); & \underline{q}^{m+1} &= \underline{s}^{m+1} + \underline{b}^m \underline{q}^m,
 \end{aligned}$$

where  $\underline{P} = \text{diag}(\underline{A})$  is the diagonal preconditioner, superscripts denote iteration number, and  $(\cdot, \cdot)$  refers to the usual inner product. Note that  $\underline{P}$  can be formed explicitly without constructing the entire  $\underline{A}$  operator.

From (30) we see that, per iteration, the conjugate gradient scheme requires: one matrix-vector evaluation— $Z_1^c$  cycles; several local collocation operations— $O(KN^d)$  cycles; and two inner products— $O(KN^d)$  cycles. If we denote by  $N_\epsilon^A$  the number of iterations required to bring the error in the solution down to  $O(\epsilon)$  in some appropriate norm [22, 26], the number of clock cycles for conjugate gradient solution of (21) is

$$Z_1^A = N_\epsilon^A (c_1 KN^{d+1} + c_2 KN^d + c_3 KN^{d-1}), \tag{31}$$

where the  $c_1$  term represents all matrix-vector products, the  $c_2$  term represents all collocation operations and vector reduction (inner product/norm) calculations, and the  $c_3$  term represents direct stiffness summation.

Although in the evaluation of parallel performance in Section 3 the number of iterations,  $N_\epsilon^A$ , will scale out, it is nevertheless appropriate to comment on the number of iterations required to achieve convergence. Denoting  $\kappa_M$  the condition number of any symmetric matrix  $M$ , it can be shown that  $\kappa_A \sim K_1^2 N^3$ , and  $\kappa_{(P^{-1/2}AP^{-1/2})} \sim O(K_1^2 N^2)$  [27] (note this convergence rate is basis-dependent). Here  $K_1$  is the number of spectral elements in one spatial direction. It thus follows that for conjugate gradient iteration  $N_\epsilon^A \sim K_1 N$ . The convergence rate can be improved to near optimality by the use of spectral element multigrid algorithms [27, 28].

### 3.3. Uzawa Steady-Stokes Algorithm

The Uzawa iterative procedure [16, 29, 30] for the Stokes problem (27) is based on decomposition of the saddle problem into two symmetric-definite forms,

$$\underline{A} \underline{u} - \underline{D}^T \underline{p} = \underline{g} \tag{32a}$$

$$\underline{S} \underline{p} = -\underline{D} \cdot \underline{A}^{-1} \underline{g}, \tag{32b}$$

where

$$\underline{S} \equiv \underline{D} \cdot \underline{A}^{-1} \underline{D}^T. \tag{33}$$

The system (33) is solved by nested conjugate gradient iteration,

$$\begin{aligned}
 \underline{p}^0; \underline{r}^0 &= -\underline{\mathbf{D}} \cdot \underline{\mathbf{A}}^{-1} \underline{\mathbf{g}} - \underline{\mathbf{S}} \underline{p}^0; & \underline{q}^0 &= \underline{\tilde{\mathbf{B}}}^{-1} \underline{r}^0; \underline{s}^0 = \underline{q}^0 \\
 \alpha^m &= (\underline{r}^m, \underline{s}^m) / (\underline{q}^m, \underline{\mathbf{S}} \underline{q}^m); & \underline{p}^{m+1} &= \underline{p}^m + \alpha^m \underline{q}^m \\
 \underline{r}^{m+1} &= \underline{r}^m - \alpha^m \underline{\mathbf{S}} \underline{q}^m; & \underline{s}^{m+1} &= \underline{\tilde{\mathbf{B}}}^{-1} \underline{r}^{m+1} \\
 b^m &= (\underline{r}^{m+1}, \underline{s}^{m+1}) / (\underline{r}^m, \underline{s}^m); & \underline{q}^{m+1} &= \underline{r}^{m+1} + b^m \underline{q}^m,
 \end{aligned} \tag{34}$$

in which the  $\underline{\mathbf{S}} \underline{q}$  product is evaluated by

$$\underline{\mathbf{y}} = \underline{\mathbf{D}}^T \underline{q} \tag{35a}$$

$$\underline{\mathbf{A}} \underline{\mathbf{z}} = \underline{\mathbf{y}} \tag{35b}$$

$$\underline{\mathbf{S}} \underline{q} = \underline{\mathbf{D}} \cdot \underline{\mathbf{z}}, \tag{35c}$$

and the system (35b) is solved by an *inner* conjugate gradient iteration (30). The matrix  $\underline{\tilde{\mathbf{B}}}$  appearing as preconditioner in (34) is the diagonal pressure mass matrix. Once the pressure is known, (32a) is solved following (30).

To arrive at a final work estimate for the Stokes problem we note that all the operations present in (34)–(35) are already present in the conjugate gradient iteration (30); as  $N_\epsilon^A$  will always be “large” compared to unity, it follows that the number of clock cycles for a Stokes solve on a single processor scales as

$$Z_1^S = N_\epsilon^S N_\epsilon^A (c_1 K N^{d+1} + c_2 K N^d + c_3 K N^{d-1}), \tag{36}$$

to leading order in  $1/N_\epsilon^A$ . As far as the conditioning of the outer iteration is concerned, it can be shown that the matrix  $\underline{\mathbf{S}}$  is spectrally quite close to the identity operator, that is,  $\kappa_S \sim O(1)$  independent of  $K$  and  $N$  [23, 26]. This good conditioning can be intuited by remarking that  $\underline{\mathbf{S}}$  is essentially the product of two gradient operators and an inverse Laplacian; the latter balances the former, resulting in a bounded operator. It follows that the outer conjugate gradient iteration (34) will converge in order unity iterations,  $N_\epsilon^S \sim O(1)$ . We conclude that the Uzawa algorithm effectively reduces the Stokes problem (27) to the Poisson problem (21) as regards computational complexity and parallelism; the same is true for other Stokes discretization/solvers, such as the operator-splitting techniques [31].

#### 4. PARALLEL SPECTRAL ELEMENT SOLUTION TECHNIQUES

Our construction of parallel algorithms is broken into three coupled steps. In the first step, described in the previous section, discretizations and solvers are developed that are designed to exploit concurrency and minimize communication. In the second step, we consider the effectiveness of the resulting algorithms on a “native” parallel processor, that is, a conceptual machine which directly mimics the



natural parallelism in the underlying numerical approach. In the third step, the adaptation of the method to existing (and typically non-specialized) architectures is considered, with various non-idealities characterized by efficiencies or multipliers with respect to the "native" architecture.

The purpose of breaking apart the second and third steps is to provide, through the second step, a fairly generic analysis of the method that is not tied to the specifics of any particular machine. Furthermore, breaking apart the second and third steps allows for architectural comparisons, rather than case-by-case algorithm-architecture analyses. It should be clear that all three steps of the development process are closely coupled; a numerical method which results in a native parallelism which has no efficient realization does not constitute a viable approach.

#### 4.1. Native Parallelism

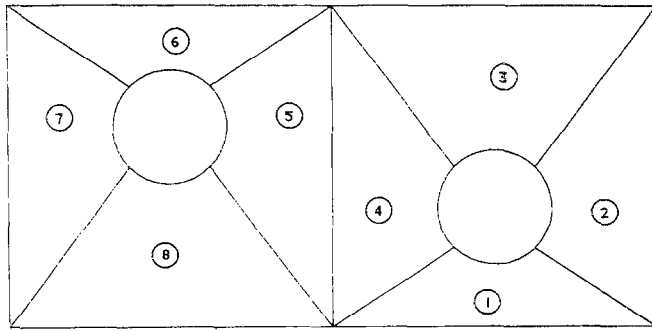
The spectral element discretizations, bases, and iterative solvers of the previous sections are constructed so as to admit a native, geometry-based data parallelism [32], in which each spectral element (or group of spectral elements) is mapped to a separate *processor/memory* unit, with the individual 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 [4, 5, 12], and many of our results are germane to both computational models. This latter point will be discussed in greater detail in Section 4.3.

We shall begin by considering the performance of the method on the native medium-grained distributed-memory parallel processor shown in Fig. 6, in which  $K$  spectral elements are partitioned amongst  $M \leq K$  independent processor/memory units,  $P_1, \dots, P_M$ . (Our terminology will be two-dimensional; however, the methods readily extend to three space dimensions, as will be demonstrated by examples in Section 5.) In essence, each processor contains a "super-substructure" of several spectral elements. We denote the set of all elements  $E = \{1, \dots, K\}$  and the set of elements associated with processor  $P_q$  as  $E_q = \{\dots\}$ , with  $E = \bigcup_q E_q$ , and  $E_p \cap E_q = \emptyset$  for  $p \neq q$ . We assume load balance in the sense that all processors have an equal number of elements.

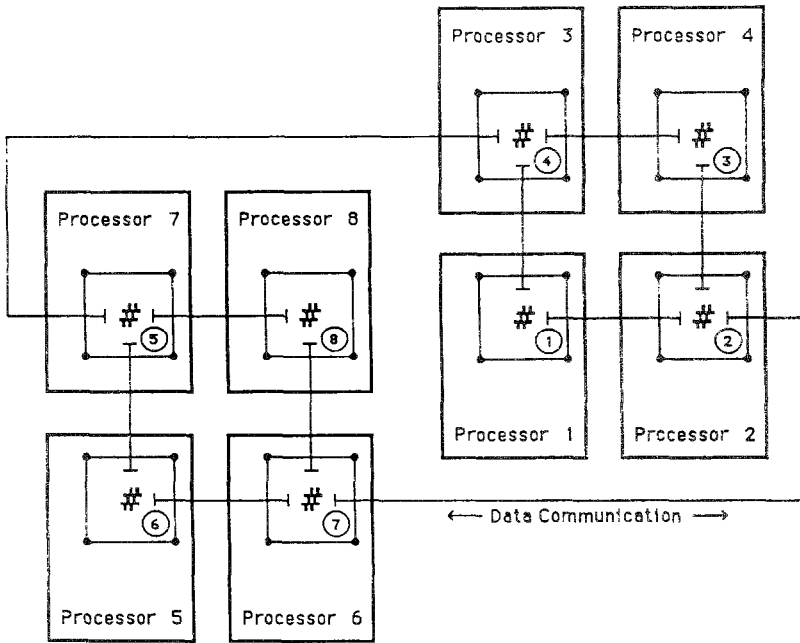
The communications network of the native parallel processor is assumed to satisfy two constraints:

$$\begin{aligned} & \text{a distinct, direct link exists between two processors } P_p \text{ and } P_q, \\ & p \neq q, \text{ for each distinct pair of elements } (m, n), m \in E_p, n \in E_q \text{ that} \\ & \text{share an edge;} \end{aligned} \tag{37a}$$

$$\begin{aligned} & \text{a summation of } M \text{ values distributed over } M \text{ processors can be} \\ & \text{performed in } O(\log M) \text{ communication steps.} \end{aligned} \tag{37b}$$



(a)



(b)

FIG. 6. (a) Spectral element decomposition for a multiply connected domain (element numbers denoted by  $\odot$ ); (b) associated "native" parallel processor.

These two requirements relate directly to the two communication constructs central to our algorithm, direct stiffness summation and vector reduction, respectively. (Note that  $\log$  denotes logarithm base 2 and  $\ln$  denotes logarithm base  $e$ .)

We characterize the "hardware" associated with the processors and communication networks in Fig. 6 by a basic clock cycle for calculation,  $\delta$  (measured, say, in seconds), and the time-per-word required to send  $m$  words across a direct link,

$\Delta(m)$ . It is assumed that data transfer can occur simultaneously over all distinct links. The ratio  $\Delta/\delta$  is denoted  $\sigma(m)$ ;  $\sigma(m)$  is assumed to be a decreasing function of  $m$ , with  $\sigma(1)$  appreciably greater than  $\sigma(\infty)$  due to message startup overhead. Messages travelling more than one link (or "hop") can be penalized in terms of both longer transmission time and potential contention. (Contention represents network imbalance/saturation and arises when more than one potentially parallel communication requires the same link.)

As indicated previously, if no real machines were similar to the hypothetical architecture described above, the resulting analysis would be of fairly limited value. However, there are many architectures which are identical to, or at least very similar to, the native architecture of Fig. 6. In particular, reconfigurable lattices [33] readily satisfy constraints (37), and lattice or hypercube message-passing architectures satisfy all constraints save the assurance of nearest neighbor, contention-free communication. Most importantly, the native processor is in some sense a lowest common denominator for architectures, in that coarser grained, shared-memory and large-cache machines imply algorithm simplifications, not additions. This implies that an algorithm designed to perform effectively on the native machine should, in fact, perform well on a large class of real machines. This will be demonstrated in Section 5.

#### 4.2. Parallel Algorithms and Computational Complexity

We consider here  $M$ -parallel solution of the  $h = (K, N)$  elliptic spectral element discretization (21) by conjugate gradient iteration (30) on the native architecture of Fig. 6. As described in Section 4.2, the performance of the conjugate gradient iteration is determined by the following representative computational kernels:

$$r = \underline{A}u \quad (38a)$$

$$r = \underline{P}u \quad (38b)$$

$$a = (u, u), \quad (38c)$$

corresponding to operator evaluation, diagonal-matrix collocation, and norm (or inner product) calculation, respectively. We now discuss how each of these operations is performed in parallel and present computational complexity estimates [8] for the resulting algorithms.

We begin with the evaluation of a representative term of  $r = \underline{A}u$ , (20), which, by construction, admits the following simple concurrency. First, we calculate an "incomplete" residual  $\hat{r}_h \in Y_h$  in each element concurrently,

$$\hat{r}_{ij}^k = \left\{ \left( \sum_{p=0}^N \hat{A}_{ip}^k \left( \sum_{q=0}^N \hat{B}_{jq}^k u_{pq}^k \right) \right) \right\} \quad \forall k \in \{1, \dots, K\}, \quad \forall ij \in \{0, \dots, N\}^2. \quad (39)$$

(This residual is incomplete in the sense that the element-boundary-node displacements are not admissible; direct stiffness is the process by which appropriate

contributions are added from neighboring-element test functions.) This operation is communication-free and will require

$$c_1 KN^{d+1}/M$$

“wall” clock cycles, where  $c_1$  is defined by (31).

Next, we perform direct stiffness to find the actual nodal residuals  $r_h \in \bar{X}_h$ .

$$r_{ij}^k = \Sigma_{ij}^k r_{ij}^k, \tag{40}$$

where the direct stiffness procedure is described in Fig. 5. The flow of information at vertices in Fig. 5 is clearly not coincident with the possible single-hop flow of information along links in our parallel processor described by Fig. 6. However, an efficient direct stiffness procedure based on nearest-neighbor use of the edge-based communication network can be constructed, as we now describe.

We first consider the simple spectral element mesh shown in Fig. 4a, in which data  $\hat{r}_n \in Y_h$  is given on each element. In Fig. 7 we show diagrammatically how direct stiffness summation can be performed by local directional splitting of the operation into a sequence  $\langle \Sigma \rangle$  of  $d = 2$  element edge exchanges. It can be seen that the nodal values at the vertices are, indeed, correct; that is, the sequence  $\langle \Sigma \rangle$  of Fig. 7 is algebraically equivalent to Fig. 5. The advantages of this splitting method over, say, a bidirectional parallel edge pass followed by vertex-specific operations are: the splitting method is algorithmically clean; the splitting method avoids costly short messages; the splitting method avoids non-nearest-neighbor communication and contention. These advantages are even clearer in three space dimensions.

For a particular spectral element decomposition in  $\mathbf{R}^d$  we denote an associated sequence of  $d$  edge-pass operations  $\langle \Sigma \rangle$  as “regular” if it is algebraically equivalent to the direct stiffness summation of Fig. 5. For many spectral (or substructured finite) element decompositions it is difficult, if not impossible, to find an edge-pass sequence,  $\langle \Sigma \rangle$ , which results in correct nodal values at all vertices. However, it is often relatively simple to find a sequence  $\langle \Sigma \rangle$  for which the number of vertices with incorrect values, denoted “special” nodes, is small. This suggests the following

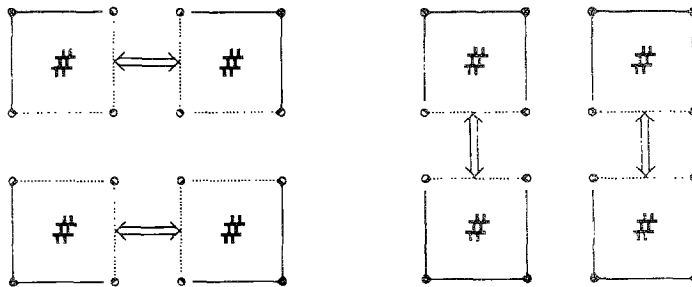


FIG. 7. Direct stiffness summation for the four element configuration of Fig. 4 employing directional factorization.

strategy: first, a vector reduction (sum) operation is performed to accumulate the contributions of all elements to the residuals associated with the special nodes:

second, a standard  $d$ -pass sequence is performed ( $\Sigma$ ); third, the results of the illustrated diagrammatically in Figs. 8b-f for the spectral element mesh shown in Fig. 8a.

If we (suggestively) denote the number of special nodes by  $\varepsilon$ , the number of clock cycles required to perform direct stiffness summation is then

$$c_3 KN^{d-1}/M + c_4 \sigma(N^{d-1}) N^{d-1} + c_5 \sigma(\varepsilon) \varepsilon \log M,$$

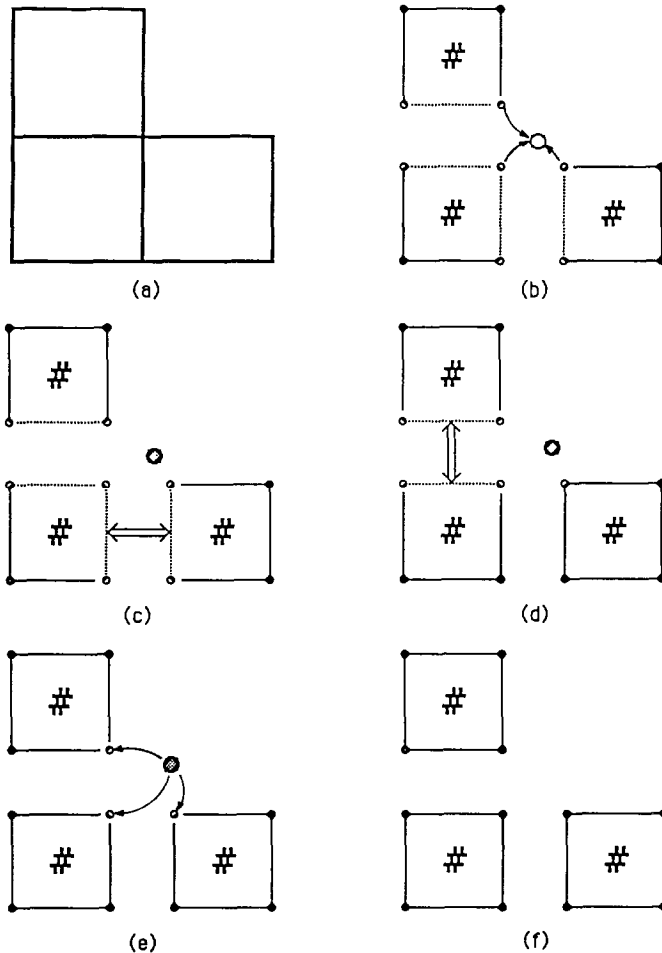


FIG. 8. (a) Spectral element Neumann problem for which standard directional factorization direct stiffness summation fails; the interior corner corresponds to a "special" node. The modified directional factorization procedure involves: (b) gather and summation of special nodes, (c)-(d) standard directional exchange, and (e) redistribution of special nodes. The final data in  $X_n$  is shown in (f).

where the  $c_3$  term (of (31)) represents the summation of all edge values, the  $c_2$  term represents the communication of edge values between processors, and the  $c_5$  term accounts for the special-node treatment. Note that  $c_4$  is independent of  $K$  and  $M$  by virtue of (37a). More details of the direct-stiffness procedure can be found in [34].

The residual calculation (38a) is the most complicated operation in conjugate gradient iteration. The diagonal-collocation operation (38b) is completely concurrent and communication-free, with computational complexity

$$c_6 KN^d/M.$$

Similarly, the inner product (38c) is a standard vector reduction, which is evaluated by first performing intra-element/intra-processor sums, and then evoking an inter-processor vector reduction (37b). The resulting computational complexity is

$$c_7 KN^d/M + c_8 \sigma(1) \log M,$$

where the first and second terms reflect intra- and inter-processor summation, respectively. Note that for inner products of functions  $u_h \in X_h$ , the elemental partial sums must take into account the “multiplicity” of boundary nodes.

On the basis of the preceding analysis we arrive at an approximate expression for the number of wall-clock cycles required to solve (21) by conjugate gradient iteration on  $M$  processors,

$$\begin{aligned} Z_M^A = N_\varepsilon^A \{ & c_1 KN^{d+1}/M + c_2 KN^d/M + c_3 KN^{d-1}/M \\ & + c_4 \sigma(N^{d-1}) N^{d-1} + c_5 \sigma(\varepsilon) \varepsilon \log M + c_8 \sigma(1) \log M \} \end{aligned} \quad (41)$$

We identify the  $c_1$ -,  $c_2$ -, and  $c_3$ -terms of our serial estimate (31); however, there are now three new terms associated with communication and (only)  $\log M$ -parallelizable operations. From (41) and (31) we can derive the inverse parallel speedup,  $S_\tau^{-1} = Z_M^A/Z_1^A$ , in which we keep only the leading order terms in computation and communication,

$$\begin{aligned} S_\tau^{-1} = & 1/M + \alpha \cdot (1/M)(M/K) \sigma(N^{d-1})/N^2 \\ & + \beta_1 \cdot (\log M/M)(M/K) \sigma(1)/N^{d+1} + \beta_2 \cdot (\log M/M)(M/K) \sigma(\varepsilon) \varepsilon/N^{d+1}. \end{aligned} \quad (42)$$

Here  $\alpha, \beta_1, \beta_2$  are constants independent of  $N, K$ , and  $M$ . The estimate (42) is general not only as regards geometry, but also in the fact that it extends to the full Stokes (and Navier–Stokes) problem by virtue of the Uzawa method.

Note that (42) is the inverse speedup for the same algorithm on the same basic processor/memory unit; vectorization is assumed to occur *within* the spectral elements and, thus, scales out of the speedup analysis. Vectorization can be efficiently applied to (39) given the local structure internal to elements, as described in further detail in Section 5.2.

We illustrate our speedup analysis graphically by plotting in Fig. 9 a simplified plot of  $S_r^{-1}$  and its constituents as a function of  $M$  for some representative parameter set ( $K, N, d, \sigma$ ). In particular, we break (42) into three parts: the first term—the calculation term—labelled “ca” in Fig. 9; the second term—the direct stiffness term—labelled “ds”; and the third and fourth terms—the inner product terms—labelled “ip.” The direct stiffness curve is shown to grow slightly with increasing  $M$ , in anticipation of possible network contention associated with an increasing number of processors.

We make several comments concerning the computational complexity (41) and speedup (42). First, in the limit of vanishing communication,  $\sigma = 0$ , we achieve unity-parallel-efficiency performance; this is due to the choice of an intrinsically concurrent iterative procedure. Second, even in the case of non-negligible communication time,  $\sigma \neq 0$ , the method can maintain good performance due to the  $1/N^2$  and  $1/N^{d+1}$  algorithmic ratio of communication to computation; this favorable ratio derives from the geometric decomposition of work, the intrinsic substructuring associated with spectral elements, and the correct choice of boundary-minimal polynomial bases. In essence, *most memory accesses are local*. Third, for a fixed discretization  $h = (K, N)$  there is an optimal number of processors  $M_{opt}$ ,  $1 \leq M_{opt} \leq K$ , that maximizes speedup by trading off concurrency and communication through “super-substructuring” of spectral elements. For instance, if for

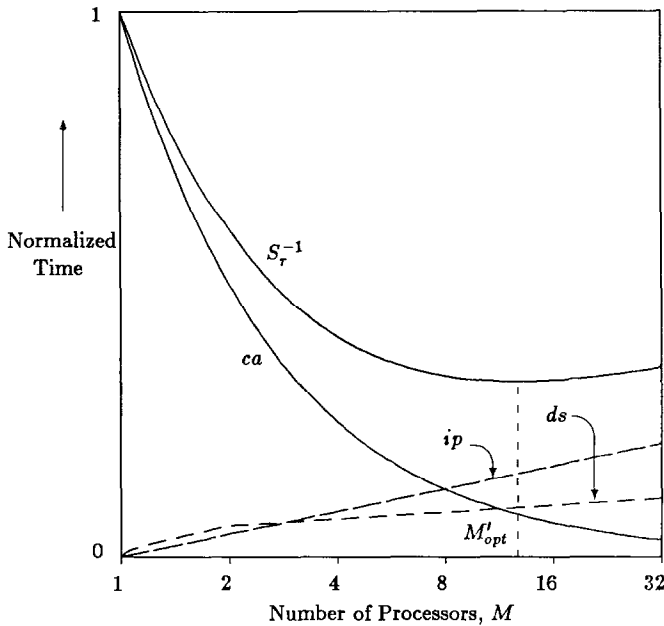


FIG. 9. Illustration of computing times associated with the  $M$ -parallel conjugate gradient solution

illustrative purposes we neglect the  $\beta_2$  term in (42), we find that  $M_{opt} = \max\{1, \min[K, M'_{opt}]\}$ , where  $M'_{opt}$  is the local minimum  $M'_{opt} = KN^{d+1} \ln 2/\beta_1 \sigma(1)$  shown in Fig. 9.

By virtue of the facts that  $M'_{opt}$  scales with  $K$  and that  $M$  is bounded by  $K$  (the spectral element being an “indestructible” data unit), we conclude that the high-order spectral element parallel paradigm is intrinsically *medium-grained* in character; the number of processors and speedup grows with problem size (in contrast to fixed speedup in *coarse-grained* processing); however, the number of degrees-of-freedom per processor remains “large” (in contrast to *fine-grained* processing). In the case where  $M_{opt} = M'_{opt} < K$  it can be argued that the implicit granularity of the spectral element discretization is not the limiting factor in speedup; in the case where  $M'_{opt} > K \rightarrow M_{opt} = K$ , it is clear that the spectral granularity is potentially limiting performance. In the latter situation it is of interest to consider intra-element parallelism.

We close this section by noting that on the basis of our performance estimates (42) we can predict the solution times for the case where the number of processors and number of degrees-of-freedom (elements) grow in fixed proportion. This corresponds to the “scaled” speedup analysis presented in [35], and is, in fact, arguably a more interesting analysis than that of fixed problem size with increasing number of processors, given the general trend in computational fluid mechanics to solve ever larger problems in a reasonable time, rather than fixed-size problems in reduced time. (One instance where this trend does not hold is in the case of parametric studies; however, in such instances one can often exploit parameter-based parallelism.)

Noting that the work requirements per node will scale as

$$w = c_1 KN^{d+1}/M, \tag{43}$$

we can write the speedup for a fixed amount of work per node,  $w$ , as (assuming  $\varepsilon = 0$ ),

$$S_{\tau}^{-1}|_w = \frac{1}{M} \left( 1 + \frac{\alpha\sigma(N^{d-1}) N^{d-1}}{w} + \beta_2 \frac{\sigma(1) \log M}{w} \right), \tag{44}$$

which now tends to zero as  $M \rightarrow \infty$  (i.e., there is no optimal number of processors). Furthermore, as the solution time  $\tau = (w/10^6 s \eta)(S_{\tau}^{-1} M)$ , we see that for fixed  $w$  the problem solution time is *constant* to within small logarithmic constants. In essence, by increasing the problem size ( $K$ ), the inner product curve in Fig. 9 is depressed, thereby allowing a proportionately larger number of processors  $M$  to be *effectively* utilized.

*Remark.* Although we have focussed primarily on the variational structure of our methods, there is, of course, an equivalent linear-algebra interpretation of the parallel work decomposition. To illustrate this algebraic viewpoint we consider the problem of Fig. 2a, for which the matrix structure (11)–(12) is shown explicitly in



Fig. 10. It is seen that  $\underline{A}$  comprises four ( $K=4$ ) full submatrices,  $[\hat{A}^k] = \hat{A}_{pq}^k$ , with each pair of adjoining submatrices coupled by a single overlapping row and column associated with the condition (8a).

We then consider the elemental “incomplete” residual  $[\hat{r}^k] = \hat{r}_j^k = \hat{A}_{ij}^k u_j^k = [\hat{A}^k](u^k)$ . At internal nodes ( $i \in \{1, \dots, N-1\}$ )  $\hat{r}_i^k$  is complete ( $= r_i^k$ ), as there is no overlap between the  $[A^k]$  for these nodes. However, at the element interface between two elements  $k$  and  $k+1$ , the complete residual must be the sum of two inner products:  $(u^k)$  with the last row of  $[A^k]$ ; and  $(u^{k+1})$  with the first row of  $[A^{k+1}]$ . The sum of these two inner products is precisely the sum of the incomplete residuals  $[\hat{r}^k]$  and  $[\hat{r}^{k+1}]$  at their shared node, thus showing how the incomplete residual/direct stiffness procedure is related to the underlying matrix structure.

### 4.3. Comparison with *h*-Type Substructure Methods

The description of spectral element discretizations in Section 2, of iterative solvers in Section 3, and of parallel constructs in Section 4 can be readily extended to *h*-type finite element substructure techniques [4, 5, 12]. We consider first the one-dimensional problem (1), and introduce the finite element substructure discretization parameter,  $\tilde{h} = (\tilde{K}, \tilde{N})$ . The interval  $A$  is divided into  $\tilde{K}$  substructures  $A^k$ , where  $A^k$  is defined by  $a^k \leq x \leq a^k + b$ ; each substructure is, in turn, broken up into  $\tilde{N}$  linear finite elements of equal length,  $b/\tilde{N}$ . The finite element approximation space  $\tilde{X}_h$  is thus given by

$$\tilde{X}_h = \{v \mid v|_{A^k} \in \tilde{\mathbf{P}}_{\tilde{N}}(A^k)\} \cap H_0^1(A), \tag{45}$$

where  $\tilde{\mathbf{P}}_{\tilde{N}}(A^k)$  is defined by

$$\tilde{\mathbf{P}}_{\tilde{N}}(A^k) = \{v \mid v|_{\xi_q^k, \xi_{q+1}^k} \in \mathbf{P}_1(\xi_q^k, \xi_{q+1}^k) \forall q \in \{0, \dots, \tilde{N}-1\}\} \cap H^1(A^k). \tag{46}$$

The finite element “collocation points” are given by  $\xi_q^k = a^k + (\xi_q + 1)b/2$ ,  $1 \leq k \leq \tilde{K}$ ,  $0 \leq q \leq \tilde{N}$ , with  $\xi_q = -1 + 2q/\tilde{N}$  for  $0 \leq q \leq \tilde{N}$ . The finite element equations are then given by (5) and (6), with appropriate modification of the inner products to reflect the low-order space and new collocation points. Note that in one space dimension  $\tilde{h} = (\tilde{K}, \tilde{N})$  is equivalent to  $h = (\tilde{K}\tilde{N}, 1)$ ; we choose the substructure notation to more clearly illustrate common data structures.

$$\begin{bmatrix} [A^1] & & & 0 \\ & [A^2] & & \\ & & [A^3] & \\ 0 & & & [A^4] \end{bmatrix} \begin{pmatrix} (u^1) \\ (u^2) \\ (u^3) \\ (u^4) \end{pmatrix}$$

FIG. 10. Structure of the discrete Laplacian operator,  $\underline{A}$ , in  $\mathbf{R}^1$  for the discretization in Fig. 2a. The submatrices have a single overlapping row and column corresponding to degrees-of-freedom shared at element interfaces.

The local nodal basis for  $\tilde{w}_h \in \tilde{X}_h$  can be written analogously to (7)–(8) as

$$\tilde{w}_h(x)|_{A^k} = \sum_{p=0}^N \tilde{w}_p^k \tilde{h}_p(r) \quad x \in A^k \Rightarrow r \in I \tag{47a}$$

$$\tilde{h}_p \in \{v \mid v|_{\tilde{\xi}_q, \tilde{\xi}_{q+1}} \in \mathbf{P}_1(\tilde{\xi}_q, \tilde{\xi}_{q+1}) \forall q \in \{0, \dots, \tilde{N}-1\}\} \cap H^1(A^k), \tag{47b}$$

$$\tilde{h}_p(\tilde{\xi}_q) = \delta_{pq} \quad \forall p, q \in \{0, \dots, \tilde{N}\}^2, \tag{47c}$$

with  $x \in A^k$  and  $r \in I$  related by  $x = a^k + (r + 1)b/2$ . The global  $H^1$  condition and Dirichlet conditions on the  $\tilde{w}_p^k$  are the same as for the spectral element basis, (8)/Fig. 2; this fact will be critical in evaluating computational complexity. The final finite element equations are very similar to (11)–(12), however there is now a great deal of intra-substructure sparsity in the matrix due to the locally compact support of the low-order finite element space. That is, the submatrix blocks in Fig. 10 which are full for the spectral element discretization are now sparse (in fact, tri-diagonal).

The extension of (45)–(47) to the multi-dimensional case closely parallels that of the spectral element development. We use tensor product spaces and bases, (19)/Fig. 4, arriving at the final set of Eq. (21) and direct stiffness procedure shown in Fig. 5. Note that the tensor product forms are not important in  $h$ -type methods as regards sum factorization (28) ( $\underline{A}\underline{u}$  products are evaluated in terms of local stencils); however, they are important in maintaining local structure. The finite element substructure equations and spectral element equations are readily defined in terms of the same quantities due to their common variational foundation; in fact, substructure  $h$ -type and spectral element methods can be used simultaneously in the same calculation using nonconforming “mortar” methods [36–38].

We can now consider the computational complexity,  $\tilde{Z}_M^A$ , associated with conjugate gradient solution (30) of substructured finite elements on the native medium-grained processor of Fig. 6. For the discretization parameter  $\tilde{h} = (\tilde{K}, \tilde{N})$  and  $M$  processors we find, to leading order,

$$\tilde{Z}_M^A = \tilde{N}_\epsilon^A \{ \tilde{c}_1 \tilde{K} \tilde{N}^d / M + \tilde{c}_4 \sigma (\tilde{N}^{d-1}) \tilde{N}^{d-1} + \tilde{c}_8 \sigma(1) \log M \}, \tag{48}$$

where the significant difference between (48) and (41) is the reduced work needed to evaluate the  $h$ -type residual. Of interest is comparing the time to compute,  $\tau$ , for the spectral element and finite element approximations,  $\rho_M = \tau/\tilde{\tau} = Z_M^A/\tilde{Z}_M^A$ ,

$$\rho_M = \frac{N_\epsilon^A \cdot \{ c_1 K N^{d+1} / M + c_4 \sigma (N^{d-1}) N^{d-1} + c_8 \sigma(1) \log M \}}{\tilde{N}_\epsilon^A \cdot \{ \tilde{c}_1 \tilde{K} \tilde{N}^d / M + \tilde{c}_4 \sigma (\tilde{N}^{d-1}) \tilde{N}^{d-1} + \tilde{c}_8 \sigma(1) \log M \}}, \tag{49}$$

where we assume that the same number of spectral element/substructures,  $K = \tilde{K}$ , and processors,  $M$ , are used in each calculation.

We (plausibly) assume  $\tilde{N}_\epsilon^A = N_\epsilon^A$ , and take  $\tilde{N} = \mu N$ ;  $\mu \geq 1$  as the spectral element approximation will always be at least good as the linear  $h$ -type approximation (recall that the error,  $\epsilon$ , is fixed). We start by considering only the first terms in the numerator and denominator; this ratio is the usual serial work comparison of high-

order and low-order methods,  $\rho_1 = Z_1^A / \tilde{Z}_1^A = N/\mu^d$ . It can be shown that  $\rho_1$  is significantly less than unity for an interesting class of problems, in particular for smaller  $\varepsilon$ , smoother solutions, and higher space dimension  $d$  [23]. Of interest in the context of the current parallel analysis is the fact that the two remaining (communication) terms in the work estimate (49) are at least as large for the low-order method ( $\tilde{c}_4^A$ ,  $\tilde{c}_8$ - terms in the denominator) as for the high-order method ( $c_4$ ,  $c_8$ -

important, the low-order-method communication terms can be larger than their spectral element counterparts by the factor  $\mu^{d-1}$ . We thus see that the relative advantage of high-order methods *improves* in a medium-grained parallel environment, due to the fundamental fact that communication is independent of scheme order if proper boundary-minimal bases are chosen; this is a general argument for high-order methods, and need not be restricted to  $p$ -type ( $N \rightarrow \infty$ ) convergence strategies.

Lastly, it should be noted that the  $\tilde{h} = (\tilde{K}, \tilde{N})$  description of the finite element discretization is artificial in that it imposes a coarser granularity on the problem than is actually present. The preceding analysis is thus only directly relevant when  $M'_{\text{opt}} < K$ . When this condition is not satisfied, the finite element substructure approach can be readily refined due to the homogeneity of the approximation, whereas extension of the spectral element method to intra-element parallelism is less straight forward. Our conclusions for the medium-grained paradigm should, therefore, not be applied to the fine-grained case without additional analysis and without further specification of the cost objective function.

#### 4.4. Architecture Mappings: Message-Passing Hypercube

In this section we consider how the native parallel processor defined in Fig. 6 and analyzed in Sections 4.1–4.3 maps to message-passing hypercube architectures. (The arguments should apply with little modification to message-passing lattices; mapping to a more natural but less general-purpose and less commercially developed reconfigurable lattice structure is discussed in [32, 38, 39].) We recall that a medium-grained hypercube network is defined by  $M = 2^D$  “large” processors,  $P_p$ ,  $p = 1, \dots, M$ , with a direct link between any two processors  $P_p$  and  $P_q$  for which  $p - 1$  and  $q - 1$  differ only in one bit in their binary representation. The topological properties of hypercubes are summarized in [9], and numerous applications of hypercubes are described in [40, 41].

We assume that the spectral elements have been distributed amongst the processors according to some partition  $E_q$ ,  $q = 1, \dots, M$ . If we compare an arbitrary partition on the hypercube to the ideal partition on our model parallel processor of Fig. 6, our communication estimates (41) will be modified by the introduction of non-nearest-neighbor communication (non-unity-dilation mappings) and possible contention (network load imbalance/saturation). In general, it will not be possible to find a mapping for which there exists a direct link in the hypercube for every direct link in our model processor. That is, the hypercube architecture violates assumption (37a).

The first, and most obvious, effect of not satisfying (37a), with which we associate a multiplier  $\lambda_1$ , is that the direct stiffness summation will require more communication steps due to the lack of direct links between element pairs assumed in the ideal model of Fig. 6. The second effect, with which we associate a multiplier  $\lambda_2$ , derives from the fact that a particular hypercube partition  $E_p$  may give physically adjacent spectral elements non-nearest-neighbor positions in the hypercube network. This will potentially increase the transmission time between these spectral elements in the direct stiffness summation procedure: the magnitude of the deterioration will depend on the message-passing protocol. For the case of store-and-forward we expect a maximum increase in transmission time of  $O(\log M)$ ; for the case of wormhole or pipeline routing we expect substantially less deterioration. The third effect, with which we associate a multiplier  $\lambda_3$ , is the fact that, in the absence of direct links between communicating elements, contention can occur during routing through the hypercube. This effect can be quite difficult to quantify, in particular for general partitions on large cubes.

We denote that all of these effects are associated with the direct stiffness term of (41); the  $\log M$  communication terms are unaffected by the hypercube mapping as the hypercube architecture honors (37b) by virtue of simple binary-tree-like embeddings [9]. We thus arrive at our new estimate for speedup for the hypercube system,

$$S_{\tau}^{-1} = 1/M + \lambda_1 \lambda_2 \lambda_3 \cdot \alpha \cdot (1/M)(M/K) \sigma(N^{d-1})/N^2 \\ + \beta_1 \cdot (\log M/M)(M/K) \sigma(1)/N^{d+1} + \beta_2 \cdot (\log M/M)(M/K) \sigma(\varepsilon) \varepsilon/N^{d+1}, \quad (50)$$

in which only the direct stiffness term is modified. This speedup model will serve to interpret the hypercube computational results to be presented in the next section.

The above considerations suggest that the spectral element-to-hypercube partition can lead to computational inefficiencies. Although on computers with fast communication and direct routing these effects may not be leading order, it is likely that computation speeds will always outpace off-board communication rates, and that these mapping issues should therefore not be ignored. We briefly discuss here several fairly standard mapping strategies. The first strategy, an intra-processor strategy,  $S1_{\text{intra}}$ , attempts to partition elements such that members of  $E_q$  share edges; this reduces  $\lambda_1$ . Furthermore, this intra-processor strategy promotes inter-element nearest-neighbor mappings,  $S1_{\text{inter}}$ , which reduces  $\lambda_2$  and  $\lambda_3$ . The second intra-processor strategy  $S2_{\text{intra}}$ , randomly partitions the elements to form the  $E_q$ ; the motivation behind this strategy is to render the calculation load-balance-intensive with respect to local mesh refinement [42]. Although we do not consider refinement-induced load imbalance in this paper, it is certainly an important issue. The strategy  $S2_{\text{intra}}$  does not preclude subsequent attempts at  $S1_{\text{inter}}$ ; however, it certainly makes the task difficult, and one must conclude that  $S2_{\text{intra}}$  will tend to increase not only  $\lambda_1$ , but also  $\lambda_2$  and  $\lambda_3$ . Heuristics for achieving these strategies are described in [34, 42, 43].

#### 4.5. Performance Measures and Optimality

The speedup  $S_\tau$  and parallel efficiency,  $\eta = S_\tau/M$ , only signify the extent to which a particular discretization-solver can exploit multiple copies of a particular processor with a particular communication system. Although  $S_\tau$  can be effectively used to evaluate optimal operation on a particular machine, its value as an absolute performance measure is limited. First, if a value of  $S_\tau$  on one computer is to have any significance on a second computer, the nondimensional similarity variable,  $\sigma$ , must be similar in the two cases [8]. Second, we note that large speedup or parallel efficiency does not necessarily imply small compute time  $\tau$  ( $=\delta Z_1^A/S_\tau$ ) or run-time cost  $c$  ( $=\tau fC/t_{\text{dep}}$ ); the former depends on  $\delta$  and  $Z_1^A$ , whereas the latter depends on both  $\tau$  and  $C$ . (The parameter  $f$  here is the ratio  $M/M_{\text{max}}$ , that is, the fraction of available processors on a machine being dedicated to a particular calculation.) For instance, by increasing  $\delta$  one decreases  $\sigma$ , thereby increasing parallel efficiency (for fixed  $M$ ) while simultaneously increasing  $\tau$ . Similarly, although one might achieve high speedup with a readily parallelized “poor” algorithm (large  $Z_1^A$ ), the resulting compute time  $\tau$  might be larger than that for a less parallelizable but “better” algorithm (small  $Z_1^A$ ). Furthermore, if one obtains high parallel efficiency due to a small  $\sigma$  that derives from a costly switching system, high parallel efficiency need not imply low cost  $c$ .

It thus follows that to evaluate the performance of different computers with respect to our particular algorithm we should compare some measure of  $\tau$  and  $c$  directly. To this end, we rewrite  $\tau$  as  $\tau = Z_1^A/10^6 c_\nu s'$ , where  $Z_1^A = c_\nu W$  ( $c_\nu$  relates clock cycles to operations, e.g., through vectorization), and  $s'$  is the *actual* speed achieved,  $s' = S_\tau/10^6 \delta c_\nu$ , in MFLOPS. We can then construct a plot analogous to Fig. 1 in which we characterize the performance of an algorithm on different computers by a point in  $s' - e'$  space, where  $e' = s'/fC$ . As for Fig. 1, an algorithm-architecture point  $A$  is better than any other point  $B$  if  $A$  is in the quadrant above and to the right of  $B$ ; the  $s' - e'$  space is different from Fig. 1 in that it now includes non-constant algorithm-architecture interactions through the dependence of  $s'$  on speedup,  $S_\tau$ , and  $c_\nu$ .

The  $s' - e'$  characterization of an algorithm-architecture is preferable to direct reporting of  $\tau$ ,  $c$  in that it is a more universal measure of performance; for instance, the number of iterations,  $N_e^A$ , and even the base (serial) work per iteration, scales out of the  $s'$  rating. However, by the same arguments, the  $s' - e'$  representation can be misleading. First, one can easily construct methods based on algorithms which are highly concurrent but poorly convergent (in discretization parameter,  $h$ , and iteration number,  $N_e^A$ ), for which high  $s'$  does not reflect low  $\tau$ . Second, a poor algorithm-architecture coupling, that is low efficiency, need not imply either a poor algorithm or a poor architecture; in comparing the performance of two computers one should use the best possible  $\tau$ -minimizing algorithm on each computer. In order to address these two reservations, we shall supplement our  $s' - e'$  data with  $\tau$  and  $\eta$ , respectively.

## 5. COMPUTATIONAL RESULTS

### 5.1. General Implementation

In the same way that we decompose the algorithm development into distinct stages, the software development process is also broken up into generic and machine-specific stages. In the first stage (analogous to the first step of the development process), we write a serial code in which, in the bulk of the code, the element number,  $k$ —the parallelization index—is only referenced as a counter within the element set  $E$ , with no relational or logical significance. All other operations (i.e., those that will typically require communication) are relegated to isolated subroutines. For our algorithms the latter include only the direct stiffness algorithm and vector reduction (inner products and norms). In the second stage of the development process, we port the serial code to a particular MIMD parallel machine by: writing device drivers for the communication subroutines; descending identical copies of the serial program to all processors; defining the workload for processor  $P_q$  by replacing  $E$  of the serial code with  $E_q$ . It is important to note that most of the effort is in stage one, which is machine-independent, with subsequent porting activities in stage two being relatively simple.

We note that our software model is SPMD (single program multiple data), but requiring a MIMD rather than SIMD environment. The former appears to have real advantages over the latter in flexibility, particularly if automatic refinement procedures are to be considered.

### 5.2. Intel Vector Hypercube Calculations

We have implemented our methods on the Intel vector hypercubes, the iPSC/1-VX/d4 and its successor, the iPSC/2-VX/d4 (d4 denoting dimension  $D = 4$ , that is  $M_{\max} = 16$ ). The iPSC/1-VX is a 286-based system with store-and-forward message-passing; the iPSC/2-VX is a 386-based system with pipelined communication routing. In both cases the same vector hardware is used, capable of a peak speed of 10 MFLOPS/board. The two machines differ primarily in scalar speed and communication speed and robustness, with the iPSC/2 representing a significant improvement in both capabilities due to advances in technology and architecture. The iPSC/1 (iPSC/2) 286-based (386-based) mother board achieves .02 (.06) MFLOPS, and communication rates of  $\Delta(1), \Delta(\infty) = 5960\mu\text{s}, 33\mu\text{s}$  ( $300\mu\text{s}, 1.4\mu\text{s}$ ). The iPSC software environment is MIMD, requiring explicit (i.e., non-automatic) parallelization and data management.

We now analyze the spectral element-Intel vector hypercube algorithm-architecture coupling based on the framework of Section 4.5 and the complexity estimates of Section 4.2 and 4.4. We begin by analyzing the simple three-dimensional "chain" shown in Fig. 11, with periodic boundary conditions imposed on all sides. We consider six problems of increasing size,  $K = 1, 2, 4, 8, 16$ , and 32, respectively, with  $N = 10$  in all cases; the partitions  $E_q$  for each problem are given in Table III. Note that for a particular  $K$  the number of processors that can be used is limited by three

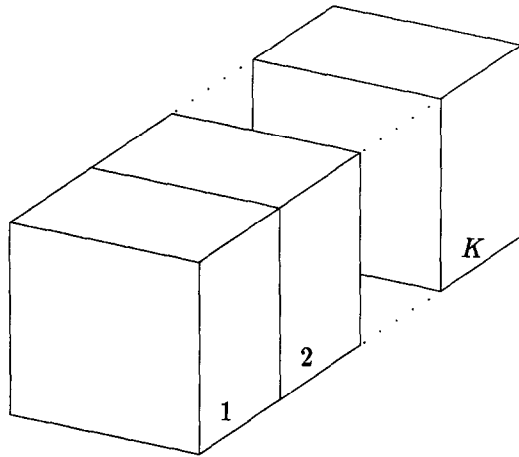


FIG. 11. Periodic chain of  $K$  elements ( $N=10$ ,  $d=3$ ) used for multi-processor timing analysis.

factors: memory constraints preclude  $M < K/2$ ; machine size precludes  $M > M_{\max}$ ; and algorithm granularity precludes  $M > K$ . By virtue of the Gray-code mapping [9] used for the partitions  $E_q$  the hypercube implementation maps exactly to our model processor system,  $\lambda_1 = \lambda_2 = \lambda_3 = 1$ . (Note that communication between faces of elements on the same processor do not pass through the network.)

We tabulate the results of our numerical experiments for the iPSC/1-VX in

TABLE III

Element Distribution  $E_q$ ,  $q=1, \dots, 16$ , for the Periodic Chain of Fig. 11

Processor number	$K/M=1$					$K/M=2$				
	$K=1$	$K=2$	$K=4$	$K=8$	$K=16$	$K=2$	$K=4$	$K=8$	$K=16$	$K=32$
1	1	1	1	1	1	1,2	1,2	1,2	1,2	1,2
2		2	2	2	2		3,4	3,4	3,4	3,4
3			4	4	4			7,8	7,8	7,8
4			3	3	3			5,6	5,6	5,6
5				8	8				15,16	15,16
6				7	7				13,14	13,14
7				5	5				9,10	9,10
8				6	6				11,12	11,12
9					16					31,32
10					15					29,30
11					13					25,26
12					14					27,28
13					9					17,18
14					10					19,20
15					12					23,24
16					11					21,22

Table IV as a table of  $T(K, M)$ ,  $T_{ca}(K, M)$ ,  $T_{ds}(K, M)$ ,  $T_{ip}(K, M)$ . Here  $T$  is the time to calculate 250 conjugate gradient iterations for the  $A$  system, (30), and  $T_{ca}$ ,  $T_{ds}$ ,  $T_{ip}$  represent the breakdown of  $T(K, M)$  in terms of calculation time, direct stiffness communication time, and inner product communication time. 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  $T$ ,

$$f_T(K, M) = aK/M + (b + c \log M) \cdot (1 - \delta_{1,M}), \quad (51)$$

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  $T$  of Table IV, finding  $a = 9.2$  s,  $b = 3.1$  s, and  $c = 6.2$  s; these values are not inconsistent with the direct breakdown of  $T(K, M)$  into  $T_{ca}$  ( $a$ -term),  $T_{ds}$  ( $b$ -term), and  $T_{ip}$  ( $c$ -term), which serves to verify the form of (51). Note also the constancy of  $T_{ds}$  for  $M \geq 4$ .

From (51) we calculate the inverse speedup,  $S^{-1} = f_T(K, M)/f_T(K, 1)$ , which is plotted in Fig. 12, also plotted are the measured speedups for the data of Table IV,  $T(K, M)/f_T(K, 1)$ . The reasonably good fit of (51) to the data is further verification of the model. We make several comments concerning the speedup curve of Fig. 12. First, the optimal number of processors,  $M'_{opt}$ , is less than  $K$ ; furthermore, the ratio  $M'_{opt}/K$  is roughly constant, as predicted by the models of the last section. The fact that  $M'_{opt} < K$  implies that for this machine, which is a fast calculator and a slow communicator ( $\sigma$  relatively large), the spectral element granularity is not limiting. Second, the speedup grows with problem size, as must be the case (the dashed-line contributions in Fig. 9 being less important as  $K$  increases). Third, the maximum speedup on 16 processors is roughly 5.0, corresponding to a parallel efficiency of  $\eta = 0.3$ .

We now repeat the chain experiment, but now on the iPSC/2-VX machine. We show in Table V and Fig. 13 the iPSC/2 analogues of the iPSC/1, Table IV and

TABLE IV  
iPSC/1-VX Timing Results for 250  $A$  Iterations,  $N = 10$ ,  $d = 3$

$K/M$	Time (s)	Time <sub>ca</sub> (s)	Time <sub>ds</sub> (s)	Time <sub>ip</sub> (s)
1/1	9.7	8.9	0.40	0.40
2/2	14.0	9.0	2.8	2.3
4/4	24.8	8.5	9.1	7.5
8/8	31.7	8.6	9.0	14.0
16/16	37.1	8.6	8.5	19.8
2/1	18.5	17.7	0.40	0.37
4/2	22.9	17.6	2.6	2.3
8/4	33.4	17.2	7.3	7.7
16/8	40.1	17.4	8.5	14.2
32/16	46.6	17.4	8.1	20.2



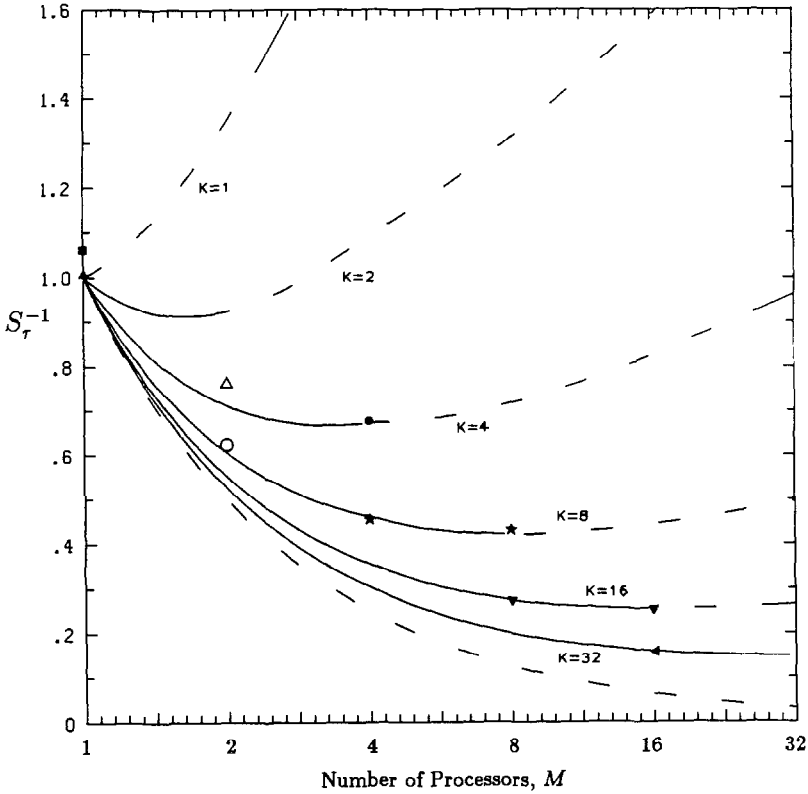


FIG. 12. Inverse speed-up on the iPSC/1-VX for 250  $A$  matrix iterations of the spectral element configuration in Fig. 11 for problems defined by  $K=1, 2, 4, 8, 16, 32$ . The solid line indicates the fit  $f_T(K, M)/f_T(K, 1)$  to the data of Table IV; the symbols represent the actual data. Open symbols indicate the data points for the  $M=2$  cases which are anomalous due to the message passing protocol; these points are not used in computing the fit. The upper dashed lines indicate the (unobtainable) operating regime where  $M > K$ . The lower dashed line is the peak theoretical speedup,  $1/M$ .

Fig. 12, respectively. First, as before, the observed performance follows closely that predicted by our models; we find the coefficients in (51) to be  $a = 8.1$  s,  $b = 0.46$  s, and  $c = 0.32$  s. The iPSC/2 is a sufficiently fast communicator that  $M'_{\text{opt}}$  is now greater than  $K$ , indicating that, for this machine, finer grain algorithms may be of interest. Second, the maximum speedup of 14 on 16 processors is much larger than for the iPSC/1 (corresponding to an efficiency of  $\eta = 0.88$ ) due to a decrease in  $\sigma$ —this increase in speedup is significant (i.e., results in a decrease in  $\tau$ ) as it originates in a decrease in  $A$ , not an increase in  $\delta$ .

To investigate “non-idealities,” we have considered two additional tests of the  $N=10, K=32, M=16$  problem on the iPSC/1. In the first test, we replace the partition of Table III with the partition  $E_q = \{2q-1, 2q\}$ , in which we now have a non-Gray ordering, but the amount of data passed across the network is unchanged

TABLE V  
iPSC/2-VX Timing Results for 250 A Iterations,  $N = 10$ ,  $d = 3$

$K \backslash M$	Time (s)	Time <sub>ca</sub> (s)	Time <sub>ds</sub> (s)	Time <sub>ip</sub> (s)
1:1	8.3	8.2	0.06	0.06
2:2	9.4	8.3	0.70	0.39
4:4	9.8	8.1	0.68	0.97
8:8	10.1	8.4	0.53	1.15
16:16	10.4	8.3	0.62	1.53
2:1	16.3	16.2	0.06	0.06
4:2	17.5	16.1	0.83	0.56
8:4	17.8	16.1	0.76	0.94
16:8	18.1	15.9	0.79	1.47
32:16	18.4	16.1	0.76	1.59

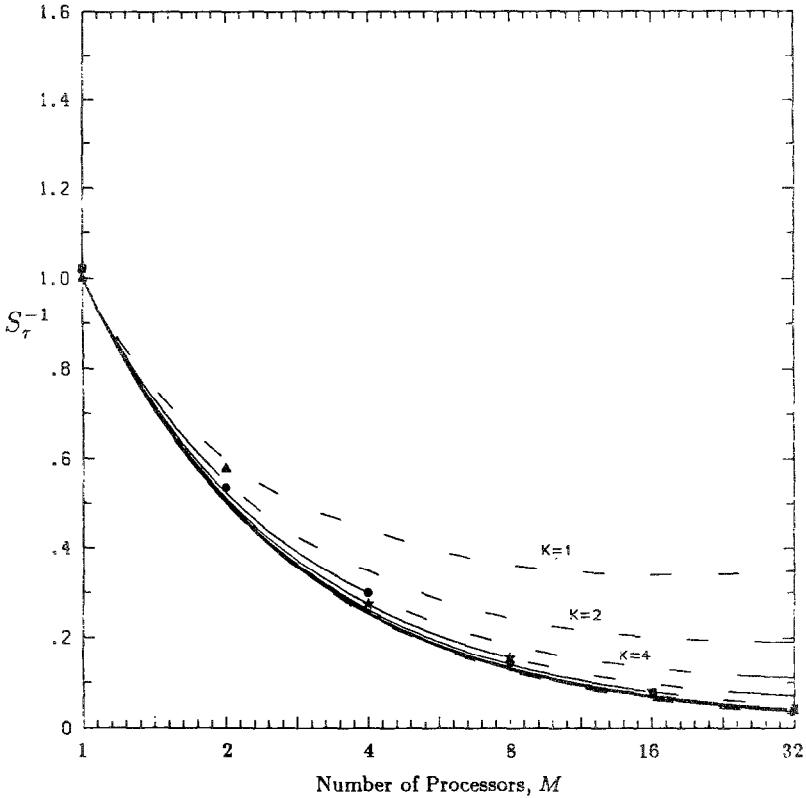


FIG. 13. Inverse speed-up on the iPSC/2-VX for 250 A matrix iterations of the spectral element configuration in Fig. 11 for problems defined by  $K = 1, 2, 4, 8, 16, 32$ . The iPSC/2-VX performs significantly better than the iPSC/1-VX due to decreased communication time.

(that is,  $\lambda_1$  is still unity, but  $\lambda_2, \lambda_3$  are potentially greater than unity). In this case  $T_{ds}$  (and hence  $T$ ) are increased by 9 s to  $T_{ds} = 26$  s ( $T = 55$  s), resulting in a 17% decrease in speedup. In the second test, we replace the partition of Table III with the  $S2_{\text{intra}}$  partition  $E_q: E_1, E_2, \dots, E_{16} = \{1, 3\}, \{2, 4\}, \{5, 7\}, \{6, 8\}, \{9, 10\}, \{11, 13\}, \{12, 14\}, \{15, 16\}, \{17, 18\}, \{19, 21\}, \{20, 22\}, \{23, 24\}, \{25, 26\}, \{27, 28\}, \{29, 30\}, \{31, 32\}$ , in which we now not only have a non-Gray ordering, but also require twice the amount of data to flow across the network (that is,  $\lambda_1, \lambda_2$ , and  $\lambda_3$  are all potentially greater than unity). In this case  $T_{ds}$  (and hence  $T$ ) are increased by 11 s to  $T_{ds} = 28$  s ( $T = 57$  s), resulting in a 20% decrease in speedup. Both of these effects are significantly less on the iPSC/2 (decrease in speedups of less than 1%) due not only to decreased  $A$ , but also to wormhole routing. The latter is very significant in de-sensitizing the calculation to mapping. We conclude that non-idealities as regards mappings are significant but not dominant.

We finish our analysis of the chain problem by plotting in Fig. 14 the results

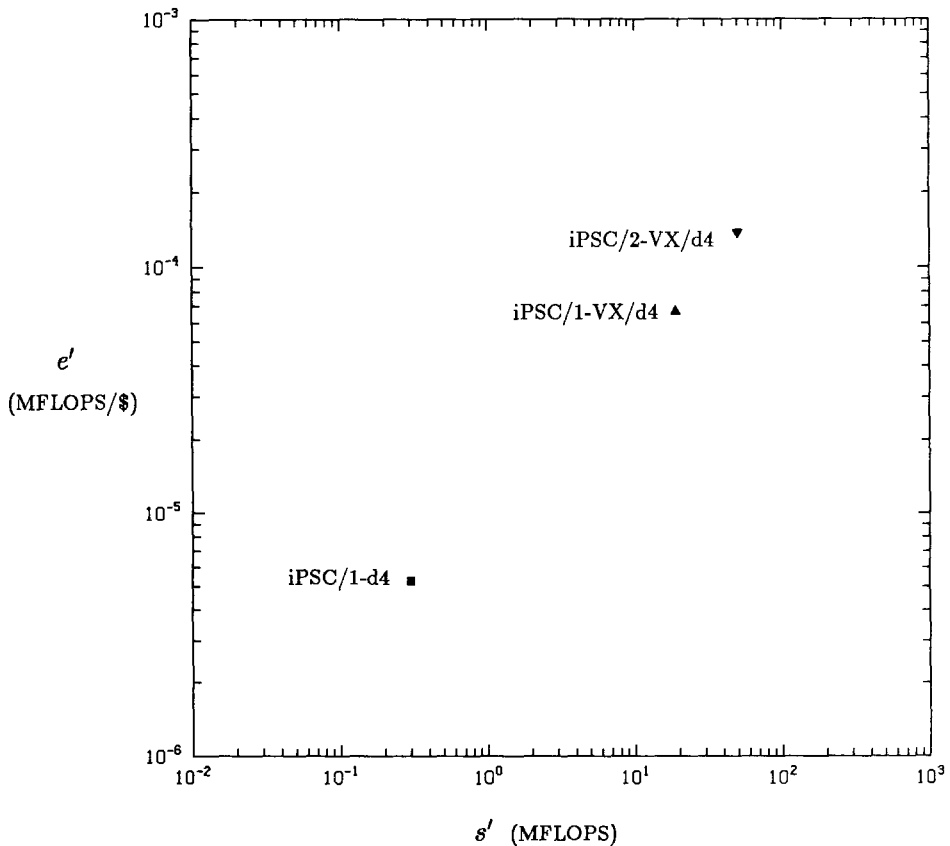


FIG. 14. Computational resource efficiency for the  $K=32$  chain problem of Fig. 11 for the iPSC/1-d4, iPSC/1-VX/d4, and iPSC/2-VX/d4 hypercubes ( $M=16$ ).

for  $f_T(K=32, M=16)$  in  $s' - e'$  space:  $s'$  is calculated as  $(0.1)\text{MFLOPS} \cdot T(K)_{\mu\text{VAX}}/T(K, M)$ , where  $T(K)_{\mu\text{VAX}}$  is the timing on the DEC  $\mu\text{VAX-II}$  ( $T(K)_{\mu\text{VAX}} = 275$  s for 250  $\Delta$  iterations per element), and 0.1 MFLOPS is taken to be the application-independent speed rating of the  $\mu\text{VAX-II}$ ;  $e'$  is calculated from  $s'$  and the cost data summarized in Appendix A. It is seen that the hypercube  $s' - e'$  point is indeed interesting in that it achieves near supercomputer performance at a fraction of the cost. To illustrate the importance of the  $s' - e'$  framework, we have also included the data point for the  $K=32, M=16$  problem on the *nonvector* iPSC/1; although the parallel efficiency on the nonvector machine is close to unity ( $\eta > 0.99$ ), the nonvector machine is obviously uninteresting compared to its vector counterpart. This is due to the fact that the nonvector machine achieves high efficiency due to a decrease in  $\sigma$  brought about by an increase in  $\delta$ , not a decrease in  $\Delta$ .

It is apparent from the nonvector iPSC/1 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. It should be noted that, despite this natural hierarchy, vectorization and parallelization are not independent opportunities. For example, on a vector machine vector lengths for matrix multiples of the form (28) in  $\mathbf{R}^3$  are typically of length  $KN^2$ ,  $KN$ , and  $N^2$  (depending on spatial direction, assuming no transposes), whereas on a parallel machine the same operations entail vectors of length  $KN^2/M$ ,  $KN/M$ , and  $N^2$ . This problem is not too serious, in particular given the smaller vector start-up costs of most new machines; the current iPSC vector board achieves the peak speed of 10 MFLOPS for matrix multiples (28) independent of spatial direction by vectorizing over the inner product (vs. row) index. Our codes achieve 3–4 MFLOPS per node out of a possible 10 MFLOPS vector; the iPSC/2 is slightly faster per node given the faster scalar speed.

As a major point of this paper is the development of general methods for "real" fluid flow problems, we conclude with the solution of a full three-dimensional Stokes problem. We consider the geometry of Fig. 15, with periodic boundary conditions in the flow direction, and no-slip boundary conditions on all solid walls. The discretization parameter is taken to be  $h = (K=32, N=10)$ , and the problem is solved on  $M=16$  processors. The  $S1_{\text{intra}}$  and  $S1_{\text{inter}}$  strategies are pursued so as to achieve a nearest-neighbor mapping, thereby minimizing  $\lambda_1, \lambda_2$ , and  $\lambda_3$ . The element to processor mapping is an extension of that depicted in Fig. 6; pairs of vertically adjacent elements are placed on each processor, and a copy of the mapping is repeated on processors  $P_8, \dots, P_{16}$  to effect nearest neighbor communication between the upper and lower levels of elements. The results of the calculation are shown in Fig. 16 in terms of the velocity field.

On the basis of timings similar to those described for the "chain" problem we plot in Fig. 17 the  $s' - e'$  points for this calculation on the  $\mu\text{VAX}$ , CRAY-2/4-256, IPSC/1-VX/d4, and iPSC/2-VX/d4 computers; the actual timing data is given in Appendix B (Note that all  $e'$  data is computed based on the full  $M_{\text{max}}$  configuration described in Appendix A with  $f = M/M_{\text{max}}$ ). As expected, the full Stokes solver

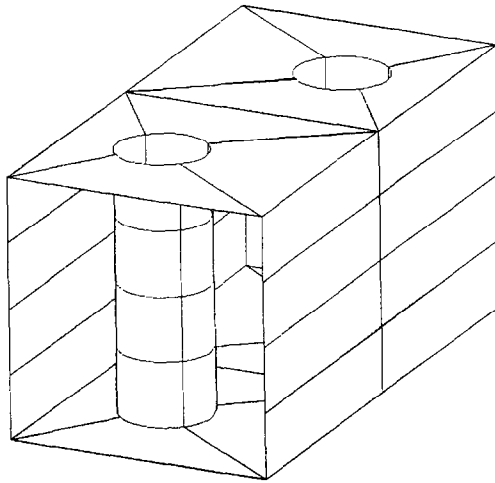


FIG. 15. The  $K=32$  spectral element domain for the steady Stokes problem of flow past two cylinders in a duct.

behaves in a similar fashion to the elliptic solves, with the various machines performing as for the chain problem. The most important conclusion from Fig. 17 is that the new generation of parallel machines is able to achieve supercomputer speeds at a cost advantage of order ten-to-one. Furthermore, this performance is scalable, following the nearly-constant-time rule described by equation (44) (see Table V), indicating that in both  $\tau$  and  $c$  distributed-memory parallel machines have much to offer compared to their vector or serial counterparts.

The CRAY-2/4 results are of interest in their own right. The CRAY-2/4 is a

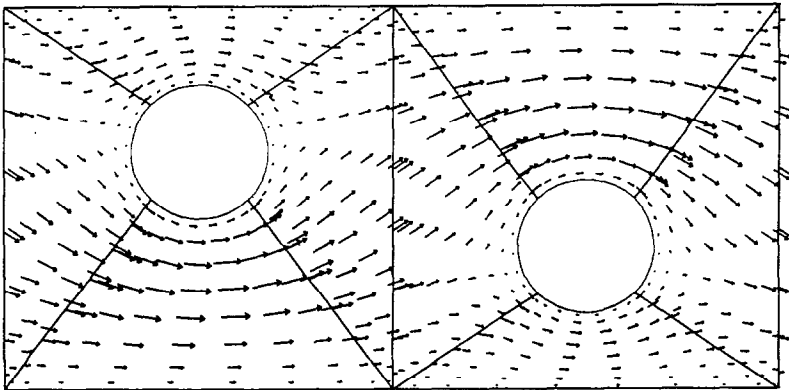


FIG. 16. Velocity vectors at the mid-plane of the domain shown in Fig. 15.

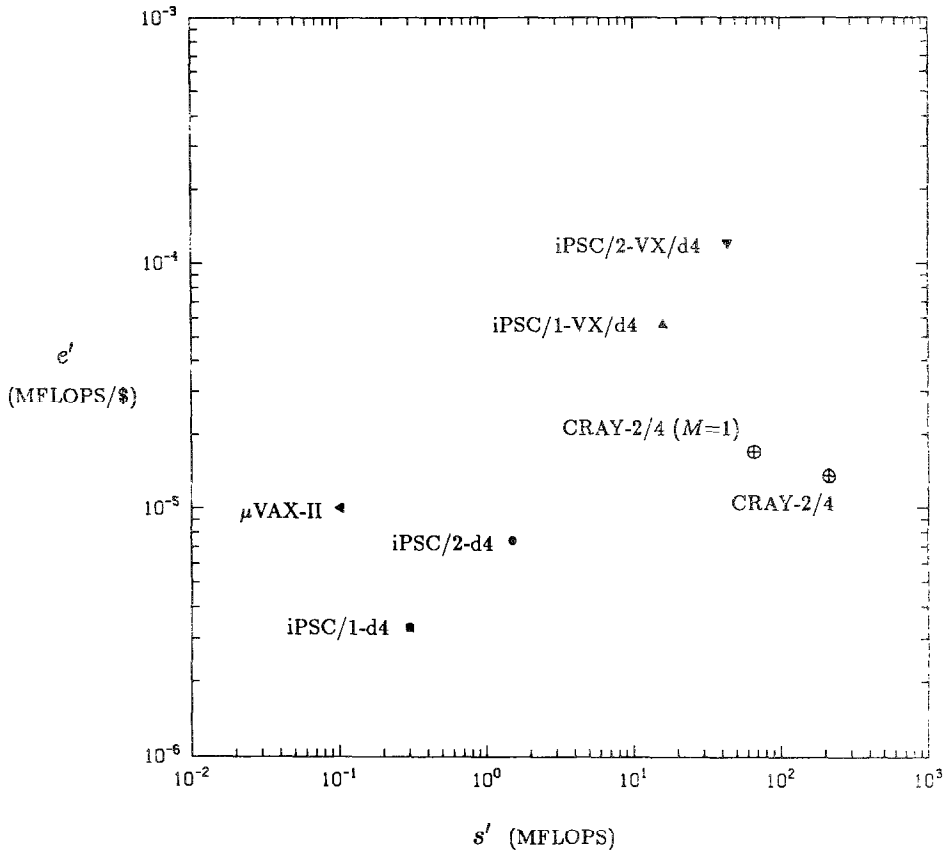


FIG. 17. Measured computational resource efficiency,  $e'$ , for the Stokes problem of Figs. 15 and 16. Solution times are given in Appendix B; in all cases  $M_{\max}$  processors are used, save for the CRAY-2,4 ( $M=1$ ) calculation, in which a single processor is used.

coarse-grained (four-headed) shared-memory machine, typically focussed on through-put parallelism, in contrast to the medium-grained distributed memory speedup machines that are the primary concern of this paper. The CRAY-2 parallelism is motivated primarily by cost-effectiveness, that is, sharing of memory costs; however, it is clear that the potential of  $4 \times$  speedup is significant. We believe that the good performance of our algorithm on the CRAY-2/4 (parallel efficiency of 80% on four processors) reflects the success of our lowest common denominator approach to parallelism and software development.

The conclusion that the iPSC/2 and related medium-grained distributed memory machines are superior to current vector supercomputers is clearly open to much criticism. First, vector supercomputers are much more easily programmed, as are coarse-grained parallel machines (the CRAY-2 results were achieved with auto-tasking); this implies greatly reduced development time. Second, and relatedly, the

presence of shared memory and fast random access allows many algorithms not appropriate in a distributed-memory environment to run effectively on a vector or coarse-grained system. However, there are also other reasons that support the parallel argument; large, expensive systems require multiple-user support, implying that the effective MFLOPS rate for conventional machines can, in fact, be much *less* than that indicated in Fig. 17. In summary, we feel that Fig. 17 does correctly indicate the architectural approach best suited for the problems of large-scale engineering and scientific computation, but that this conclusion must be tempered by due consideration of algorithm and software development issues.

The results of Fig. 17 indicate that properly designed numerical algorithms can solve real problems on parallel processors at serial-supercomputer speeds, using only a fraction of serial-supercomputer resources. Full unsteady Navier–Stokes calculations have recently been performed on the iPSC/2-VX (with up to 64 processors) in a number of different configurations, including rotating flows, natural convection, the von Karman vortex street, vortex breakdown, and the horseshoe vortex [34, 44–46].

### CONCLUSIONS

The discretization-solver-architecture (or algorithm-architecture) optimization problem is an order of magnitude more difficult than the still unsolved discretization-solver problem of classical numerical analysis. The introduction of architectural considerations not only creates new algorithmic issues, but also brings into the analysis complex economic issues not readily quantified or modelled; it is clear that the algorithm-architecture optimization problem will ultimately be solved by intelligent evolution, not by explicit analysis. We have presented here a rational framework in which to evaluate parallel methods and have proposed an algorithm-architecture coupling for flow problems which is, if not optimal, certainly respectable.

### APPENDIX A

Speed and Cost Data for Several Modern Computer Systems

	Price (K\$)	$M_{\max}$ , # of procs.	Rated peak $s$ (MFLOPS)	$\epsilon$ (MFLOPS/S)
iPSC/1-d4	91	16	0.3	$0.33 \times 10^{-5}$
iPSC/1-VX/d4	286	16	160	$55.9 \times 10^{-5}$
iPSC/2(4M)/d4	203	16	1.8	$0.87 \times 10^{-5}$
iPSC/2-VX/d4	363	16	160	$44.1 \times 10^{-5}$
CRAY-2/4-256	15500	4	1000	$6.45 \times 10^{-5}$
$\mu$ VAX-II	10	1	0.1	$1.0 \times 10^{-5}$
FPS-164	500	1	5.0	$1.0 \times 10^{-5}$

## APPENDIX B

Timing Results for 80,000 Degree-of-Freedom Stokes Problem

	$T_{\text{solve}}$ (seconds)	Parallel efficiency $\eta$ , (%)	$s'$ (MFLOPS)	$e'$ (MFLOPS/\$)
iPSC/1-d4	19100	99	0.3	$0.33 \times 10^{-5}$
iPSC/1-VX/d4	360	25	16	$5.6 \times 10^{-5}$
iPSC/2(4M).d4	5760	99	1.0	$0.47 \times 10^{-5}$
iPSC/2-VX/d4	130	75	44	$12.1 \times 10^{-5}$
CRAY-2/4-256(1)	87	100	66	$1.70 \times 10^{-5}$
CRAY-2/4-256(4)	27	80	211	$1.36 \times 10^{-5}$
$\mu$ VAX-II	57200	100	0.1	$1.0 \times 10^{-5}$

## ACKNOWLEDGMENTS

We acknowledge the significant contributions to this work by Einar Rønquist of MIT, by David Scott and Justin Rattner of Intel Scientific Computers, and by Charles Finan of Cray Research. This work was supported by the ONR and DARPA under Contracts N00014-85-K-0208, N00014-88-K-0188, and N00014-89-J-1610, by the NSF under Grants DMC-8704357 and ASC-8806925, and by Intel Scientific Computers. Some calculations were performed on the MIT Supercomputer Facility Cray-2.

## REFERENCES

1. T. F. CHAN, Y. SAAD, AND M. H. SCHULTZ, in *Hypercube Multiprocessors 1986*, edited by M. T. Heath (SIAM, Philadelphia, 1986), p. 196.
2. O. A. MCBRYAN AND E. F. VAN DE VELDE, in *Selected Papers from the Second Conference on Parallel Processing for Scientific Computing*, edited by C. W. Gear and R. G. Voigt (SIAM, Philadelphia, 1987), p. s227.
3. T. F. CHAN AND D. C. RESASCO, in *Selected Papers from the Second Conference on Parallel Processing for Scientific Computing*, edited by C. W. Gear and R. C. Voigt (SIAM, Philadelphia, 1987), p. s14.
4. R. GLOWINSKI AND M. F. WHEELER, in *Proceedings, First International Conference on Domain Decomposition Methods for Partial Differential Equations*, edited by R. Glowinski *et al.* (SIAM, Philadelphia, 1987), p. 144.
5. O. B. WIDLUND, in *Proceedings, First International Conference on Domain Decomposition Methods for Partial Differential Equations*, edited by R. Glowinski *et al.* (SIAM, Philadelphia, 1987), p. 113.
6. D. E. KEYES AND W. D. GROPP, in *Proceedings, Second International Conference on Domain Decomposition Methods for Partial Differential Equations, Los Angeles*, edited by T. Chan (SIAM, Philadelphia, 1988), p. 260.
7. L. ADAMS AND R. G. VOIGT, in *Large Scale Scientific Computation*, edited by S. Parter (Academic Press, Orlando, 1984), p. 301.
8. W. D. GROPP AND D. E. KEYES, *SIAM J. Sci. Stat. Comput.* **9**, 312 (1988).
9. Y. SAAD AND M. H. SCHULTZ, *Topological Properties of Hypercubes*, Research Report YALEU/DCS/RR-389, Yale University, New Haven, 1985 (unpublished).



10. H. X. LIN AND H. J. SIPS, in *Proceedings 1986 International Conference on Parallel Processing*, edited by K. Hwang *et al.* (IEEE Comput. Soc. Washington, 1986), p. 503.
  11. A. T. PATERA, *J. Comput. Phys.* **54**, 468 (1984).
  12. J. S. PRZEMIENIECKI, *AIAA J.* **1**, 138 (1963).
  13. 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 and J. T. Oden (ASME, New York, 1989), p. 71.
  14. A. H. STROUD, AND D. SECREST, *Gaussian Quadrature Formulas* (Prentice-Hall Englewood Cliffs, NJ, 1966).
  15. F. BREZZI, *RAIRO Anal. Numer.* **8**, R2, 129 (1974).
  16. V. GIRAULT AND P. A. RAVIART, *Finite Element Approximation of the Navier–Stokes Equations* (Springer-Verlag, New York/Berlin, 1986).
  17. Y. MADAY, A. T. PATERA, AND E. M. RØNQUIST, A well-posed optimal spectral element approximation for the Stokes semi-periodic problem, *SIAM J. Numer. Anal.*, to appear.
  18. P. J. DAVIS AND P. RABINOWITZ, *Methods of Numerical Integration* (Academic Press, Orlando, FL, 1985).
  19. O. PIRONNEAU, *Numer. Math.* **38**, 309 (1982).
  20. L. HO, Y. MADAY, A. PATERA, AND E. RØNQUIST, in *Proceedings International Conference on Spectral and High Order Methods for Partial Differential Equations*, edited by C. Canuto and A. Quateroni (North-Holland, Amsterdam, 1989).
  21. G. E. KARNIADAKIS, *Int. J. Heat Mass Transfer* **31**, No. 1, 107 (1988).
  22. E. M. RØNQUIST AND A. T. PATERA, in *Proceedings Seventh GAMM Conf. on Num. Methods in Fluid Mechanics*, edited by M. Deville (Vieweg, Braunschweig, 1988), p. 318.
  23. E. M. RØNQUIST, Ph. D. thesis, Massachusetts Institute of Technology, 1988 (unpublished).
  24. S. A. ORSZAG, *J. Comput. Phys.* **37**–**70** (1980).
- 
26. Y. MADAY, D. I. MEIRON, E. M. RØNQUIST, AND A. T. PATERA, "Iterative Saddle Problem Decomposition Methods for the Steady and Unsteady Stokes Equations," MIT 1987 (unpublished).
  27. Y. MADAY AND R. MUNOZ, *J. Sci. Comput.* **3**, 4 (1988).
  28. E. M. RØNQUIST AND A. T. PATERA, *J. Sci. Comput.* **2**, 4 (1987).
  29. J. CAHOUEU AND J. P. CHABARD, in *Proceedings, Fourth International Symposium on Innovative Numerical Methods in Engineering*, edited by R. P. Shaw *et al.* (Springer-Verlag, New York/Berlin, 1986), p. 317.
  30. M. O. BRISTEAU, R. GLOWINSKI AND J. PERIAUX, *Comput. Phys. Rep.* **6**, 73 (1987).
  31. S. A. ORSZAG AND L. C. KELLS, *J. Fluid Mech.* **96**, 159 (1980).
  32. P. F. FISCHER, E. M. RØNQUIST, D. DEWEY AND A. T. PATERA, in *Proceedings, First International Conference on Domain Decomposition Methods for Partial Differential Equations*, edited by R. Glowinski *et al.* (SIAM, Philadelphia, 1987), p. 173.
  33. G. ANAGNOSTOU, P. F. FISCHER, D. DEWEY, AND A. T. PATERA, in *Recent Advances in Computational Fluid Dynamics*, edited by C. A. Brebbia *et al.* (Springer-Verlag, Berlin, 1989).
  34. P. F. FISCHER, Ph. D. thesis, Massachusetts Institute of Technology, 1989 (unpublished).
  35. J. L. GUSTAFSON, G. R. MONTRY, AND R. E. BRENNER, *SIAM J. Sci. Stat. Comput.* **9**, No. 4, 609 (1988).
  36. C. BERNARDI, N. DEBIT, AND Y. MADAY, *Note C. R. Acad. Sci. Paris* **305-I**, 353 (1987).
  37. C. MAVRIPLIS, Ph. D. thesis, Massachusetts Institute of Technology, 1989 (unpublished).
  38. G. ANAGNOSTOU, Ph. D. thesis, Massachusetts Institute of Technology, in progress (unpublished).
  39. D. DEWEY AND A. T. PATERA, "Geometry-Defining Processors for Partial Differential Equations," in *Special Purpose Computers*, edited by B. J. Alder (Academic Press, New York/London, 1988), p. 67.
  40. A. K. NOOR (Ed.), *Parallel Computations and Their Impact on Mechanics* (ASME, New York, 1987).
  41. G. FOX, M. JOHNSON, G. LYZENGA, S. OTTO, J. SALMON, AND D. WALKER, *Solving Problems on Concurrent Processors. Vol. 1. General Techniques and Regular Problems* (Prentice-Hall, Englewood Cliffs, N J, 1988).

42. G. C. FOX AND S. W. OTTO, in *Hypercube Multiprocessors 1986*, edited by M. T. Heath (SIAM, Philadelphia, 1986), p. 244.
43. B. NOUR-OMID, A. RAEFSKY, AND G. LYZENGA, "Solving Finite Element Equations on Concurrent Computers." in *Parallel Computations and Their Impact on Mechanics*, edited by A. K. Noor (ASME, New York, 1987), p. 209.
44. P. F. FISCHER, E. M. RØNQUIST, AND A. T. PATERA, "Parallel Spectral Element Methods for Viscous Flow," in *Parallel Supercomputing: Methods, Algorithms, and Applications*, edited by G. F. Carey (Wiley, Chichester, 1989), p. 223.
45. P. F. FISCHER AND A. T. PATERA, "Parallel Spectral Element Methods for the Incompressible Navier–Stokes Equations," in *Solutions of Super Large Problems in Computational Mechanics*, edited by J. H. Kane and A. D. Carlson (Plenum, New York, 1989).
46. P. F. FISCHER, in *Proceedings, International Conference on Spectral and High Order Methods for Partial Differential Equations*, edited by C. Canuto and A. Quateroni (North-Holland, Amsterdam, 1989).