INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids **31**: 37–52 (1999)

PARALLEL FINITE ELEMENT SOLUTION OF THREE-DIMENSIONAL RAYLEIGH–BÉNARD–MARANGONI FLOWS

G.F. CAREY*, R. McLAY, G. BICKEN, B. BARTH, S. SWIFT AND A. ARDELEA
The University of Texas at Austin, Austin, TX, USA

SUMMARY

A domain decomposition strategy and a parallel gradient-type iterative solution scheme have been developed and implemented for the computation of complex three-dimensional viscous flow problems involving heat transfer and surface tension effects. Special attention has been paid to the kernels for the computationally intensive matrix–vector products and dot products, to memory management, and to overlapping communication and computation. Details of these implementation issues are described together with associated performance and scalability studies. Representative Rayleigh–Bénard and microgravity Marangoni flow calculations and performance results on the Cray T3D and T3E are presented. Performance studies have been recently carried out and sustained rates above 50 gigaflops and 100 gigaflops have been achieved on the 512-node T3E-600 and 1024-node T3E-900 configurations respectively. The work is currently being extended to tightly-coupled parallel 'Beowulf-type' PC clusters and some preliminary performance results on this platform are presented. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: Rayleigh–Bénard–Marangoni flows; parallel finite element solution; three-dimensional

1. INTRODUCTION

Coupled three-dimensional viscous flow and heat transfer computations are of great interest in studying many manufacturing processes and natural phenomena [1]. Familiar examples include hot forming and welding in manufacturing and the motion of the earth's molten core. Typically, buoyancy is a dominant component in driving this type of flow and there have been many studies of buoyancy-driven flows, such as the Rayleigh–Bénard problem [2,3]. These have been motivated in part by the early experimental studies of Bénard on the cell structures observed in thin liquid layers heated from below. While the essential physics of this particular problem was misunderstood for decades, it is now recognized that surface tension effects associated with temperature gradients on the free surface actually provide the dominant force driving the flow [4–11]. Accordingly, these flows are now termed Rayleigh–Bénard–Marangoni (R–B–M) problems and their study is particularly important for fluids in microgravity environments, such as the space station, and for terrestrial applications involving thin fluid layers, such as industrial coating processes [12,13]. In both of these scenarios, the surface tension (thermocapillary effect) is dominant and strongly influences the structure of the

* Correspondence to: College of Engineering, WRW 305, The University of Texas at Austin, Austin, TX 78712, USA.

flow. For example, in recent terrestrial flow experiments on thin liquid layers, new non-linear instabilities that lead to significant surface deformation and the formation of 'dry spots' have been observed (M.F. Schatz, S.J. Vanhook, W.D. McCormick, J.B. Swift and H.L. Swinney, 'Instability and transition to disorder in surface-tension driven Bénard convection', submitted).

The main objective in the present work is to develop effective parallel algorithms and a distributed parallel implementation capable of high-resolution three-dimensional coupled flow and heat transfer computations, including surface tension effects. This will permit fundamental phenomenological flow studies at the grid resolution necessary to represent the fine-scale surface-driven phenomena to be made and the associated non-linear free surface behavior to be studied. The discretization involves three-dimensional isoparametric 'quadrilateral brick' finite elements with triquadratic velocity, trilinear pressure and triquadratic temperature approximation on the elements. A non-overlapping domain decomposition of the grid is generated with processor interfaces coincident with a subset of element surfaces. This implies that nodes on the subdomain interfaces are shared by adjacent processors [14,15].

In previous work by the authors [16], the main algorithm employed a decoupled repeated block iteration between the viscous flow and heat transfer computations for stationary or transient flow and transport computations. Each of these main simulation steps was solved by iterative linearization and the linear subsystems in turn were solved by biconjugate gradient iteration with diagonal preconditioning. The new work is based on a fully-coupled formulation for velocity, pressure and temperature. This implies that a larger fully-coupled linear system must be solved at each non-linear iteration within a given time step. The basic approach involves parallelization of the computationally intensive matrix–vector products and dot products of the generalized gradient solvers across a partition to subdomains. The subdomain contributions are not assembled and instead the calculations are carried out using an element-by-element approach as before, but this now involves a larger coupled element matrix. Both the decoupled and fully-coupled schemes will be considered here. Special care is taken to ensure efficient processor computation of the computationally intensive matrix–vector product kernels and to overlap communication and computation for elements adjacent to the processor subdomain boundaries. Performance studies of the decoupled scheme on 512 nodes of a Cray T3D at NASA Goddard were verified in March 1997 at a sustained rate in excess of 16 gigaflops. More recent performance studies on the Cray T3E-600 and T3E-900 have run at sustained rates above 50 gigaflops and 100 gigaflops on 512 and 1024 processors respectively.

In the next two sections, the problem formulation and then the BCG kernel are briefly summarized. Then in Section 4, basic domain decomposition and element-by-element strategy within subdomains are given, and the main ideas of the parallel implementation are presented. Then, scaled speed-up studies for representative Rayleigh–Bénard–Marangoni calculations are provided. Finally, in the concluding remarks, work in progress on a very large distributed system for complex applications is indicated.

2. FORMULATION

The transient flow of a viscous incompressible fluid as described by the Navier–Stokes equations coupled to the transport of heat by conduction and convection in the fluid is considered. Buoyancy is included by means of the Boussinesq approximation as a temperature-dependent body force term in the momentum equations and the velocity field enters the convective term in the heat transfer (energy) equation. The effect of thermocapillary surface tension enters as an applied shear stress which is also dependent on the surface temperature gradient. The Navier–Stokes equations for viscous flow of an incompressible fluid may be written

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t}+\boldsymbol{u}\cdot\nabla\boldsymbol{u}\right)+\nabla\cdot\boldsymbol{\tau}=\boldsymbol{f}+\beta(T-T^*)\boldsymbol{g}, \tag{1}$$

$$\nabla\cdot\boldsymbol{u}=0, \tag{2}$$

in flow domain Ω, where $\boldsymbol{u}$ is the velocity field, $\boldsymbol{\tau}$ is the stress tensor and is specified by Stokes hypothesis for a Newtonian fluid, $\boldsymbol{f}$ is an applied body force, $\boldsymbol{g}$ is the gravity vector, T^* is the reference external temperature, T is the fluid temperature and β is the thermal coefficient. At the solid wall boundaries, the no-slip condition applies so $\boldsymbol{u}=\boldsymbol{u}_{\mathrm{w}}$, where $\boldsymbol{u}_{\mathrm{w}}$, is the specified wall boundary velocity.

At the free surface, a shear stress due to thermocapillary surface tension acts. For example, on a horizontal free surface, the tangential shear stress component τ_{zx} is given by

$$\tau_{zx}=\frac{\partial\gamma}{\partial x}=\frac{\partial\gamma}{\partial T}\frac{\partial T}{\partial x}, \tag{3}$$

with a similar expression for τ_{zy}, where $\gamma(T)$ is the surface tension and T is temperature. The heat equation is

$$\rho c_p\left(\frac{\partial T}{\partial t}+\boldsymbol{u}\cdot\nabla T\right)-\nabla\cdot(k\nabla T)=Q, \tag{4}$$

where k is the thermal conductivity of the fluid, ρ is the density, c_p is the heat capacity and Q is a heat source term. Temperature, flux or mixed thermal boundary conditions may be applied. For example, in the R–B–M test problem later, temperature is specified as $T=T_{\mathrm{b}}$ on the base, zero normal flux $k(\partial T/\partial n)=0$ on the side walls, and mixed conditions $k(\partial T/\partial n)=\alpha(T-T^*)$ (Robin) on the free surface, where T^* is the exterior temperature and α is the heat transfer coefficient for the medium. Then (1), (2) and (4) constitute a coupled system to be solved for velocity, pressure and temperature.

Introducing test functions $\boldsymbol{v}$ and $\boldsymbol{q}$ in a weighted-residual statement for (1) and (2), integrating by parts using (3) and introducing the Stokes hypothesis, we obtain the weak variational statement: find the pair $(\boldsymbol{u},p)\in V\times Q$ with $\boldsymbol{u}=\boldsymbol{u}_{\mathrm{w}}$ at the wall boundaries and such that

$$\int_\Omega\frac{\partial\boldsymbol{u}}{\partial t}\cdot\boldsymbol{v}\,\mathrm{d}x+\int_\Omega\boldsymbol{u}\cdot\nabla\boldsymbol{u}\cdot\boldsymbol{v}\,\mathrm{d}x+\int_\Omega(v\nabla\boldsymbol{u}:\,\nabla\boldsymbol{v}-p\nabla\cdot\boldsymbol{v})\,\mathrm{d}x+\int_\Gamma\frac{\partial\gamma}{\partial T}\nabla T\cdot\boldsymbol{v}\,\mathrm{d}s$$
$$=\int_\Omega(\boldsymbol{f}\cdot\boldsymbol{v}+\rho\beta(T-T^*)\boldsymbol{g})\cdot\boldsymbol{v}\,\mathrm{d}x \tag{5}$$

and

$$\int_\Omega\nabla\cdot\boldsymbol{u}q\,\mathrm{d}x=0 \tag{6}$$

hold for all admissible $(\boldsymbol{v},q)$. Similarly, the weighted integral for (4) yields: find $T\in W$ satisfying any specified (essential) temperature boundary conditions and such that

$$\int_\Omega\left\{\rho c_p\left(\frac{\partial T}{\partial t}w+\boldsymbol{u}\cdot\nabla Tw\right)+k\nabla T\cdot\nabla w\right\}\mathrm{d}x-\int_\Gamma\alpha(T-T^*)\omega\,\mathrm{d}s=\int_\Omega Qw\,\mathrm{d}x \tag{7}$$

for all admissible test functions w with $w=0$ on those parts of the boundary where T is specified.

The finite element formulation for (5)–(7) follows immediately on introducing the approximation subspaces V_h, Q_h, W_h for V, Q and W respectively to obtain: find $(\boldsymbol{u}_h, p_h, T_h)$ with $\boldsymbol{u}_h$ and T_h satisfying the essential boundary conditions and initial conditions and such that

$$\int_\Omega \frac{\partial \boldsymbol{u}_h}{\partial t}\cdot v_h \,\mathrm{d}x + \int_\Omega \boldsymbol{u}_h\cdot\nabla\boldsymbol{u}_h\cdot\boldsymbol{v}_h \,\mathrm{d}x + \int_\Omega (v\nabla\boldsymbol{u}_h\colon \nabla\boldsymbol{v}_h - p\nabla\cdot\boldsymbol{v}_h)\,\mathrm{d}x + \int_\Gamma \frac{\partial\gamma}{\partial T}\nabla T_h\cdot\boldsymbol{v}_h \,\mathrm{d}s$$

$$= \int_\Omega (\boldsymbol{f}\cdot\boldsymbol{v}_h + \beta(T_h - T^*)\boldsymbol{g})\cdot\boldsymbol{v}_h \,\mathrm{d}x, \tag{8}$$

$$\int_\Omega \nabla\cdot\boldsymbol{u}_h q_h \,\mathrm{d}x = 0 \tag{9}$$

and

$$\int_\Omega \left\{\rho c_p\left(\frac{\partial T_h}{\partial t} w_h + \boldsymbol{u}_h\cdot\nabla T_h w_h\right) + k\nabla T_h\cdot\nabla w_h\right\}\mathrm{d}x - \int_\Gamma \alpha(T_h - T^*)\omega \,\mathrm{d}s = \int_\Omega Q w_h \,\mathrm{d}x \tag{10}$$

hold for all admissible $\boldsymbol{v}_h$, q_h and w_h, with $\boldsymbol{v}_h = \boldsymbol{0}$ and $w_h = 0$ on those parts of the boundary where $\boldsymbol{u}$ and T are specified respectively. Now (8)–(10), define the coupled finite element problem to be solved.

Introducing the discretization of elements and the finite element basis functions, the respective velocity, pressure and temperature expansions are

$$u_h^k(\boldsymbol{x}, t) = \sum_{i=1}^{n} u_j^k(t)\phi_j(\boldsymbol{x}),$$

$$T_h(\boldsymbol{x}, t) = \sum_{j=1}^{n} T_j(t)\phi_j(\boldsymbol{x}), \tag{11}$$

$$p_h(\boldsymbol{x}, t) = \sum_{j=1}^{n} p_j(t)\phi_j(\boldsymbol{x}),$$

where k is the velocity component index ($k = 1, 2, 3$ for three-dimensional flow) and u_j^k, p_j, T_j are the nodal values. Substituting into (8)–(10) and writing in matrix form we have a semi-discrete finite element system of the form

$$\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{U}}{\mathrm{d}t} + \boldsymbol{s}(\boldsymbol{U}) + v\boldsymbol{A}\boldsymbol{U} + \boldsymbol{B}\boldsymbol{p} = \boldsymbol{F} + \boldsymbol{b}(\boldsymbol{T}), \tag{12}$$

$$\boldsymbol{B}^T\boldsymbol{U} = \boldsymbol{0}, \tag{13}$$

$$\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}t} + \boldsymbol{K}\boldsymbol{T} + \boldsymbol{C}\boldsymbol{T} = \boldsymbol{Q}. \tag{14}$$

A variety of integration schemes are applicable to advance the solution from a specified initial state $\boldsymbol{U}(0)$, $\boldsymbol{T}(0)$. In the present work, a standard θ method is used, with $0 \le \theta \le 1$, so that at time step (t_n, t_{n+1}):

$$\frac{\boldsymbol{M}(\boldsymbol{U}^{n+1} - \boldsymbol{U}^n)}{\Delta t} + \theta[v\boldsymbol{A}\boldsymbol{U}^{n+1} + \boldsymbol{s}(\boldsymbol{U}^{n+1}) + \boldsymbol{B}\boldsymbol{p}^{n+1}] + (1-\theta)[v\boldsymbol{A}\boldsymbol{U}^n + \boldsymbol{s}(\boldsymbol{U}^n) + \boldsymbol{B}\boldsymbol{p}^n]$$

$$= \theta\boldsymbol{G}^{n+1} + (1-\theta)\boldsymbol{G}^n, \tag{15}$$

$$\boldsymbol{B}^T\boldsymbol{U}^{n+1} = \boldsymbol{0}, \tag{16}$$

$$\frac{\boldsymbol{M}(\boldsymbol{T}^{n+1}-\boldsymbol{T}^{n})}{\Delta t}+\theta[\boldsymbol{K}\boldsymbol{T}^{n+1}+\boldsymbol{C}^{n+1}\boldsymbol{T}^{n+1}]+(1-\theta)[\boldsymbol{K}\boldsymbol{T}^{n}+\boldsymbol{C}^{n}\boldsymbol{T}^{n}]=\theta\boldsymbol{Q}^{n+1}+(1-\theta)\boldsymbol{Q}^{n}, \tag{17}$$

where $\boldsymbol{s}$ is evaluated using the current velocity iterate and $\boldsymbol{G}=\boldsymbol{F}+\boldsymbol{b}(\boldsymbol{T})$. Here $\theta=0, 1/2, 1$ correspond respectively to the familiar forward, mid-step and backward Euler integrators. In the results shown later we employ the mid-step $\theta=1/2$ scheme. Hence, a fully-coupled linear system in parallel must be solved for each iterate within a given time step.

The decoupled transient scheme proceeds by 'lagging' the temperature in the momentum equations (15), (16) to enable the decoupling within each time step. Then, using the resulting velocity field, $\boldsymbol{C}$ is computed and (17) is solved for the temperature iterate. These two solve steps can be repeated until the convergence criterion is met for the time step. The procedure is then repeated for the next time step. In the next section, the BCG algorithm for the kernel system solves is given.

3. BCG KERNEL

It can be seen that both the coupled and decoupled algorithms require repeated system solves within each time step. These systems are sparse and non-symmetric, but the asymmetry in the R–B–M microgravity problem is not strong. This implies that iterative methods with Jacobi preconditioning should be quite effective in solving each linear subsystem. In the present work, the respective systems are solved in parallel over subdomains using biconjugate gradient iteration (BCG) with diagonal preconditioning.

Consider such a non-symmetric system

$$\boldsymbol{A}\boldsymbol{u}=\boldsymbol{f}. \tag{18}$$

Then the basic BCG scheme applied here proceeds as follows.

Algorithm

```
1. Initialization
    Set preconditioner $\underline{\boldsymbol{Q}}$
    Compute $\|\boldsymbol{f}\|=\sqrt{(\boldsymbol{f}^T\boldsymbol{f})}$
    Set iteration counter $n=0$
Restart:
    Compute initial residual $\boldsymbol{r}_n=\boldsymbol{f}-\boldsymbol{A}\boldsymbol{u}_n$
    Set $\bar{\boldsymbol{r}}_n=\boldsymbol{r}_n$
    Apply preconditioning by solving $\boldsymbol{Q}\boldsymbol{z}_n=r_n$ and $\boldsymbol{Q}\bar{\boldsymbol{z}}_n=\bar{\boldsymbol{r}}_n$ for $\boldsymbol{z}_n$ and $\bar{\boldsymbol{z}}_n$
    Compute $\|\boldsymbol{r}_n\|=\sqrt{\boldsymbol{z}_n^T\boldsymbol{r}_n}$ and check for convergence
    Compute $\rho_n=\boldsymbol{z}_n^T\bar{\boldsymbol{r}}_n$; set $\boldsymbol{p}_{n+1}=\boldsymbol{z}_n$ and $\bar{\boldsymbol{p}}_{n+1}=\bar{\boldsymbol{z}}_n$
    Change $\|\boldsymbol{f}\|$ to be $\|\boldsymbol{r}_n\|$ if $\sqrt{\boldsymbol{f}^T\boldsymbol{f}}$ is small
    Set $n=n+1$ to update counter inside Restart

2. Iteration:
    For $k=n, n+1, \ldots iterMax$
        set $n=n+1$
        Compute $\boldsymbol{w}_k=\boldsymbol{A}\boldsymbol{p}_k$, $\bar{\boldsymbol{w}}_k=\boldsymbol{A}^T\bar{\boldsymbol{p}}_k$
        Compute $\sigma_k=\bar{\boldsymbol{p}}_k^T\boldsymbol{w}_k$ and Restart if $|\sigma_k|$ is small
        Set $\alpha_k=\rho_{k-1}/\sigma_k$
```

```
Update $\boldsymbol{u}_k = \boldsymbol{u}_{k-1} + \alpha_k \boldsymbol{p}_k$
Update $\boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \alpha_k \boldsymbol{w}_k$
Apply preconditioning by solving $\boldsymbol{Q}\boldsymbol{z}_k = \boldsymbol{r}_k$ for $\boldsymbol{z}_k$
Compute $\|\boldsymbol{r}_k\| = \sqrt{(\boldsymbol{z}_k^T \boldsymbol{r}_k)}$
Stopping test, if converged then break loop
Update $\bar{\boldsymbol{r}}_k = \bar{\boldsymbol{r}}_{k-1} - \alpha_k \bar{\boldsymbol{w}}_k$
Apply preconditioning by solving $\boldsymbol{Q}\bar{\boldsymbol{z}}_k = \bar{\boldsymbol{r}}_k$ for $\bar{\boldsymbol{z}}_k$
Compute $\rho_k = \boldsymbol{z}_k^T \bar{\boldsymbol{r}}_k$
Restart if $|\rho_{k-1}|$ is small
Set $\beta_k = \rho_k / \rho_{k-1}$
Compute $\boldsymbol{p}_{k+1} = \beta_k \boldsymbol{p}_k + \boldsymbol{z}_k$, $\bar{\boldsymbol{p}}_{k+1} = \beta_k \bar{\boldsymbol{p}}_k + \bar{\boldsymbol{z}}_k$
```

Note that all the dot products when done in parallel require a *mask* that is described in the next section. Also, when either $|\sigma_k|$ or $|\rho_{k-1}|$ is small the program returns to the `Restart` label, recomputes the residual and continues from there as before. The reason that counter n was added is that it prevents the code from looping forever.

The main computational steps in this solution algorithm are matrix–vector products, transpose matrix–vector products and vector dot products. Since the matrices are sparse (because of the local support of the element bases) fast parallel sparse matrix–vector product (MATVEC) routines are required, and also a fast parallel dot product routine. Frequently, in the parallel iterative literature, the MATVEC is left to the user to provide but this is the 'heart' of the calculation and must be handled efficiently if a fast scalable parallel solver is to be designed. The parallel MATVEC and the dot product routines are elaborated on in the next section.

4. DOMAIN DECOMPOSITION

This study uses a non-overlapping decomposition by elements. This implies that the processor interfaces coincide with a subset of element faces and the nodes on those faces are shared by adjacent processors. This is natural in a finite element framework since it implies that the element calculations can be parallelized easily over the processor partition and also lends itself to element-by-element solution strategies by either iterative schemes [17,18] or frontal elimination within each subdomain (e.g. see A. Bose, V.F. de Almeida and G.F. Carey, 'A class of multiple front algorithms on subdomains', in preparation, 1999) for a parallel element-by-element multi-front scheme).

A subdomain element-by-element approach (in which each subdomain contains a sufficient number of elements) provides an efficient parallel strategy. Here the dense matrix operations at the element level can be exploited in the intensive computational kernels and the communication at the processor interface nodes can be overlapped with computation in the interior of each subdomain.

Consider a typical interior subdomain Ω_s containing elements w_e^s, $e = 1, 2, \ldots, E_s$ and let B_j^s, $j = 1, 2, \ldots, J$ denote those border elements of Ω_s that are adjacent to the subdomain boundary $\partial\Omega_s$. Let I_k^s, $k = 1, 2, \ldots, K$ denote the remaining elements interior to the subdomain. Computations involving this interior subset are local to the processor. This implies that communication requests can be initiated for the subdomain border element calculations while computation begins on the interior subset. Then the border element computations can be completed. Provided there are sufficient elements in the interior subset this strategy will permit complete communication overlap.

For simplicity, consider an interior cube subdomain. There are six surface subdomain neighbors, 12 edge subdomain neighbors (that are not also surface neighbors) and eight further vertex neighbors. Hence, there are 26 subdomain neighbors involved in message passing to or from any given interior subdomain cube. For subdomain interface nodes of the border elements, the face-interior nodes are duplicated twice, once on each neighboring processor, edge-interior nodes are duplicated four times and corner nodes are duplicated eight times. This has some bearing on the way we compute the global dot products in the algorithm. More specifically, a 'masked' dot product is computed with masking weight, ω_i for node i. Here ω_i is the reciprocal of the number of subdomains sharing node i. e.g. $\omega_i = 1$ for subdomain interior node i, $\omega_i = \frac{1}{2}$ for face-interior node, etc.

The element matrix calculations are trivially parallelized over the processor subdomains and the subdomain matrix–vector product can be conceptually separated as follows:

```
For elements $e$ in the border:
    Compute EBE matrix–vector product.
    Sum element result into nodal result vector.
Extract interface nodal MVP results to send buffers and transfer to neighbor processors.
For elements $e$ in the interior:
    Compute EBE matrix–vector product.
    Sum element result into nodal result vector.
Sum receive buffer data for neighboring processors into nodal MVP result vector.
```

5. COMMUNICATION STRATEGY

A combination of MPI and SHMEM is used to handle the communications for the T3E. SHMEM is used where speed is important, but SHMEM is less portable. For example, global operations with SHMEM require a buffer at the same address on all processors. Hence, for operations like dot products, only one global location is needed and SHMEM is used. Operations that require broadcasting buffers of unknown size are more difficult and MPI is used here.

The basic approach is as follows. First, the input file needs to be read by the processors. Since the bandwidth speed of the network is much faster than reading from disk, it is better to make a single read and then broadcast rather than have each processor read the input file from disk. Accordingly, in the present approach, processor zero reads the input file a line at a time and simultaneously broadcasts each line to all other processors. This step is implemented using MPI.

When the global mesh is partitioned across processors the 'send list' of nodes is also set up. These are nodes on the subdomain boundaries and hence are shared by neighboring processors. One approach would be to set up a translation table to relate global and local node numbers for these shared nodes on neighboring processors. Instead this is done implicitly by setting up local and global numbering systems as described in Section 4. Then each processor has a send list that puts values on the send buffer and reads from the receive buffer in the same order. For example, if a value for the third node (in the buffer) sent between neighboring processors p and q corresponds to local node 12 on processor p and local node 124 on processor q then this local identification between processors is maintained. This implicit equivalence of local node values avoids sending global node numbers but complicates forming send lists. However, this step is required only once for the static processor partitions used here. Since the implementation is restricted to structured meshes, the send list can be constructed on each processor by tracing each face in the same direction.

Communication is required in the solution algorithm for global operations, such as dot products, other global sums and maximum value calculations. For example, consider a typical global dot product computation in the gradient iterative solution algorithm (Section 3). We need to make local computations in parallel, communicate these results and accumulate the global sum. Since nodes on subdomain boundaries are shared, there are duplicated values and the local dot products have to be weighted appropriately. These weighted or 'masked' local dot products are computed in parallel as described previously (Section 4) and the local scale-up results are accumulated to the correct global sum. This latter step is carried out using the routine: shmem_double_sum_to_all.

Neighbor-to-neighbor communications are also required, most notably in the matrix–vector product computations. Specifically, the result of a global matrix–vector product is a global vector. Nodes on a subdomain boundary are shared by adjacent processors and the local contributions to these nodes must be accumulated and shared as described in Section 4. Accordingly, we first initiate computations on the subdomain 'boundary strips' and communicate these values to the neighboring processors using SHMEM. During this communication step, the local matrix–vector contributions on the remaining interior elements of each subdomain are carried out. A barrier is inserted at the end of this interior computation so that calculation can not proceed until remaining communication (if any) is completed. The communicated border values are then accumulated into the local vectors. This results in a correct local processor extraction of the global vector (with duplication of values for nodes shared by neighboring processors). We see only a small (approx. 2%) improvement by overlapping communication with calculation on the T3E but since we use asynchronous communication it is just as convenient to overlap as not.

6. SOFTWARE DESIGN

The program is designed for fast scalable parallel computation of steady and transient solutions to coupled incompressible viscous flow with heat and mass transfer. Parallel efficiency is achieved by careful implementation of MPI and customized communication software (such as SHMEM on the T3E) using a domain decomposition strategy. Since most of the computation time is taken solving the associated large sparse systems, a major effort was devoted to optimizing the solver. Within each time step or Newton iteration, we need to solve sparse linear non-symmetric systems. Biconjugate gradient (BCG) and conjugate gradient schemes are implemented in parallel over the partition to subdomains, with an element-by-element data structure for the subdomain computations. Since there is need to compute matrix–vector (MATVEC) products repeatedly, special consideration is given to this component [19]. For the triquadratic velocity, trilinear pressure, triquadratic temperature bases, the element matrix for the coupled formulation is of size greater than 100×100 and relatively dense (few zero entries). This large element matrix size implies that the subdomain matrix–vector products can be computed efficiently element-by-element. More specifically, assembly coded BLAS matrix–vector product routines (SGEMV or DGEMV) are used for element matrix–vector products and the resulting subdomain vectors are assembled.

MGFLO is designed for unstructured mesh finite element simulations on irregular partitions, such as those generated by Chaco [20] or Metis [21]. This flexibility is facilitated by means of lists in C. Each processor has a set of elements that is further broken down to (1) 'frame' elements, which have a face, edge or corner shared with elements residing on another processor and (2) the remaining 'interior' elements on the processor. Accordingly, three lists of pointers are used—one list is to all elements of the processor and the other two to the respective frame and interior elements. Note that elements are only stored once but there are two pointers to each element. To process elements, simply loop over the appropriate list (all elements, frame elements or interior elements). We remark that essential boundary conditions are stored nodally, whereas flux data is stored by elements. Communication between processors is implemented using a list of nodes that are shared by each processor.

In the numerical studies presented later, only structured subdomains are considered, and a simple mesh generator that can be partitioned regularly is used. That is, the mesh routine knows that it has a structured mesh to create and partition. It also creates the three element lists and the list of nodes that need to be exchanged between processors. (For more general partitionings and grids one must set up the communication node lists accordingly).

The input files are described by a simple mini-language so that the code acts on them as an interpreter. The program has no ordered script of tasks. After reading in the data set, it performs the steps in the order that the user requests them. This provides great flexibility in how the code gets used. Simple things like checking the mesh without running the solver are easy: just do not request the solver to be run. Obviously, this also requires a more sophisticated user because the commands must be placed in an appropriate order. (Requesting the code to solve a problem before the mesh and material properties are given is possible but wrong.)

The operational statements of the code are written in C but the actual programming style and implementation is carried out using a higher-level 'literate' programming tool called '`noweb`' [22,23]. This has two main features: (1) it permits direct in-line documentation of the code using LATEX so that to a significant degree, the code can be internally documented (with glossary, index and cross-referencing) and this encourages good programming practice; (2) it encourages a structured programming style that has some of the attributes of object-oriented programming independent of the operational language (C or FORTRAN, etc.). The basic approach to coding can then be described as follows.

Each section of code can have a documentation section (a text chunk) which is written in LATEX followed by a computer code section (a code chunk). The pairs of chunks can be combined in an order to suit a human reader. This combined file can be processed two ways. The first way extracts the computer code in an order suitable for a compiler. This is called tangling. The second way (called weaving) produces a human readable document.

The layout for a `noweb` program is to specify a text chunk by starting a line with @ symbol followed by a new line. The code chunk begins with a $\langle\langle$*chunk name*$\rangle\rangle =$ on a line by itself. Chunks are terminated implicitly by the beginning of a new chunk or the end of the file. Code chunks can refer to other code chunks. This nesting of code chunks can be as deep as necessary.

A simple example may clarify how this works. By default, all `noweb` files start expanding from the $\langle\langle * \rangle\rangle$ chunk. So a simple C program that calls a routine, `vecAXPBY` that computes $y = \alpha x + \beta y$ is given below.

```
⟨⟨*⟩⟩=
⟨⟨Headers⟩⟩
⟨⟨Vector α x+β y⟩⟩
void main ( )
```

```
{
  ⟨⟨Initialize arrays⟩⟩
  vecAXPBY(n, alpha, x, beta, y);
}
@
⟨⟨Initialize arrays⟩⟩=
int n=10;
int j;
float alpha=1.23, beta=3.14;
float x[10], y[10];
for (j=0; j<n; j++) {
  x[j]=2.0*j; y[j]=3.0*j;
  }
@
⟨⟨Vector αx=βy⟩⟩=
void vecAXPBY(int n, float alpha, float*x, float beta, float*y)
{
  int ja;

  ⟨⟨Compute αx=βy⟩⟩
}
@
⟨⟨Compute αx=βy⟩⟩=
#if (defined(_CRAYMPP))
  int Iskip=1;
  SAXPBY(&n, &alpha, x, &Iskip, &beta, y, &Iskip);
#else
  for(ja=0; ja<n; ja++)
    y[ja]=alpha*x[ja]=beta*y[ja];
#endif

@
⟨⟨Headers⟩⟩=
#include ⟨stdio.h⟩
```

would expand to

```
#include ⟨stdio.h⟩
void vecAXPBY(int n, float alpha, float*x, float beta, float*y)
{
  int ja;

   #if (defined(_CRAYMPP))
    int Iskip=1;
    SAXPBY(&n, &alpha, x, &Iskip, &beta, y, &Iskip);
   #else
    for(ja=0; ja<n; ja++)
      y[ja]=alpha*x[ja]=beta*y[ja];
  #endif
}

void main( )
{
  int n=10;
  int j;
  float alpha=1.23, beta=3.14;
  float x[10], y[10];
  for (j=0; j<n; j++) {
    x[j]=2.0*j; y[j]=3.0*j;
  }
  vecAXPBY(n, alpha, x, beta, y);
}
```

This example shows how chunks like ⟨⟨`Initialize arrays`⟩⟩ and ⟨⟨`Compute` $\alpha x+\beta y$⟩⟩ are expanded in to code. The chunk ⟨⟨`Compute` $\alpha x + \beta y$⟩⟩ also shows how `noweb` can hide conditional compilation statements from the main algorithm. Furthermore, it is in large programs developed by one or more analysts that the benefits of tools like `noweb` become evident.

Combining the documentation with the code is certainly an important aspect. This means that the documentation and the code are less likely to be out of date with each other. Also, having access to features of LATEX, such as the mathematical equations and ability to add figures, or formatted tables greatly improve the documentation. Such programs now have a table of contents and an index that facilitate understanding and debugging.

The ability to collapse entire algorithms to a single page listing key steps that include mathematical expressions is of great value and makes the code easily understood by software team developers. For example, the major loop over Gauss points for generating the matrices can be expressed succinctly as:

```
/* Loop over Gauss points */
  for (igp=0; igp<gp3d; igp++)
    {
      ⟨Compute $x, y, z, \frac{\partial x}{\partial \xi}, \frac{\partial x}{\partial \eta}, \frac{\partial x}{\partial \zeta}$ at gp⟩
      ⟨Compute the Jacobian at gp⟩
      ⟨Compute $\frac{\partial \Phi}{\partial x}, \frac{\partial \Phi}{\partial y}, \frac{\partial \Phi}{\partial z}$ at gp⟩
      ⟨Compute Soln and derivs $\frac{\partial T}{\partial x}, \frac{\partial U}{\partial x}, \frac{\partial V}{\partial x}$ at gp⟩
      ⟨Compute old Velocity and Temperature at gp⟩
      ⟨Compute Material Properties at gp⟩
      ⟨Compute Mass Matrix if TRANSIENT⟩
      ⟨Compute Steady State matrices and rhs⟩
    }
```

One might argue that subroutines provide a similar benefit, since routines could be called instead of chunks. In theory this is true, but very long descriptive routine names are almost never used, and it is not possible to use mathematical symbols for subroutine names. More importantly, any given routine will also be made of up of steps that can be broken up into chunks. Replacing all chunks by subroutines would also add the overhead involved in calling the routine. Also local variables that can be shared directly between chunks could not be shared directly between subroutines unless they are passed through on the argument lists or declared as global variables. This could force very long argument lists between routines, which would hinder understanding the code. We remark that the use of these techniques imposes no execution overhead and can be added to any computer language. It is clear that all these ideas can be used in straight Fortran or C code, but combining these languages with a tool like `noweb` makes all programming much easier.

7. BEOWULF-TYPE CLUSTERS

A Beowulf-type cluster comprised of 16 Intel Pentium II processors has been recently assembled by the authors for the investigation of parallel computing on workstation clusters. Each node is equipped with a single Pentium II processor with 128 MB of RAM. The processors are interconnected with a cross-bar hub running 100 MBit Fast Ethernet (see Figure 1). With the goal of optimizing our MPI-based flow analysis software to run efficiently in parallel on this cluster, we are currently studying communication performance. Owing to the fact that these workstations are part of a cluster built from commodity parts, we do not have a customized communications controller and high-speed proprietary network, such as those available on modern supercomputers like the T3E. However, a large cluster with myrianet switching has been assembled at TICAM and we are beginning to carry out performance benchmark tests. It is, therefore, necessary to develop an understanding of the network available to our cluster in order to determine the best way to optimize communication performance. To this end, we have also developed and are continuing to develop an MPI-based communication performance test code. The communications test code starts up MPI and sends a series of point-to-point messages between all processors. The messages sent in this study are of a fixed size supplied by the user. The code then performs a number of realizations of these point-to-point messages, from which we can compile statistical information about the underlying network.

Concurrently with this network performance study, the flow analysis program with MPI has been implemented for this architecture and preliminary performance studies have been carried

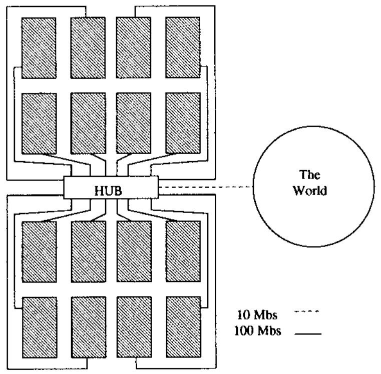

Figure 1. Beowulf cluster network topology.

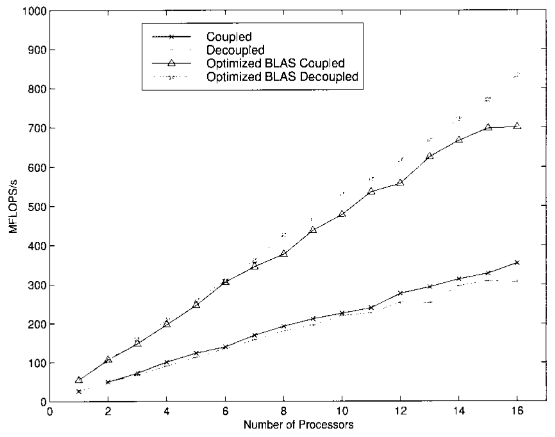

Figure 2. Parallel speed-up scaling on the Beowulf cluster.

out. Figure 2 shows the results of a scaled speed-up study on the Beowulf cluster. Four cases were examined in this study: coupled and decoupled flow and heat transfer with non-optimized FORTRAN-coded BLAS and coupled and decoupled flow and heat transfer with Pentium II optimized assembly coded BLAS. In both cases, the results show almost linear scaling for both coupled and decoupled flow regimes, with the variation from linearity due to system software running in the background.

Plate 1 shows results for natural convection in the unit cube run with the code on eight nodes of this cluster. In this study with non-dimensional variables, the bottom and two of the side walls are insulated, the near wall is held at $T=1$, and the far wall is held at $T=0$. All of these walls are no-slip. The top is a free surface with thermocapillary surface tension and a mixed heat transfer boundary condition ($h=1$, $T_{\text{ref}}=1$, Rayleigh number $=2200$). The simulation was made using a $10\times 10\times 10$ mesh in a $2\times 2\times 2$ Cartesian partition to subdomains (each subdomain is then $5\times 5\times 5$) with parallel block-Jacobi preconditioning. The tolerance for the non-linear Newton iteration was 10^{-11} with a maximum of 50 Newton iterations per non-linear solve. The maximum number of conjugate gradient iterations per

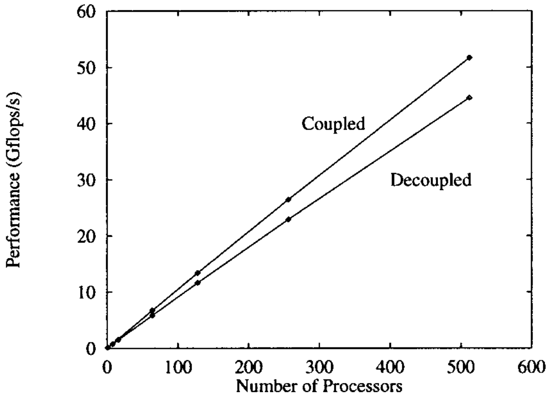

Figure 3. Parallel speed-up scaling on the T3E-600.

linear subsystem solve is 3000 with a relative residual tolerance of 10^{-6} for convergence. Shown in the figure are 'stream-ribbons' to illustrate the computed flow field behavior and base and side-wall temperature contours. The computation required 8 h for steady state solution.

8. T3E THERMOCAPILLARY STUDIES

A series of parallel performance studies for the Cray T3E-600 were first conducted for a representative R–B–M problem using both the fully-coupled and decoupled formulations. Parallel speed-up scaling showing sustained gigaflops versus number of processors is shown in Figure 3. A similar study was undertaken later for scaling through 1024 processors on the T3E-900, as shown in Figure 4, and delivers a maximum of approximately 118 gigaflops. Both figures are obviously for scaled problem size (the subgrid size per processor is fixed and sufficiently large).

The results of two case studies carried out on the Cray T3E are presented. In the first problem, the fluid flow in an $L \times L \times L/4$ domain is considered, driven by an axisymmetric Gaussian heat flux distribution given by $q = \exp(-50((x-0.5)^2 + (y-0.5)^2))$ and $d\gamma/dT = -5\ \mathrm{N\,m^{-1}K^{-1}}$ on the top surface of the domain. Here $L = 1$ m is taken as the reference length of the problem. On the other faces of the domain, Dirichlet boundary conditions are applied (i.e. zero velocity, reference temperature $T = 10°\mathrm{C}$). The Prandtl number is 0.3 and we simulate the flow under low gravity ($g = 0.01\ \mathrm{m\,s^{-2}}$) on a $16 \times 16 \times 8$ mesh with an $8 \times 8 \times 4$ processor partitioning. The Rayleigh and Marangoni numbers are 761 and 43.25 respectively. The flow pattern and temperature distribution on the top surface and on the mid vertical ($y = 0.5$ m) plane are given in Plates 2 and 3.

In the second problem, the fluid flow and heat transfer in an $L \times L \times L/2$ domain on a $12 \times 12 \times 6$ mesh with a $4 \times 4 \times 2$ processor partitioning are simulated. Here, the reference length L is 1 as before. The flow is subject to a mixed thermal boundary condition ($q = T - 1$) on the free surface. Side-walls and base have no-slip boundary conditions as before. The temperature on two adjacent side-walls is 0 and on the remaining side-walls $T = 1$. The other parameters are: $Pr = 1$, $Ra = 2200$ and $Ma = 2.5$. The temperature distribution on the surface of the domain is given in Plate 4. In Plate 5, temperature contours and velocity vectors are shown on various slices of the domain.

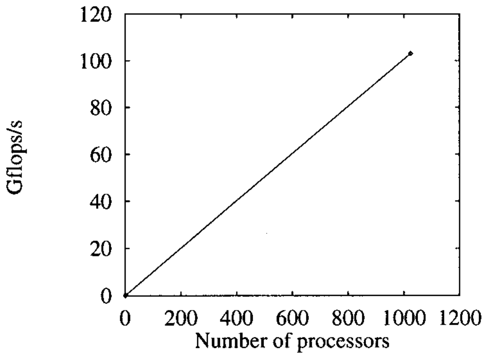

Figure 4. Parallel speed-up scaling on the T3E-900.

9. CONCLUDING REMARKS

Since the development of the transistor 50 years ago, there has been a remarkable expansion in microelectronics and computing. Scientific computing has been a major beneficiary of the microelectronics revolution and at the same time has been a catalyst spurring development of computer hardware, software and algorithms. During the last decade we have seen the emergence of parallel architectures that exploit the price performance of commodity processor availability. This has permitted scientists and engineers to address more complex problems with more complete physical models because larger problems can be addressed in reasonable computation time. The possibility of 'building your own parallel computer' from inexpensive PCs has also opened other avenues for affordable simulation of larger analysis and design problems.

The present work is part of a grand challenge HPC study funded by NASA to investigate scalable parallel coupled viscous flow and heat transfer analysis to explore microgravity and thermocapillary free surface effects. The basic analysis code is, however, more general and provides a framework for scalable distributed HPC applications to this class of transport processes of interest to NASA, industry and elsewhere. The authors have developed a Galerkin finite element formulation and solution algorithm for both fully coupled and decoupled strategies with an implementation using noweb and C. Noweb enables the use of alternate compiler systems and enhances portability of the code (e.g. the 'same' code runs MPI only on the Beowulf system and both MPI and SHMEM on the Cray T3E). The program is also designed for analysis with unstructured grids and irregular partitions [20,21], this being achieved through the use of lists.

As the numerical results indicate, the software can be applied to complex three-dimensional coupled problems and delivers linear near-optimal scaling through 1024 processors on the T3E. Performance on the small PC cluster is encouraging. There are still some major problems at the algorithm level that need to be addressed. Foremost among these is the need for better parallel preconditioners that are robust and dramatically reduce the number of iterations in the linear system subsolves while not significantly degrading scalability and parallel performance. This is, of course, a key open problem for most PDE applications. The most appropriate preconditioning strategies are those based on reduced subdomain (approximate Schur's complement) or multilevel approaches [24] and are part of our ongoing studies in this area. Other future work will be directed to deforming free surfaces, adaptive grids and more complex non-Newtonian fluids with temperature dependent parameters.

ACKNOWLEDGMENTS

This research has been supported by NASA ESS Grand Challenge Project NCCSS-154. The authors express their appreciation to D. Cline, H. Swinney and S. van Hook for their suggestions, to A. Pehlivanov and C. Harle for their assistance.

REFERENCES

1. G.F. Carey, R.T. McLay and R.J. Mackinnon, 'Finite elements modelling of *in situ* vitrification', *In Situ*, **17**, 201–226 (1993).
2. B. Gebhart, Y. Jaluria, R.L. Mahajan and B. Sammakia, *Buoyancy-Induced Flows and Transport*, Reference Edition, Hemishere, Washington, DC, 1992.
3. E.L. Koschmieder, *Bénard Cells and Taylor Vortices*, Cambridge University Press, New York, 1993.
4. B.M. Carpenter and G.M. Homsy, 'Combined buoyant-thermocapillary flow in a cavity', *J. Fluid Mech.*, **207**, 121–132 (1989).

5. M.J. Block, 'Surface tension as the cause of Bénard cells and surface deformation in a liquid film', *Nature*, **178**, 650–651 (1956).
6. M.G. Braunsfurth and G.M. Homsy, 'Combined thermocapillary-buoyancy convection in a cavity. Part II. An experimental study', *Phys. Fluids*, **9**, 1277–1286 (1997).
7. A. Cloot and G. Lebon, 'A non-linear stability analysis of the Bénard–Marangoni problem', *J. Fluids Mech.*, **145**, 447–469 (1984).
8. C. Cuvelier and J.M. Dreissen, 'Thermocapillary free boundaries in crystal growth', *J. Mech.*, **169**, 1–26 (1986).
9. P. Gillion and G.M. Homsy, 'Combined thermocapillary-buoyancy convection in a cavity: an experimental study', *Phys. Fluids*, **8**, 2953–2963 (1996).
10. E.L. Koschmieder and D.S.A. Prahl, 'Surface-tension-driven Bénard convection in small container', *J. Fluid Mech.*, **215**, 571–583 (1990).
11. J.R.A. Pearson, 'On convection cells induced by surface tension', *J. Fluid Mech.*, **4**, 489–500 (1958).
12. Y. Kamotani, S. Ostrach and A. Pline, 'A thermocapillary convection experiment in microgravity', *J. Heat Transf.*, **117**, 611–617 (1995).
13. D. Schwabe, 'Marangoni effects in crystal growth melts', *PCH Phys.-Chem. Hydrodyn.*, **2**, 263–280 (1981).
14. G.F. Carey (ed.), *Parallel Supercomputing: Methods, Algorithms and Applications*, Wiley, Chichester, UK, 1989.
15. R. Glowinski, G. Golub, G. Meurant and J. Periaux (eds.), *Proceedings First International Symposium on Domain Decomposition Methods for PDEs*, SIAM Publications, Philadelphia, 1987.
16. G.F. Carey, A. Boze, B. Davis, C. Harle and R. McLay, 'Parallel computation of viscous flows', *Proc. of HPC '97*, Atlanta, GA, April 1997.
17. E. Barragy and G.F. Carey, 'Parallel-vector computation with high-p element-by-element methods', *Int. J. Comput. Math.*, **44**, 329–339 (1992).
18. P. Fischer and A. Patera, 'Parallel spectral element solution of the stokes problem', *J. Comput. Phys.*, **92**, 380–421 (1991).
19. R. McLay, S. Swift and G.F. Carey, 'Maximizing sparse matrix–vector product performance on RISC based MIMD computers', *J. Parallel Distrib. Comput.*, **37**, 146–158 (1996).
20. B. Hendrickson and R. Leland, 'Chaco: algorithms and software for partitioning meshe', http://www.cs.sandia.gov/CRF/chacp2.html, 1996.
21. G. Karypis, 'Family of multilevel partitioning algorithms', http://www.cs.umn.edu/karypsis/metis, 1998.
22. N. Ramsey, 'Literate programming simplified', *IEEE Software*, 1994, pp. 97–105.
23. N. Ramsey, Noweb Homepage, http://www.cs.virginia.edu/nr/nowela/, 1998.
24. M. Davis and G.F. Carey, 'Iterative solution of the streamfunction–vorticity equations using a multigrid solver with finite elements', *Comm. Appl. Numer. Methods*, **9**, 587–594 (1993).

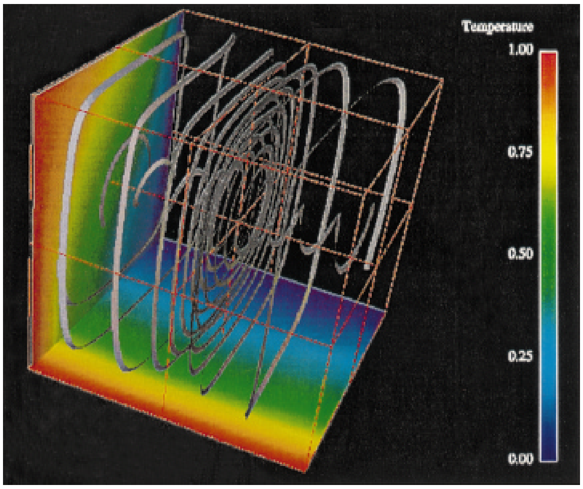

Plate 1. Natural convection flow result.

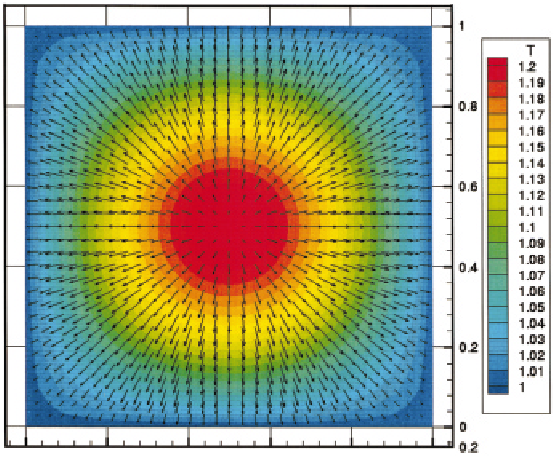

Plate 2. Color temperature plot and vector velocity plot on the top surface.

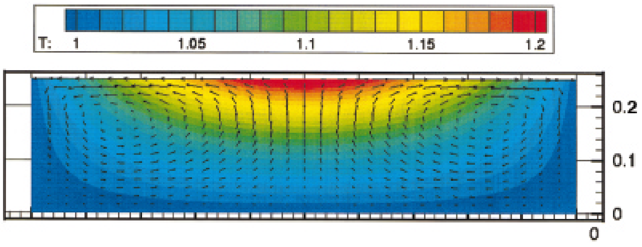

Plate 3. Color temperature plot and vector velocity plot on the section $y = 0.5$.

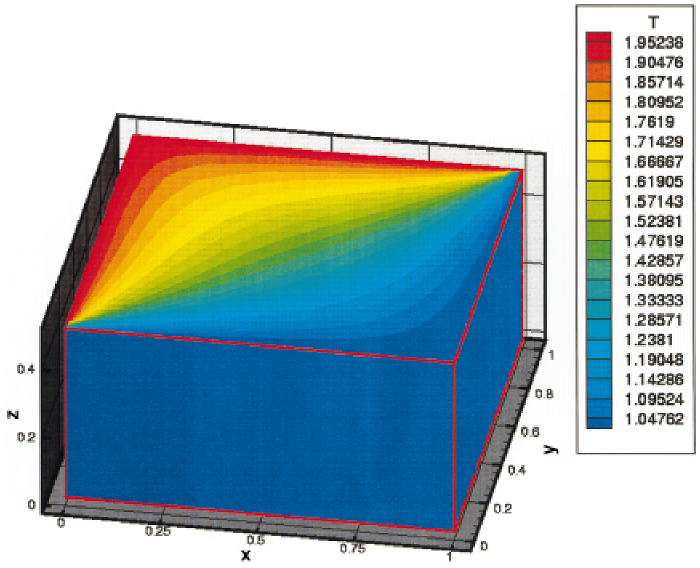

Plate 4. Color temperature plot and on the surface of the domain.

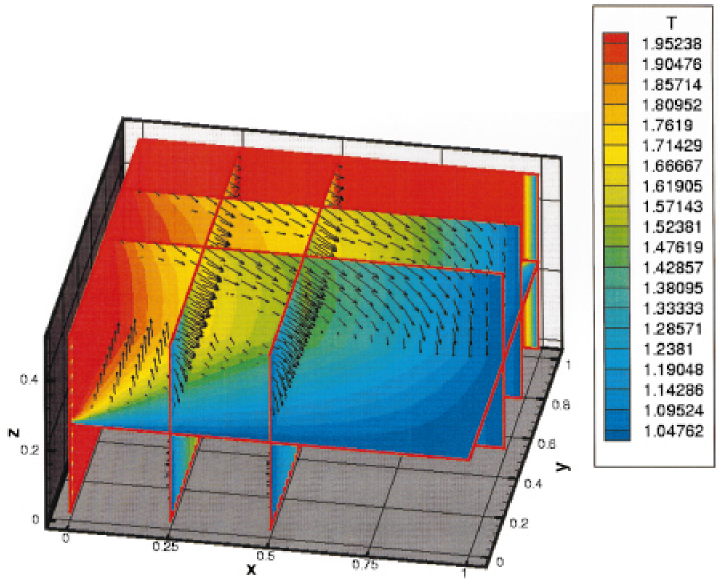

Plate 5. Color temperature plot and vector velocity plot on the internal slices in the domain.