INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids **31**: 911–935 (1999)

PARALLEL SOLUTION OF THREE-DIMENSIONAL MARANGONI FLOW IN LIQUID BRIDGES

M. LAPPA AND R. SAVINO*

Università degli Studi di Napoli "Federico II",
Dipartimento di Scienza e Ingegneria dello Spazio "Luigi G. Napolitano", P. le V. Tecchio 80, 80125 Napoli, Italy

SUMMARY

This paper describes the implementation and performances of a parallel solver for the direct numerical simulation of the three-dimensional and time-dependent Navier–Stokes equations on distributed-memory, massively parallel computers. The feasibility of this approach to study Marangoni flow instability in half zone liquid bridges is examined. The results indicate that the incompressible, non-linear Navier–Stokes problem, governing the Marangoni flows behavior, can effectively be parallelized on a distributed memory parallel machine by remapping the distributed data structure. The numerical code is based on a three-dimensional Simplified Marker and Cell (SMAC) primitive variable method applied to a staggered finite difference grid. Using this method, the problem is split into two problems, one parabolic and the other elliptic A parallel algorithm, explicit in time, is utilized to solve the parabolic equations. A parallel multisplitting kernel is introduced for the solution of the pseudo pressure elliptic equation, representing the most time-consuming part of the algorithm. A grid-partition strategy is used in the parallel implementations of both the parabolic equations and the multisplitting elliptic kernel. A Message Passing Interface (MPI) is coded for the boundary conditions; this protocol is portable to different systems supporting this interface for interprocessor communications. Numerical experiments illustrate good numerical properties and parallel efficiency. In particular, good scalability on a large number of processors can be achieved as long as the granularity of the parallel application is not too small. However, increasing the number of processors, the Speed-Up is ever smaller than the ideal linear Speed-Up. The communication timings indicate that complex practical calculations, such as the solutions of the Navier–Stokes equations for the numerical simulation of the instability of Marangoni flows, can be expected to run on a massively parallel machine with good efficiency. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: parallel computers; Navier–Stokes equations; fluid dynamic instability

1. INTRODUCTION

The floating zone method is an important technique to produce high quality crystals material. By this containerless method, semiconductor crystals can be grown with less contamination, more homogeneity and higher purity. During recent years, floating zone crystal growth in the microgravity environment offered by space laboratories has been proposed and float-zone related experiments have been performed, in order to grow larger and better crystals, since compositional inhomogeneities caused by buoyancy-driven convection can be avoided in low gravity conditions. However, even in the absence of gravity, Marangoni convection is still

* Correspondence to: Università degli Studi di Napoli "Federico II", Dipartimento di Scienza e Ingegneria dello Spazio "Luigi G. Napolitano", P. le V. Tecchio 80, 80125 Napoli, Italy.

CCC 0271–2091/99/220911–25$17.50

Received July 1998
Accepted October 1998

present due to surface tension gradients at the liquid–liquid or liquid–gas interfaces. Experiments have shown that for sufficiently small values of the Marangoni number, the convection in the liquid column is laminar, steady and axisymmetrical, but when the Marangoni number exceeds certain critical values depending on the Prandtl number of the liquid, on the geometry and on the boundary conditions, the liquid motion can undergo a transition to a three-dimensional complex flow pattern [1–3].

Numerical simulations are a decisive tool to reduce the number of expensive experiments, to plan and improve the experimental set-ups and to optimize new production techniques.

Initially, the Marangoni convection in axisymmetric liquid bridges was investigated numerically [4–8] in order to study the bifurcations from a steady to a time-dependent axisymmetric state (i.e. the instability of steady thermocapillary convection) under the constraint of axial symmetry of the flow. Two-dimensional methods are in fact convenient to use due to their efficiency in memory, computing time and costs. They can be reliable for symmetrical geometries and boundary conditions, or if the third dimension is assumed to be infinitely extended. However, if the flow is actually three-dimensional, vital information is lost. In particular, real experiments on Marangoni flow instability show that the supercritical flow and temperature fields are three-dimensional, and this peculiar information is lost in 2D models.

Linear or energy stability theories [9–15] provided important information on the critical conditions for the onset of instability (in the ideal case of steady basic states) and on the flow field organization in the neighborhood of the transition point from axisymmetric to three-dimensional flow; unfortunately these theories do not predict either the real unsteady conditions corresponding to the experimental procedure or the non-linear behavior of the flow far from the bifurcation point.

In recent years the availability of supercomputers and the development of efficient numerical methods provided the possibility to study the problem by direct numerical solution of the three-dimensional Navier–Stokes equations. However, only a few papers have been published on this subject [16–20] due to the fact that enormous speed and memory requirements are necessary. This is particularly true for large Prandtl number liquids and large Marangoni numbers, due to the presence of very thin boundary layers that require a large number of computational grid points and, consequently, a large number of unknown variables. In fact, coarse grids cannot be used because a low resolution does not permit them to 'capture' the instabilities of the Marangoni flow. For these reasons, grid quality and computation speed are 'critical factors' for the study of these problems.

Two possibilities exist for achieving high computational speeds: advanced computers with their intrinsic physical limitations, and parallel computations, that increase the computational speed as the number of processors is increased.

The objective of this work is to develop a parallel and scalable flow solver code to solve a variety of non-smooth problems related to the instability of Marangoni flow in liquid bridges. The Navier–Stokes algorithm must be numerically stable, physically robust and computationally efficient. Previous results [19,20] indicate that a Simplified Marker and Cell method (SMAC) can be successfully utilized for simulations of complex Marangoni flows. The parallelization of this algorithm can only be performed by an articulate numerical solution procedure, in order to optimize simultaneously the numerical scheme, the grid generation and the domain decomposition related to the parallel methods.

In particular, the numerical algorithm developed in the present work has been optimized for both vector and parallel computations (see Section 3). However the vectorial and the parallel performances have been tested independently, since the available computers (Convex 3860 and Cray T3E) are a vector single-processor and a scalar multiprocessor machine respectively.

Whenever vectorial and massively parallel computers were available, both the options would be possible.

The paper runs as follows: Sections 2 and 3 are related to the governing equations and to the numerical solution method respectively; Section 4 describes the vectorization and the parallel implementation of the algorithm. In particular, different approaches have been considered for the parabolic problem (4.2, 4.3, 4.4) and for the elliptic pressure problem (4.4). The general code organization is presented in Section 5. The results are illustrated in Section 6.

2. THE GOVERNING EQUATIONS AND THE BOUNDARY CONDITIONS

The geometry of the problem is shown in Figure 1. A cylindrical liquid bridge of length L and diameter D is held between two coaxial disks with different temperatures. The upper disk is kept at the temperature T_h higher than the temperature T_c of the lower cold disk. The imposed temperature difference is denoted by ΔT ($T_h = T_c + \Delta T$). The geometrical aspect ratio of the bridge is defined as $A = L/D$. The bridge is bounded by a cylindrical and undeformable liquid–gas interface with a surface tension exhibiting a linear decreasing dependence on the temperature. The field equations are the continuity, Navier–Stokes and energy equations, that in non-dimensional conservative form read [18–20]

$$\underline{\nabla} \cdot \underline{V} = 0 \tag{1a}$$

$$\frac{\partial \underline{V}}{\partial t} = -\underline{\nabla}\pi - \underline{\nabla} \cdot [\underline{V}\underline{V}] + Pr\, \underline{\nabla}^2 \underline{V} \tag{1b}$$

$$\frac{\partial T}{\partial t} = -\underline{\nabla} \cdot [\underline{V}T] + \nabla^2 T \tag{1c}$$

where V, π and T are the non-dimensional velocity, pressure and temperature, Pr is the Prandtl number, defined by $Pr = \upsilon/\alpha$ (υ is the kinematic viscosity and α the thermal diffusivity).

The geometry of the problem suggests to use the cylindrical co-ordinates system (z, r, φ) in order to easily impose the boundary conditions (the velocity components are denoted as u, v and V_φ respectively). Thus, the equations read

$$\frac{\partial u}{\partial z} + \frac{\partial v}{\partial r} + \frac{1}{r}\left(v + \frac{\partial V_\varphi}{\partial \varphi}\right) = 0 \tag{2a}$$

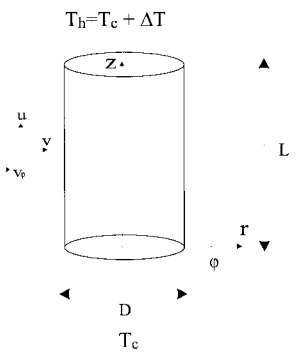

Figure 1. Scheme of the liquid bridge.

$$\frac{\partial u}{\partial t} = -\frac{\partial \pi}{\partial z} - \left(\frac{\partial u^2}{\partial z} + \frac{\partial uv}{\partial r} + \frac{uv}{r} + \frac{1}{r} + \frac{\partial uV_\varphi}{\partial \varphi}\right) + Pr\left(\frac{\partial^2 u}{\partial z^2} + \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \varphi^2}\right) \tag{2b}$$

$$\frac{\partial v}{\partial t} = -\frac{\partial \pi}{\partial r} - \left(\frac{\partial uv}{\partial z} + \frac{\partial v^2}{\partial r} + \frac{v^2}{r} + \frac{1}{r}\frac{\partial vV_\varphi}{\partial \varphi} - \frac{V_\varphi^2}{r}\right) + Pr\left(\frac{\partial^2 v}{\partial z^2} + \frac{\partial^2 v}{\partial r^2} - \frac{v}{r^2} + \frac{1}{r}\frac{\partial v}{\partial r} + \frac{1}{r^2}\frac{\partial^2 v}{\partial \varphi^2} - \frac{2}{r^2}\frac{\partial V_\varphi}{\partial \varphi}\right) \tag{2c}$$

$$\frac{\partial V_\varphi}{\partial t} = -\frac{1}{r}\frac{\partial \pi}{\partial \varphi} - \left(\frac{\partial uV_\varphi}{\partial z} + \frac{\partial vV_\varphi}{\partial r} + \frac{2vV_\varphi^2}{r} + \frac{1}{r}\frac{\partial V_\varphi^2}{\partial \varphi}\right) + Pr\left(\frac{\partial^2 V_\varphi}{\partial z^2} + \frac{\partial^2 V_\varphi}{\partial r^2} + \frac{1}{r}\frac{\partial V_\varphi}{\partial r} - \frac{V_\varphi}{r^2} + \frac{1}{r^2}\frac{\partial^2 V_\varphi}{\partial \varphi^2} + \frac{2}{r^2}\frac{\partial v}{\partial \varphi}\right) \tag{2d}$$

$$\frac{\partial T}{\partial t} = -\left(\frac{\partial uT}{\partial z} + \frac{\partial vT}{\partial r} + \frac{vT}{r} + \frac{1}{r}\frac{\partial V_\varphi T}{\partial \varphi}\right) + \left(\frac{\partial^2 T}{\partial z^2} + \frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \varphi^2}\right) \tag{2e}$$

The initial conditions are

$$t = 0\text{: } \mathbf{V}(z, r, \varphi) = T(z, r, \varphi) = 0 \tag{3}$$

i.e. the liquid is motionless and at ambient temperature.

For $t > 0$, the boundary conditions are the non-slip conditions and the condition of prescribed temperatures on the circular disks, the kinematic condition of stream surface (zero normal velocity), the Marangoni conditions (shear stress balance) and the adiabatic condition on the cylindrical interface:

on the cold disk $0 \le r \le R/L; 0 \le \varphi \le 2\pi$

$$\mathbf{V}(z = 0, r, \varphi, t) = 0;\ T(z = 0, r, \varphi, t) = 0 \tag{4}$$

on the hot disk $0 \le r \le R/L; 0 \le \varphi \le 2\pi$

$$\mathbf{V}(z = 1, r, \varphi, t) = 0;\ T(z = 1, r, \varphi, t) = 1 \tag{5}$$

on the cylindrical free surface $0 \le z \le 1; 0 \le \varphi \le 2\pi$

$$v(z, r = R/L, \varphi, t) = 0 \tag{6a}$$

$$\frac{\partial u}{\partial r}(z, r = R/L, \varphi, t) = -\mathrm{Ma}\frac{\partial T}{\partial z}(z, r = R/L, \varphi, t) \tag{6b}$$

$$r\frac{\partial V_\varphi}{\partial r}(z, r = R/L, \varphi, t) - V_\varphi(z, r = R/L, \varphi, t) = -\mathrm{Ma}\frac{\partial T}{\partial \varphi}(z, r = R/L, \varphi, t) \tag{6c}$$

$$\frac{\partial T}{\partial r}(z, r = R/L, \varphi, t) = 0 \tag{6d}$$

where the reference Marangoni number Ma is defined as $\mathrm{Ma} = \sigma_T(\Delta T)\ L/\mu\alpha$.

3. THE NUMERICAL METHOD

The equations (2a–e) and the initial and boundary conditions (3–6) were solved numerically in cylindrical co-ordinates in primitive variables by a control volume method. The domain was discretized with a non-uniform but structured axisymmetric mesh and the flow field variables defined over a staggered grid. The axial velocity component u is staggered in axial direction

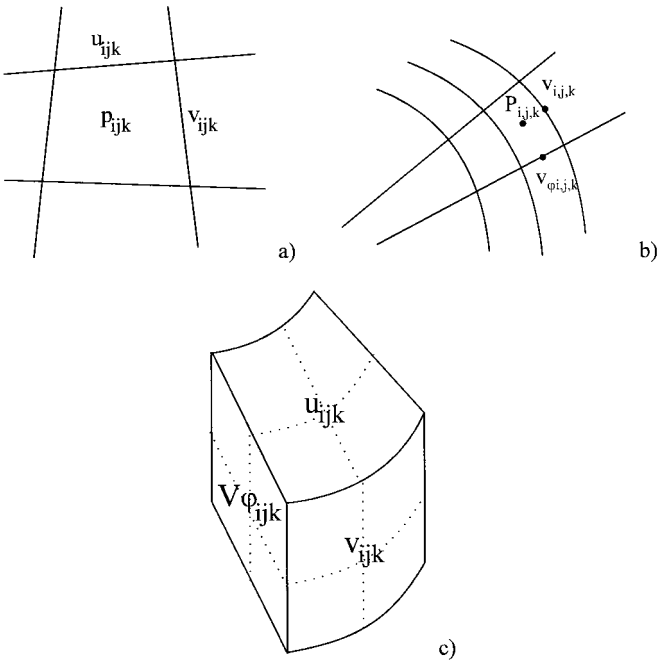

Figure 2. Collocation of the variables on the computational grid.

with respect to the point in which temperature and pressure are computed. In a similar way, the radial and azimuthal velocity components are staggered in radial and azimuthal directions respectively (see Figure 2).

The finite volume approach relies directly on the application of the integral form of balance laws. Thus, the conservation laws have been written for an arbitrary spatial domain Ω bounded by a surface $\partial\Omega$. Since the collocation of the variables on the grid is staggered, each variable is characterized by a different control volume; see Figure 3 where the control volume Ω_u for the axial velocity, Ω_v for the radial velocity and Ω_p for the pressure, the temperature and the azimuthal velocity are shown in a generic meridian plane (the control volumes for p, T and V_φ are the same). Integrating over the generic control volume and using the Gauss theorem to transform volume integrals in surface integrals, the equations read

$$\int_{\partial\Omega_p} \underline{V}\cdot\underline{i}\,\mathrm{d}S = 0 \tag{7a}$$

$$\int_{\Omega_u} \frac{\partial u}{\partial t}\,\mathrm{d}\Omega = -\int_{\partial\Omega_u} [u\underline{V}]\cdot\underline{i}\,\mathrm{d}S + Pr\int_{\partial\Omega_u} \underline{V}u\cdot\underline{i}\,\mathrm{d}S - \int_{\Omega_u} \frac{\partial \pi}{\partial z}\,\mathrm{d}\Omega \tag{7b}$$

$$\int_{\Omega_v} \frac{\partial v}{\partial t}\,\mathrm{d}\Omega = -\int_{\partial\Omega_v} [v\underline{V}]\cdot\underline{i}\,\mathrm{d}S + Pr\int_{\partial\Omega_v} \underline{V}v\cdot\underline{i}\,\mathrm{d}S - \int_{\Omega_v} \frac{\partial \pi}{\partial r}\,\mathrm{d}\Omega \tag{7c}$$

$$\int_{\Omega_{V_\varphi}} \frac{\partial V_\varphi}{\partial t}\,\mathrm{d}\Omega = -\int_{\partial\Omega_{V_\varphi}} [V_\varphi\underline{V}]\cdot\underline{i}\,\mathrm{d}S + Pr\int_{\partial\Omega_{V_\varphi}} \underline{V}V_\varphi\cdot\underline{i}\,\mathrm{d}S - \int_{\Omega_{V_\varphi}} \frac{1}{r}\frac{\partial \pi}{\partial \varphi}\,\mathrm{d}\Omega \tag{7d}$$

$$\int_{\Omega_T} \frac{\partial T}{\partial t}\,\mathrm{d}\Omega = -\int_{\partial\Omega_T} [\underline{V}T]\cdot\underline{i}\,\mathrm{d}S + \int_{\partial\Omega T} \underline{V}T\cdot i\,\mathrm{d}S \tag{7e}$$

where $\underline{i}$ is the orthogonal unit vector oriented outside the control surface.

For high aspect ratios the computational points have been distributed almost uniformly in the computational domain. For low aspect ratios, for the efficient numerical solution of the equations in the presence of the Marangoni boundary layer, the grid has been clustered near the free surface of the liquid bridge (Figure 4).

When parallel computation is involved, explicit schemes are highly efficient and easy to implement on parallel machines. For this reason, a time-explicit SMAC has been adopted (see also Fletcher [21] and Hirsch [22]).

Forward differences in time were used to discretize the time-dependent derivative in the equations (1b) and (1c), obtaining

$$\underline{V}^{n+1} = \underline{V}^n + \Delta t \frac{1}{\Omega}\left[- \int_{\partial\Omega} [\underline{V}^n \underline{V}^n]\cdot \underline{i}\, \mathrm{d}S + Pr \int_{\partial\Omega} [\underline{\nabla}\, \underline{V}^n]\cdot \underline{i}\, \mathrm{d}S \right] - \Delta t \frac{1}{\Omega} \int_{\partial\Omega} \nabla \pi^n \cdot i\, \mathrm{d}S \tag{8}$$

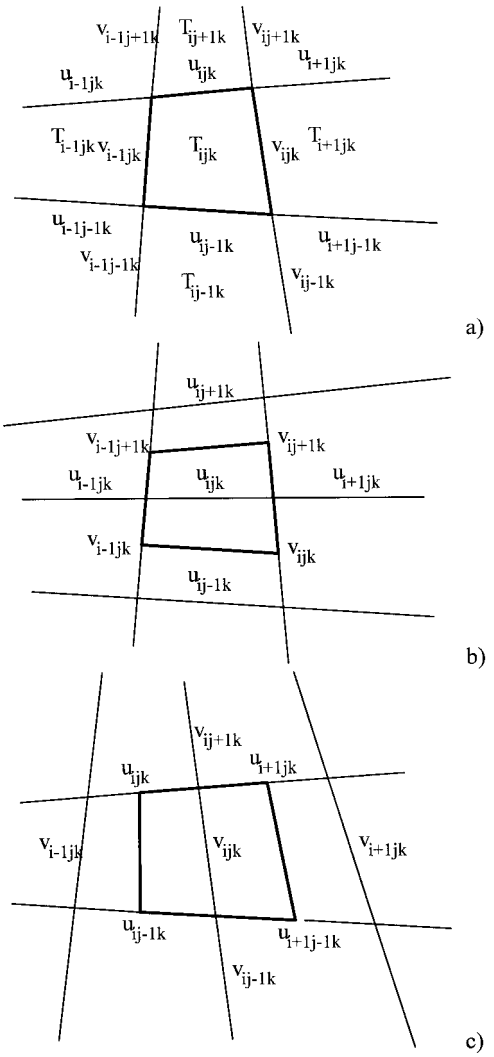

Figure 3. Control volumes adopted for the different variables.

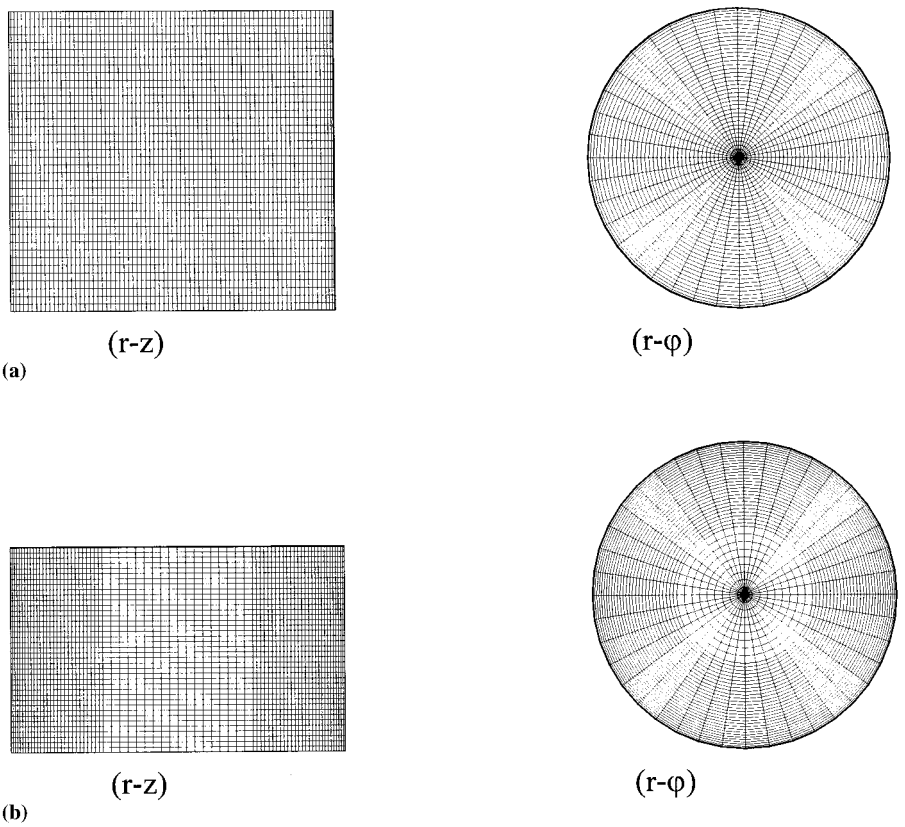

Figure 4. Computational grids used for the numerical simulations.

$$T^{n+1} = T^n + \Delta t \frac{1}{\Omega}\left[-\int_{\partial\Omega} [\underline{V}^n T^n]\cdot \underline{i}\, \mathrm{d}S + \int_{\partial\Omega} [\underline{\nabla} T^n]\cdot \underline{i}\, \mathrm{d}S \right] \tag{9}$$

The SMAC method proceeds as a type of fractional step method by first writing a modified momentum equation and then updating the velocity field using the computed pressure to account for the continuity (1a). More precisely, at each time step, an intermediate velocity field is found without the knowledge of the correct pressure field, and therefore no incompressibility condition is enforced

$$\underline{V}^* = \underline{V}^n + \Delta t \frac{1}{\Omega}\left[-\int_{\partial\Omega} [\underline{V}^n \underline{V}^n]\cdot \underline{i}\, \mathrm{d}S + Pr \int_{\partial\Omega} [\underline{\nabla} \underline{V}^n]\cdot \underline{i}\, \mathrm{d}S \right] \tag{10}$$

The intermediate velocity field is then corrected by a second step in which we solve a pressure equation and then use the computed pressure to produce a divergence-free velocity field. The control volume corresponds to a polyhedron (Figure 2(c)). The surface integrals in equations (9) and (10) (representing the convective and diffusive fluxes across the faces of the control volume) have been computed splitting the integrals in the six contributions related to the six faces of the polyhedron (see Figure 5; N, S, W, E, L and R represent respectively the 'North' and the 'South' directions for the radial coordinate, the 'West' and the 'East' directions for the axial coordinate and the 'Left' and 'Right' directions for the azimuthal coordinate).

For any variable ϕ involved in the computation (T or $\underline{V}$) the overall convective flux across the frontier of the control volume is computed as

$$\int_{\partial\Omega_\phi} [\underline{V}\phi]\cdot\underline{i}\,\mathrm{d}S = F_\mathrm{S} + F_\mathrm{W} + F_\mathrm{N} + F_\mathrm{E} + F_\mathrm{L} + F_\mathrm{R} \tag{11}$$

And the diffusive flux as

$$\int_{\partial\Omega_\phi} \underline{\nabla}\phi\cdot\underline{i}\,\mathrm{d}S = \mathfrak{F}_\mathrm{S} + \mathfrak{F}_\mathrm{W} + \mathfrak{F}_\mathrm{N} + \mathfrak{F}_\mathrm{E} + \mathfrak{F}_\mathrm{L} + \mathfrak{F}_\mathrm{R} \tag{12}$$

Each face contribution $\mathfrak{F}$ to the overall diffusive flux is computed simply evaluating the derivative orthogonal to the faces of the polyhedron.

The convective fluxes are computed using interpolation to 'reconstruct' the value of the velocity and of the convected variable (temperature or momentum) on the faces of the control volume.

In both cases it is necessary to know the distribution of the variables in a certain neighborhood of the considered control volume. In the present work due to the stencil used for the evaluation of the derivatives and to the nature of the interpolations used to compute the physical quantities on the faces of the polyhedron, this neighborhood only involves the adjacent control volumes.

The pressure is computed by solving a Poisson equation resulting from the divergence of the momentum equation with the help of the continuity equation:

$$\nabla^2\pi^n = \frac{1}{\Delta t}\underline{\nabla}\cdot\underline{V}^*$$

that in integral form becomes

$$\int_{\partial\Omega_p} \underline{\nabla}\pi\cdot\underline{i}\,\mathrm{d}S = \frac{1}{\Delta t}\int_{\partial\Omega_p} \underline{V}^*\cdot\underline{i}\,\mathrm{d}S \tag{13}$$

The Poisson equation is solved with a Successive Over Relaxation (SOR) iterative method.

The velocity field is finally updated using the computed pressure field to account for continuity

$$\underline{\boldsymbol{V}}^{n+1} = \underline{\boldsymbol{V}}^* - \Delta t\underline{\nabla}\pi^n \tag{14}$$

The temperature field at time $(n+1)$ is obtained from eq. (9) after the calculation of the velocity.

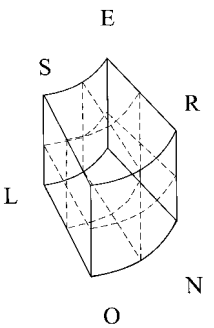

Figure 5. 3D sketch of the generic control volume.

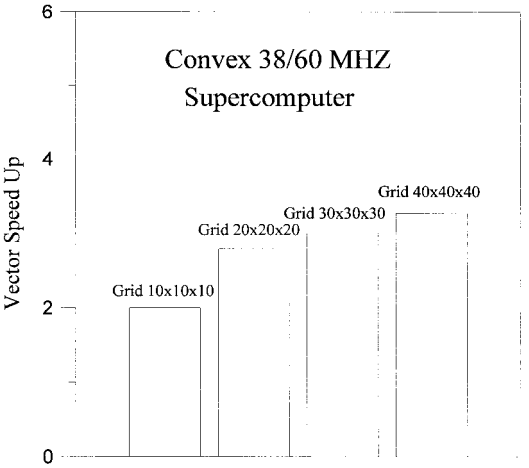

Figure 6. Speed-Up function obtained on the Convex 3860 Supercomputer.

In conclusion, with the SMAC method, the problem has been split in four scalar parabolic equations (three from equation (10) plus the energy equation (9)) and one elliptic equation (13).

4. PARALLEL/VECTOR STRATEGY

Two possible strategies can be pursued to achieve super-calculus: (1) vectorization, in which a large amount of data (arranged in arrays) can be handled at the same time by special vector registers and a special unit (pipeline); (2) true parallel computation when different instructions may be executed in parallel by several processors.

4.1. Vectorization

A higher level of parallelism is found in vector processors, where the use of special hardware (vector register) may permit parallelizing a computational loop over a segment (of length matching that of the vector registers) of an array. It must be pointed out that vector processing is only applicable to those loops where the data to be processed in successive iterations do not carry any dependency (as it would be the case for a loop where one iteration affects the input data of another one).

In this work, using an explicit in time numerical method, a full (100%) vectorization of the four parabolic equations involved in the computation was obtained. When the numerical procedure is explicit in time, in fact, the variables at the time step $n+1$ can be computed as functions of the variables at the time step n (the variables at $n+1$ depend exclusively on the variables at the previous time step and do not carry any dependency among them).

For the elliptic Poisson equation, the full vectorization was not possible (the compiler reported 72% for the subroutine used as solver of this equation) since the SOR iterative algorithm introduces strong dependency among data to be processed.

On a Convex 3880 vectorial supercomputer the code has reached a 'vector Speed-Up' (defined as the ratio of the scalar CPU time over the vector CPU time) between 2 and 4 (see Figure 6). In particular this Speed-Up is a function of the number of variables involved in the

computation (i.e. of the number of grid points). Figure 6 shows that when the number of variables (proportional to the number of grid points) is increased, the vector Speed-Up increases. When the dimensions of the arrays containing the variables of the problem increase, the calculus can take a better advantage of the vector hardware and in particular of the vector registers and of the pipeline mathematical unit [23].

A more detailed explanation of this behavior can be given describing briefly the structure of a vector pipeline unit.

A pipeline unit can be considered as an 'assembly line' characterized by several working stations where the variables to be processed 'flow' continuously. It is a sequence of elaboration stages where the stages are sub-units each executing a different sub-operation on the variables that move in the pipe.

Typically, an arithmetical operation on two scalars requires more sub-operations; each sub-operation needs a clock period; the global time required for the operation is the sum of the clock-periods for the different sub-operations in which the arithmetical operation considered can be split.

In a pipe, the variables (stored in a vector array) move continuously spending a clock period at each station so that the time required to execute the operation on the entire vector is smaller than the sum of the times that would be required to execute the same operation on all the variables stored in the vector array considered separately.

If the pipe consists of q stages, and the vector array to be processed contains n scalars, the pipe needs $q+(n-1)$ clock periods to complete the operation on the entire vector: where the first q periods are required to complete the operation on the first scalar of the vector array. Without pipelining the same global number of operations can be executed in $(n \cdot q)$ periods. On the basis of this analysis it is possible to give an estimation of the vector Speed-Up as

$$S_{\mathrm{v}} = \frac{nq}{q+n-1} \tag{15}$$

In the present work, the variables involved in the computations have been stored in 3D arrays having dimensions Nz, Nr, $N\varphi$ (the number of data items in the axial, radial and azimuthal directions respectively). The length n of the vector array processed by the pipeline unit is defined by the inner loop of the triple structure loop used to handle the entire 3D array of the variables (n is 20 for a grid $20 \times 20 \times 20$ and 40 for a grid $40 \times 40 \times 40$).

The S_{v} function increases when n is increased and this explains the behavior shown in Figure 6.

4.2. Domain decomposition and data mapping

True parallel processors have the capability of executing different instruction streams (tasks) in parallel, and that of permitting communication between tasks.

In these computers (we refer to distributed memory systems NORMA: No Remote Memory access) the global algorithm has to be split into several sub-kernels [24–27]. Each processor solves a task and cannot 'see' directly the variables stored in the local memory of the other processors solving the other tasks. The different nodes have to communicate via special messages using an 'internode communication network' (Message Passing philosophy).

This Section is dedicated to the description of the parallel algorithm used for the solution of the parabolic equations (9) and (10) on the Cray T3E massively parallel supercomputer. The data distribution among the various processors of a parallel machine is a key factor in the efficiency of a parallel implementation. For many scientific applications, the optimal data

distribution is not obvious and requires experimentation. The parallelization approach employed in the present work is based on domain decomposition. The computational domain is split into p sub-blocks (or sub-domains) without overlapping neighboring parts in a manner that provides a complete partition [24]. Each sub-domain is topologically equivalent to a simple domain such as a cylinder: this splitting comes from the grid-generation process. Hereafter p denotes the number of processors available on the parallel machine, and Nz, Nr, $N\varphi$ denote the number of data items (i.e. grid points) in the axial, radial and azimuthal directions. The three dimensional data are partitioned into Nz two dimensional planes of $NrN\varphi$ data items each. The first Nz/p planes are mapped to processor P_0, the next Nz/p planes are mapped to processor P_1 and so on. An example of this mapping is shown in Figure 7 (for $p = 4$).

In this way the solution domain is split into a connected set of axisymmetric sub-blocks in three dimensional space. Each block has at most two surrounding blocks, although some at physical boundaries have fewer neighbors. In fact, interior blocks have two neighboring blocks, whereas those at the physical boundary only have one neighbor.

Since the domain is partitioned in equal sub-domains, each processor has exactly the same number of grid points. The parabolic equations (9) and (10) are solved within each sub-domain. Since an explicit in-time scheme is employed, at each time step each sub-domain is completely independent of its surrounding (the variables computed at the step $n+1$ are completely uncoupled since they depend exclusively on the variables at the previous time step), parallelism simply being achieved by introducing a new boundary condition, denoted as inter-block or inter-domain boundary condition. Each processor works with one of these sub-blocks, and exchanges data across the boundaries of the sub-blocks. The entire application can be imagined as a number of kernels with different data-distribution requirements, which necessitates data movement between the different kernels.

4.3. Interprocessor communication and synchronization

To ensure a correct implementation of the numerical algorithm on the partitioned grid, data exchange is necessary close to the 'partition boundaries' of each subgrid local to a certain processor.

Each processor contains a cylindrical subgrid surrounded by some 'ghost grid points' which are duplicates of grid points contained in other processors, as depicted in Figure 8 where the computational mesh in a generic meridian plane is shown. In addition to the block of data owned by each processor, there are two surrounding layers of 'ghost' points. These ghost points are updated by the Message-Passing phase of the calculation, and provide data continuity from the neighbor sub-block for the computation.

An example of the parallel numerical procedure and of the utilization of the ghost points is shown in Figure 8. Suppose to compute any variable ϕ (momentum or temperature) at the time step t^{n+1} in the 'internal' point A as a function of the solution at the previous step t^n.

The processor (i) is able to compute the diffusive and convective fluxes across the faces of the control volume containing the grid point A, since the values related to the grid points adjacent to this control volume are stored in its local memory.

Similarly, at the 'boundary' point B, the processor is able to compute the diffusive and convective fluxes across the faces N, S and E but it 'cannot' compute directly these fluxes across the face W, since external values are not stored into its local memory, but they are stored in the local memory of the processor $(i-1)$. Therefore, the processor $(i-1)$ has to

'communicate' these values to (i) to complete its local computations. These value are 'received' by the processor (i) and 'located' in a 'geometrical extension' of its sub-domain represented by the ghost cells.

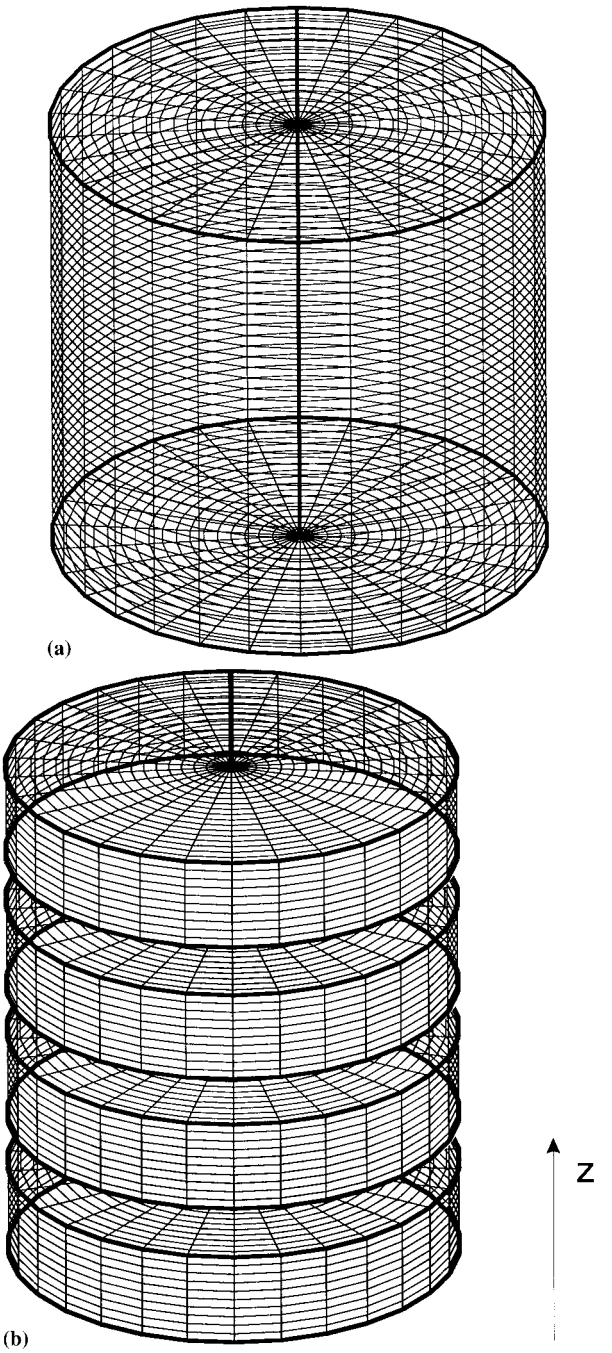

Figure 7. (a) Computational domain and (b) grid partition.

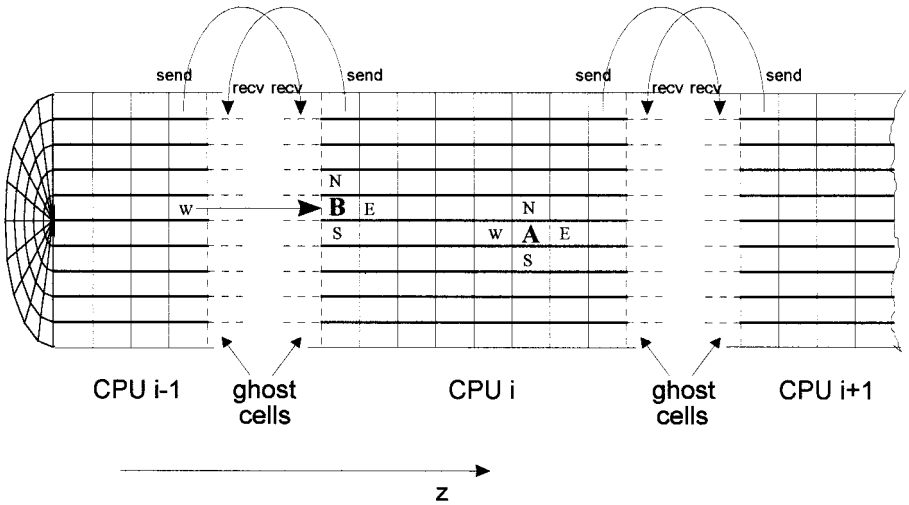

Figure 8. Data exchanges between neighboring processors (the data lying in partition boundary-layers in each processor are sent out which is stored in the blocks for ghost grid points in the neighboring processors).

The number of ghost points on each partition boundary of the subgrid depends on the numerical algorithm. For the SMAC method using a standard Laplacian stencil, one ghost grid point on each side is needed for the local subgrid at each level. Therefore, in the Navier–Stokes flow solver, it was allocated storage for one ghost grid point for each sub-grid on each partition boundary.

The code is implemented with an extremely simple message-passing model, consisting of only *send* and *receive* (see Figure 8). The simplicity of this model implies easy portability. The message-passing is written using the Message Passing Interface (MPI [28,29]), ensuring portability.

The sending data contains boundary information that is placed into the ghost locations of the neighboring processors. After data exchanges are performed, a new computation step can start. Each processor, therefore, only needs to know its nearest surrounding processors on the logical processor mesh.

Each processor computes the data of its sub-block independent of the others, then exchanges data with its neighboring sub-domains. The exchange consists of sending messages to each of the neighbor blocks, where the size of each message is proportional to the number of grid points on the common face. The processor then waits to receive the corresponding messages from the neighbor processors, and when these are received, the cycle starts again with the computation phase.

The code runs in loosely synchronous cycles of computation followed by data exchange across the faces of the sub-blocks.

In order to optimize the synchronization, the subroutines for the management of parallel events, provided by the MPI FORTRAN library (mpi_barrier), have been utilized.

4.4. The elliptic equation and the multisplitting technique

When the grid partition strategy is employed, the code consists of a number of computationally intensive kernels. Dependent upon the equations (parabolic or elliptic), some of these kernels are executed locally on a single processor and the rest are executed globally across the

processors. The kernels that are executed globally require strong communication between processors.

The efficiency of the numerical solution of the pseudo-pressure equation introduced by the SMAC method is of paramount importance in unsteady three-dimensional, incompressible flow simulations. The pressure solution demands, in fact, at least 50–70% of the CPU time depending on the grid size.

In the 'serial' version of the code an iterative SOR method was adopted for the solution of the linear elliptic equation. The disadvantage is that a fully SOR or (Gauss–Seidel) method involves the solution of linear system of equations over the whole solution domain and is therefore, on a parallel machine, much more computationally intensive than explicit steps for parabolic equations.

Over the years many solution methods have been proposed for linear problems. In particular, much attention has recently been paid to iterative 'splitting' methods, that generalize the matrix splitting methods (e.g. Jacobi, Gauss–Seidel and SOR) for solving systems of linear equations. O'Leary and White [30] have proposed a parallel 'multisplitting' iterative method using different splittings of the coefficient matrix. The results obtained from the splitting iterations are combined to define the multisplitting cycles. Thus, the method may be effectively implemented on multiprocessors.

This technique has been employed to solve the present problem; the system matrix has been decomposed by columns into p blocks. The global problem is then replaced by a sequence of local problems to be solved in parallel.

This approach has great potential for parallelism (it runs as a convergence accelerator for the parallel code), is essentially architecture-independent and uses much serial expertise. In fact, the multisplitting philosophy can be regarded as a domain decomposition algorithm, in which each CPU only provides an updated solution for a portion of the domain.

Specifically, the pressure variable domain $\underline{\pi} \in R^k$ (where k is the number of grid points, i.e. $k = NzNrN\varphi$) is partitioned according to $\underline{\pi}(\underline{\pi}_1, \underline{\pi}_2, \ldots, \underline{\pi}_p)$ where each subdomain has $\underline{\pi}_i \in R^{ki}$ and $\Sigma_{i=1}^{p} k_i = k$ without overlap of subdomains [31].

The solution π_i is then the solution with updated values only on the i portion of the partition. The basic algorithm using p processors follows for all processes i, $1 \leq i \leq p$:

1....communicate values near partition boundaries to the neighboring processors.

2....calculate the pseudo-pressure π over each subdomain using SOR.

3....synchronization.

4....test for convergence locally.

5....communicate convergence result to all processors.

The method can be seen as a Jacobi method from a global point of view (each processor does not 'see' the values computed by the others processors until each iterative step has been completed) and as a SOR method from a 'local' point of view (each processor uses a SOR method over the variables stored in its local memory). Within the 'local' iterations, the system of linear equations is solved neglecting coupling matrix elements related to grid points in different blocks. This means that each iterative step is performed within a single processor with no communication to other processors. Only after the iterative step has been completed is there communication with other processors.

Testing for convergence can be carried out either locally or globally. In the former case it is only necessary to share a logical variable with all the other processes. Otherwise the array must be accumulated and a global check performed. In this paper testing globally has been adopted. The pseudo-pressure array is accumulated and the convergence check performed by a single CPU (the 'Master') that then communicates the result of the test to the others processors

(slaves). A master–slave model has been employed because it requires minor modification of the original test employed in the serial version of the code for convergence.

Convergence results show that 'extra' iterations are needed by the multisplitting scheme with respect to the 'serial' version of the SOR method. The number of iterations required by the parallel SOR method described above is in fact larger than that required by the serial version; however it is always smaller than that required by the serial Jacobi method (according to this result and to the fact that the multisplitting method can be seen as a Jacobi method from a global point of view and as a SOR method from a local point of view, it follows that this method can be considered a hybrid technique).

5. PROGRAM ORGANIZATION

The code has been written entirely in FORTRAN 90, making full use of the advanced concepts provided by the language, such as dynamic data structures and pointers. Thus, a much more compact, well-structured, and maintainable code has been obtained. Storage for all grid variables is allocated at run time. This strategy of dynamic memory allocation offers a greater flexibility in data structure manipulations and more efficient use of memory than a static memory allocation, and the user is also alleviated from the burden of calculating storage requirements. After initialization of the problem to be solved and some algorithm parameters, a preprocessing routine is called. The preprocessing routine constructs the set of nested grids and the corresponding set of logical processor meshes. The numerical solution proceeds in three major stages:

1. Topology: perform domain decomposition of the solution domain.
2. Grid generation: create a high-quality grid within each domain. Spatial discretization reduces the Navier–Stokes equations to a set of partial differential equations.
3. Explicit solution: advance explicitly in time by using the SMAC method.

For the stage 3, the combined use of locks and events permits the creation of complex communication and synchronization patterns:

3a. Solution of the three parabolic momentum equations neglecting the pressure gradient.
3b. Synchronization.
3c. Solution of the elliptic equation (13).
3d. Synchronization.
3e. Update using the computed pressure field to account for continuity (14).
3f. Synchronization.
3g. Solution of the parabolic energy equations.
3h. Synchronization.

Parallelization is performed using FORTRAN extension and special library subroutines (mpi_send, mpi_recv, mpi_barrier).

When making the effort to parallelize the code, it is of great interest to reduce the inefficiency, which is defined as $1 - p[T(1)/T(p)]$ where p is the number of processors, $T(p)$ is the time taken to run on the p processors, and $T(1)$ is the time taken to run on a single processor (serial version of the code).

The parallel code developed in this work has an inner loop structure of loosely synchronous cycles of computation and communication between processors whose physical domains are adjacent. In this case, there are usually two contributions to inefficiency: load–balance inefficiency and communication inefficiency. Load–balance inefficiency could occur because

the processors have different amounts of computation to perform, which for the code means different numbers of grid points. In this case the algorithm in fact runs at the speed of the processor with the largest number of grid points (the minimum speed). In the present work this cause of inefficiency has been eliminated by splitting the grid into equal subblocks.

Communication inefficiency arises from the time taken for messages to pass between the processors In the present case this cause of inefficiency has been reduced in the grid-generation phase, because the subblocks have reasonably low surface area to volume ratios, so that the communication inefficiency is low.

6. RESULTS

To test the capability of the code to 'capture' the instability of Marangoni flow in cylindrical liquid bridges, two different liquid bridges have been considered with $A = 0.6$ and $A = 1$.

The Prandtl number is $Pr = 0.04$. The runs have been performed on the Convex 3880/60 Vector computer in 'serial' mode and on the Cray T3E massively parallel computer in 'parallel' mode in order to compare the results and to validate the parallel method introduced.

Some of the typical results are illustrated in Figures 9–19.

When the basic axisymmetric field becomes unstable, after a short transient a three-dimensional steady flow regime develops (stationary supercritical state). The transition between subcritical symmetric and supercritical asymmetric state can be determined by the time temperature and velocity profiles in a well-defined grid point. Before the transition point, the temperature profiles obtained with 2D and 3D numerical computations are almost coincident. After the onset of instability the results of the 3D code depart from 2D computations, due to the symmetry breaking and to the three-dimensional flow field organization (see Figure 9).

On the other hand, the computations indicate that the thermal and flow field organization in the supercritical state may be very complex, depending on the geometrical aspect ratio of the liquid bridge ($A = L/D$).

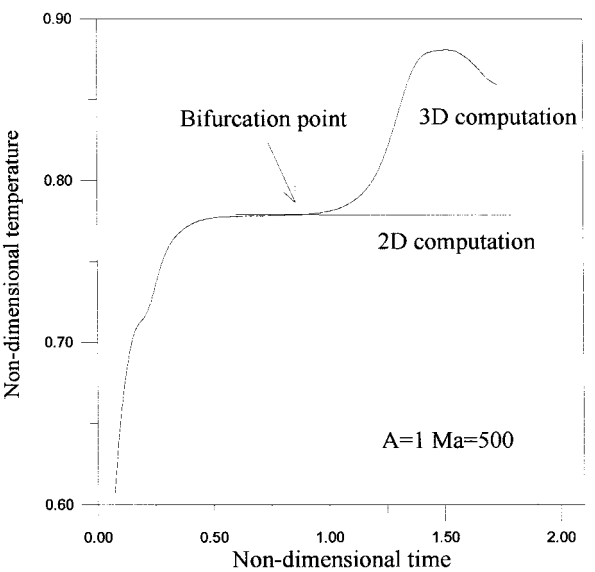

Figure 9. Temperature in the control point $z = 0.75$, $r = 0.9/2A$, $\varphi = 0$ for $A = 1$ and $Ma = 500$.

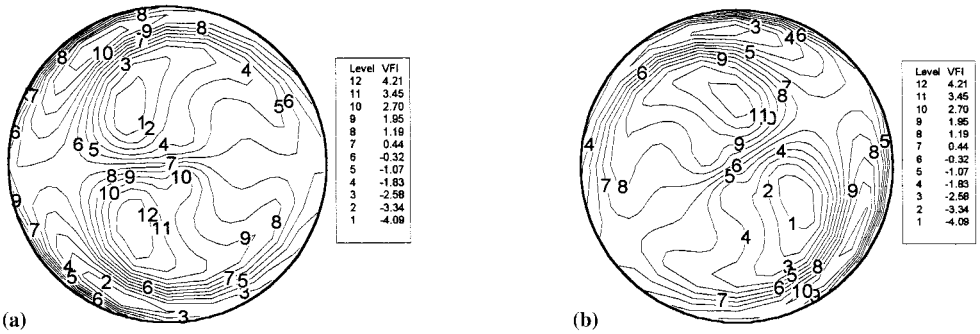

Figure 10. Azimuthal velocity at $z = 0.75$ for $A = 1$: (a) serial results, (b) parallel (4 CPU) results.

For $A = 1$ and $A = 0.6$, two different modes for three-dimensional supercritical flow pattern are present. For $A = 1$, in the generic cross-section orthogonal to the liquid bridge axis there are two azimuthal convective cells (Figure 10) and two thermal spots on the liquid bridge surface (Figure 14). For $A = 0.6$, the convective cells are four (Figure 15) and there are four thermal spots on the free surface (two hot and two cold, Figure 19).

For $A = 1$, the supercritical flow appears as an inclined toroidal vortex; for $A = 0.6$ as a deformed torous.

For $A = 1$, there are two asymmetrical vortex cells in each meridian plane of the liquid bridge. For $A = 0.6$, the flow field structure is on the whole three-dimensional and depends on the azimuthal co-ordinate, but in each meridian plane the velocity and temperature are symmetric.

This behavior is particularly evident in Figures 13 and 18 where the vector plots in a meridian plane are illustrated for $A = 1$ and $A = 0.6$. For $A = 1$ (Figure 13), one of the two vortex cells in the section prevails on the other and is extended along the whole axial plane of the bridge For $A = 0.6$ (Figure 18), the vector plot is symmetric.

The figures carrying (a) represent 'serial' results whereas the figures carrying (b) represent 'parallel' results (obtained using 4 CPUs).

The numerically determined critical Marangoni number is $\mathrm{Mac} = 85$ for $A = 0.6$ and $\mathrm{Mac} = 125$ for $A = 1.0$. The quantitative difference between 'serial' and 'parallel' results lies below 1%.

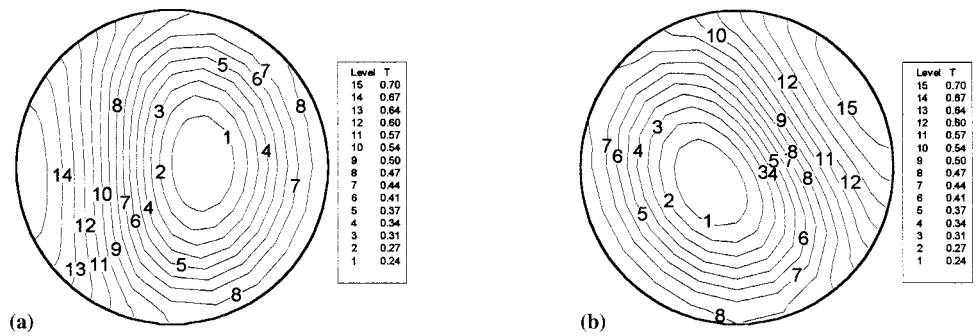

Figure 11. Temperature field at $z = 0.75$ for $A = 1$: (a) serial results, (b) parallel (4 CPU) results.

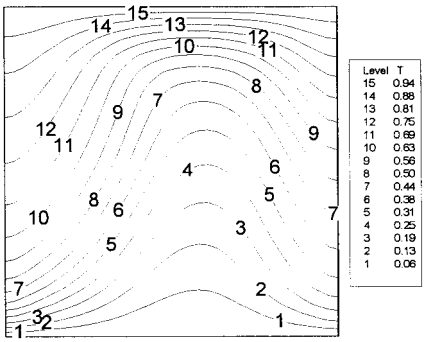

Figure 12. Temperature distribution at $\varphi = 0$ for $A = 1$.

The figures show that the supercritical flow patterns in the cross section obtained from the 'parallel' version of the code, appear to be 'rotated' with respect to those obtained with the 'serial' version.

In the linear stability analyses the process is studied, 'forcing' in the model equations 'small' disturbances typically represented by a Fourier expansion. The critical Marangoni number is then computed as the value of the Marangoni number that corresponds to a non-negative growth rate of the disturbances (above this value the arbitrary imposed small disturbances can grow and give rise to the bifurcation).

In these analyses the position of the azimuthal extreme depends on the '*a priori*' fixed shape of the functions superimposed on the basic state field as initial conditions for the disturbances.

When the problem related to the bifurcation of the Marangoni flow is solved through direct numerical solution of the time-dependent and non-linear Navier–Stokes equations, as in the present work, no *a priori* known small azimuthal disturbances are superimposed on the solution. The role of the 'small distortions' of the linear stability analyses is played in this case by the 'numerical noise' introduced in the computation by the 'round-off' errors of the memory registers of the computer used for the simulation If the numerical simulation is performed using a serial (mono-CPU) computer (that has 'fixed' memory registers) the numerical noise is fixed (it works as a 'fixed' initial condition for the disturbances) and for this reason the simulation, when repeated, gives always the same result; if a parallel computer system is used, the numerical noise is not fixed since it depends on the processors (and related 'private' memory registers) that are involved (depending on the availability) in the computation.

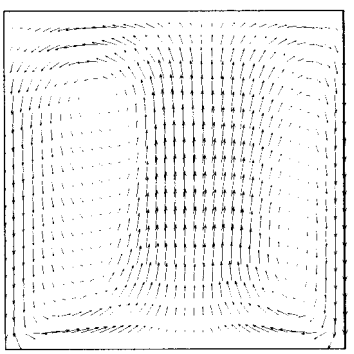

Figure 13. Velocity field at $\varphi = 0$ for $A = 1$.

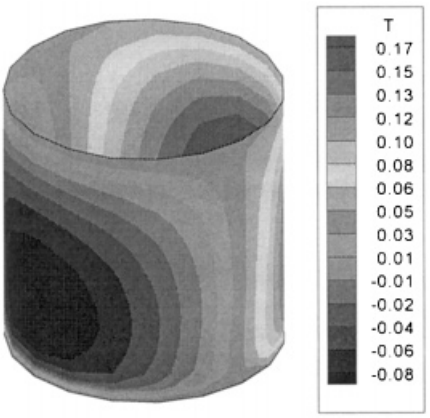

Figure 14. Temperature disturbances on the liquid bridge surface for $A = 1$.

However, an extensive numerical experimentation indicated that the critical parameters of the transition process (critical Marangoni number, critical azimuthal wave number) and the amplitude of the distortions are independent of the numerical noise related to the parallel environment.

7. PARALLEL PERFORMANCES

The Cray T3E supercomputer has 128 DEC Alpha processors (model 21164). The network topology is 3D Torus (480 Mbyte/sec payload bandwidth through each one of the six torus directions). The clock cycle is 3.33 nsec. The memory is 16 Mword (64 bits) per Processor (2 Gword global memory) and the peak performance is 76.8 GigaFLOPS.

The parallel performance of an application code is usually judged by two measurements: Speed-Up and Scaling. Speed-Up is measured by fixing the problem size (or grid size in our case) and increasing the number of processors used. Scaling is measured by fixing the local problem size in each processor and increasing the number of processors used.The Speed-Up parameter is defined as

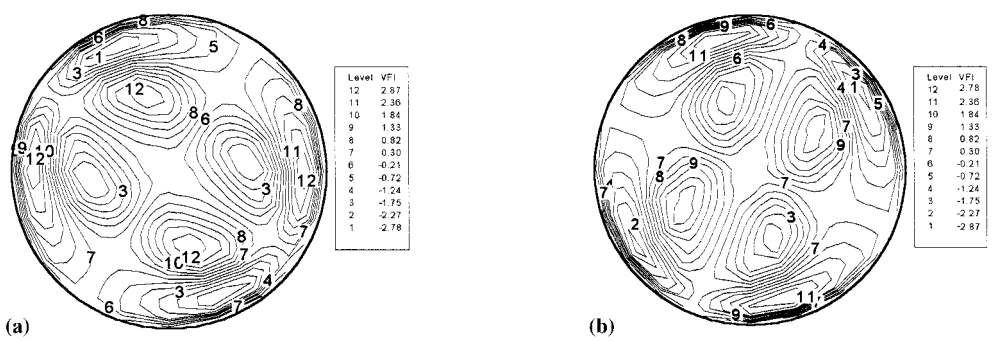

Figure 15. Azimuthal velocity at $z = 0.75$ for $A = 0.6$: (a) serial results, (b) parallel (4 CPU) results.

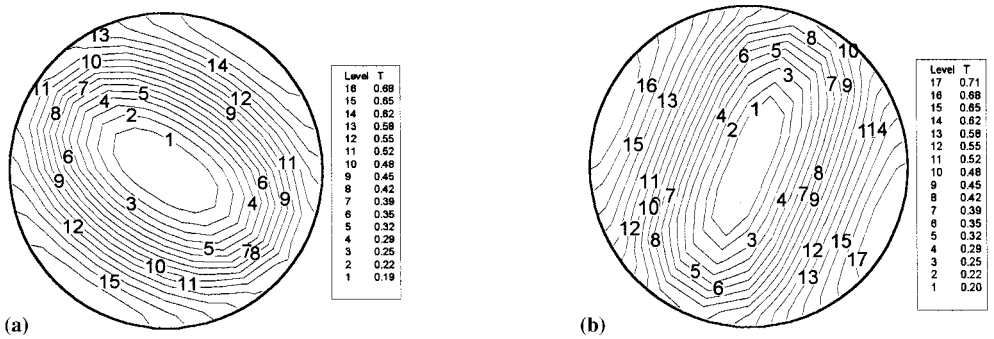

Figure 16. Temperature field at $z = 0.75$ for $A = 0.6$: (a) serial results, (b) parallel (4 CPU) results.

$$S_{\mathrm{p}} = \frac{T(1)}{T(p)} \tag{16}$$

where $T(1)$ is the time required for the 'serial' computation (single CPU) and $T(p)$ is the time required by the algorithm to perform computations using p parallel processors.

The serial version of the algorithm does not correspond to the parallel one processor execution of the code since the latter requires additional time for the initialization of the tools introduced by the compiler for the managing of the multiporcessing environment.

$T(p)$ may be expressed as the sum of the time required for the serial operations (operations which cannot be executed in parallel mode, e.g. input/output operations) plus the time required for the 'true' computations on each processor T_{p} and the communication overhead T_{o}

$$T(p) = T_{\mathrm{s}} + T_{\mathrm{p}} + T_{\mathrm{o}} \tag{17}$$

Typically T_{p} and T_{o} are combined in a single non-dimensional parameter known as the communication cost

$$c = \frac{T_{\mathrm{o}}}{T_{\mathrm{p}}} \tag{18}$$

The efficiency of the algorithm is defined as

$$E_{\mathrm{p}} = \frac{S_{\mathrm{p}}}{p} \tag{19}$$

Equation (16) is actually a comparison between the performances of the serial version of the code and of the parallel program running on p processors. The parallel version takes into account the additional source code which the programmer may have to insert (typically for

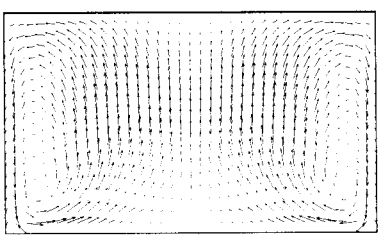

Figure 17. Temperature distribution at $\varphi = 0$ for $A = 0.6$.

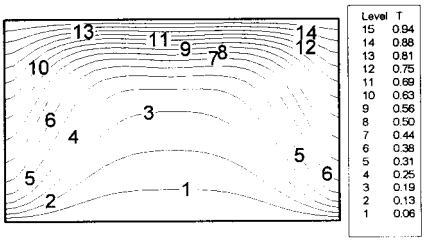

Figure 18. Velocity field at $\varphi = 0$ for $A = 0.6$.

synchronizing different tasks and for implementing the required communications between them) in the serial code in order to obtain parallel operation. Moreover it any takes into account the additional object code which the compiler may have to insert for managing the multiprocessing environment (parallel overhead). For these reasons it follows that the Speed-Up obtained by parallelizing a given code will be lower than p (and consequently the efficiency will be lower than 1). On the other hand, it can be made very close to p by taking all limiting factors into account during the design of the code.

Figure 20 shows the Speed-Up function S_p for the present code on the Cray T3E. For a fixed problem size (i.e. for a fixed grid) S_p increases with the number of processors employed and for a fixed number of CPUs it increases with the dimension of the problem. Figure 21 shows the corresponding parallel efficiency E_p. Figures 20 and 21 show that the advantages obtained by parallelism diminish rapidly as the number of processors exceeds some threshold. This behavior is explained by the fact that the parallel efficiency E_p is largely determined by the ratio of local computations over interprocessor communications. As the number of processors increases, for a fixed problem size, the local communication cost and the cost for global operations will eventually become dominant over the local computation cost after a certain stage. This high ratio of communication to computation makes the influence of a further reduction in the local computation very small on the overall cost of running the application.

For the present code, the best parallel efficiency is achieved on the finest grid (see Figure 21), where the communication overhead is less important than the large amount of computations.

Indeed, the computational cost is proportional to the number of gridpoints, and the communication overhead is proportional to the number of gridpoints associated with the

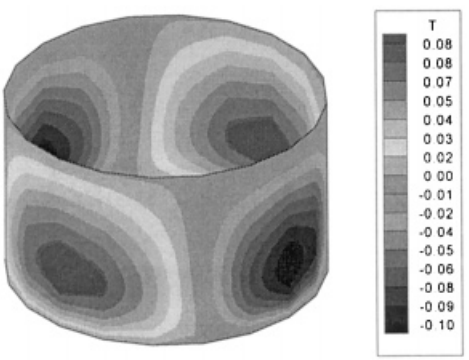

Figure 19. Temperature disturbances on the liquid bridge surface for $A = 0.6$.

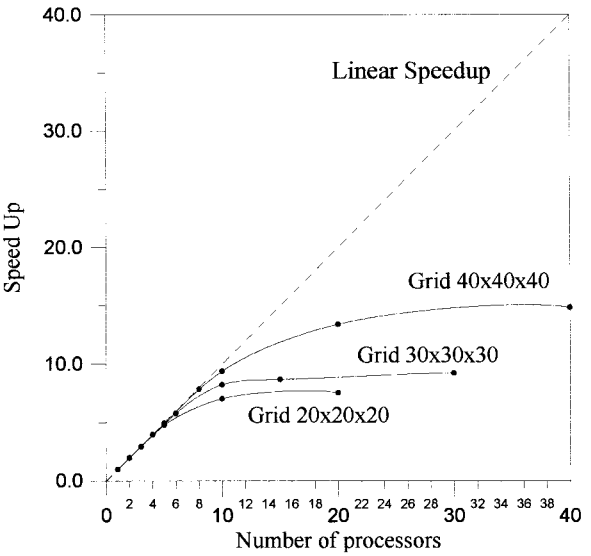

Figure 20. Speed-Up function obtained on the Cray T3E.

boundary of the sub-domains. Thus, the communication cost $c = T_o/T_p$ of the code is proportional to the surface area to volume ratio for a processor; and also proportional to the ratio of the amount of communication per boundary gridpoint, to the amount of work (flops) per internal gridpoint.

Therefore, the best performances obtained using finest grids (Figures 21 and 22) are explained by the fact that, for the present algorithm the computation cost T_p scales as $O(Nr \cdot N\varphi \cdot Nz/p)$, where Nz, Nr and $N\varphi$ are the number of grid points in the local grid and p the number of processors used; whereas the communication overhead T_o scales as $O(Nr \cdot N\varphi)$.

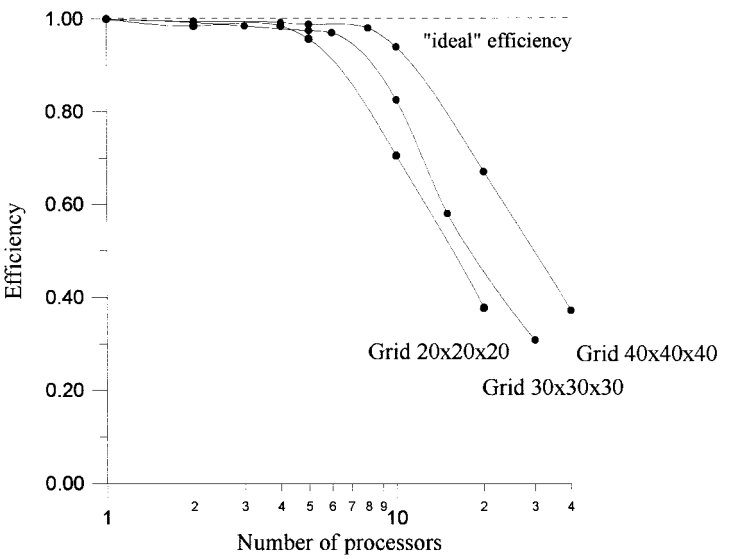

Figure 21. Efficiency obtained on the Cray T3E.

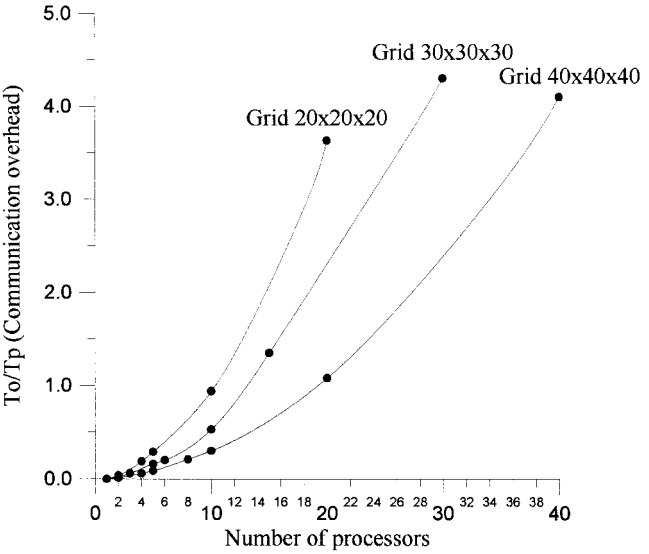

Figure 22. Communication cost.

In conclusion, the communication cost $c = T_o/T_p$ scales as $O(p/Nz)$ as well illustrated in Figure 22.

8. CONCLUSIONS

This paper describes a numerical solution strategy to solve the 3D incompressible Navier–Stokes equations for the study of three-dimensional Marangoni flows in axisymmetric geometries on parallel or serial computer systems.

The feasibility of this approach to perform transition studies of Marangoni flow in liquid bridges on these computers has been stated. The results indicate that the non-linear equations governing the behavior of Marangoni flows can effectively be parallelized on a distributed-memory parallel machine by remapping the distributed data structure.

A parallel method for solving the elliptic Navier–Stokes equations for incompressible fluid flows has been presented. A parallel, explicit in time algorithm for solving the parabolic equations and a parallel multisplitting elliptic kernel are described.

Numerical experiments and parallel performance measurements have been presented to illustrate the numerical properties and the parallel efficiency.

The parallel efficiency tests on the Cray T3E systems show that good scalability on a large number of processors can be achieved for both the multisplitting elliptic routine and the SMAC method as long as the granularity of the parallel application is not too small.

The communication timings indicate that a complex practical calculation, such as the accurate solution of the Navier–Stokes equations for the study of Marangoni flow instability, can be expected to run on a massively parallel machine with good efficiency.

Future studies will be dedicated to the assessment of two-dimensional (and eventually three-dimensional) partitions for the problem under consideration and to the analysis of the Marangoni flow instability for high Prandtl liquids.

ACKNOWLEDGMENTS

This work is part of the PhD thesis of M. Lappa. The authors would like to thank the Italian Aerospace Research Center (CIRA) that allowed the numerical calculations on the Convex 3880 Supercomputer and the Italian Interuniversity Computing Centre (CINECA) that allowed the numerical calculations on the Cray T3E massively parallel Supercomputer.

REFERENCES

1. C.H. Chun and W. West, 'Experiments on the transition from the steady to the oscillatory Marangoni convection of a floating zone under reduced gravity effect', *Acta Astronautica*, **6**, 1073–1082 (1979).
2. R. Monti, 'On the onset of the oscillatory regimes in Marangoni flows', *Acta Astronautica*, **15**, 557–560 (1987).
3. F. Preisser, D. Schwabe and A. Scharmann, 'Steady and oscillatory thermo capillary convection il liquid columns with free cylindrical surface', *J. Fluid Mech.*, **126**, 545–567 (1983).
4. N.D. Kazarinoff and J.S. Wilkowski, 'Bifurcations of numerically simulated thermocapillary flows in axially symmetric float-zones', *Phys. Fluids A*, **2**, 1797–1806 (1990).
5. Y. Zhang and J.I. Alexander, 'Surface tension and buoyancy driven flow in a non-isothermal liquid bridge', *Int. J. Numer. Methods Fluids*, **14**, 197–215 (1992).
6. R. Monti and R. Savino, 'Effect of unsteady thermal boundary conditions on Marangoni flow in liquid bridges', *Microgravity Q.*, **4**, 163–169 (1994).
7. V. Griaznov, 'CFD simulation of the oscillatory floating zone convection for High Prandtl numbers', *Proceedings of the Microgravity Science and Applications Session*, *International Aerospace Congress*, Moscow, August 16–17 1994.
8. Z.M. Tang, W.R. Hu and R Zhou, 'Fractal features of oscillatory convection in the half floating zone', *Int. J. Heat Mass Transfer*, **38**, 3295–3303 (1995).
9. J.J. Xu and S.H. Davis, 'Convective thermocapillary instabilities in liquid bridges', *Phys. Fluids*, **27**, 1102–1107 (1984).
10. Y. Shen, G.P. Neitzel, D.F. Jankowsky and H.D. Mittelmann, 'Energy stability of the thermocapillary convection in a model of the float zone crystal-growth process', *J. Fluid Mech.*, **217**, 639–660 (1990).
11. G.P. Neitzel, C.C. Law, D.F. Jancowski and H.D. Mittelmann, 'Energy stability of thermocapillary convection in a model of the float-zone crystal-growth process', *Phys. Fluids A*, **3**, 2841–2846 (1991).
12. G.P. Neitzel, K.T. Chang, D.F. Jancowski and H.D. Mittelmann, 'Linear stability of thermocapillary convection in a model of the float-zone crystal-growth process', *Phys. Fluids A*, **5**, 108–114 (1992).
13. H.C. Kuhlmann, H.J. Rath, 'Hydrodynamic instabilities in cylindrical thermocapillary liquid bridges', *J. Fluid Mech.*, **247**, 247–274 (1993).
14. H.C. Kuhlmann and H.J. Rath, 'On the interpretation of phase measurements of oscillatory thermocapillary convection in liquid bridges', *Phys. Fluids A*, **5** (9) 2117–2120 (1993).
15. M. Wanschura, V. Shevtsova, H.C. Kuhlmann and H.J. Rath, 'Convective instability mechanism in thermocapillary liquid bridges', *Phys. Fluids A*, **5**, 912–925 (1995).
16. R. Rupp, G. Muller, G. Neumann, 'Three-dimensional time dependent modelling of the Marangoni convection in zone melting configurations for GaAs', *J. Cryst. Growth*, **97**, 34–41 (1989).
17. M. Levenstam and G. Amberg, 'Hydrodynamical instabilities of thermocapillary flow in a half-zone', *J. Fluid Mech.*, **297**, 357–372 (1995).
18. R. Savino and R Monti, 'Oscillatory Marangoni convection in cylindrical liquid bridges', *Phys. Fluids*, **8**, 2906–2915 (1996).
19. R. Monti, R. Savino and M. Lappa, 'Oscillatory Thermocapillary flows in simulated floating zones with time-dependent boundary conditions' *Acta Astronautica*, **41**, 863–875, (1998).
20. R. Monti, R. Savino, M. Lappa and R. Fortezza, 'Scientific and technological aspects of a sounding rocket experiment on oscillatory Marangoni flow', *Space Forum*, **2**, 293–318 (1998).
21. C.A.J. Fletcher, *Computational Techniques for Fluid-Dynamics*, Springer, Berlin, 1991.
22. C. Hirsch, *Numerical Computations for Internal and External Flows*, Wiley, New York, 1994.
23. W. Schonauer and W. Gentzsch, 'The efficient Use of Vector Computers with Emphasis on Computational Fluid Dynamics', *Notes on Numerical Fluid Mechanics*, *12*, Vieweg, Berlin, 1986.
24. D.E. Keyes, T.F. Chan, G.A. Meurant, J.S. Scroggs and R.G. Voigt (eds.), *Proceedings of Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, SIAM, Philadelphia, 1992.
25. J. Häuser and R.D. Williams, 'Strategies for parallelizing a Navier–Stokes code on the Intel Touchstone machines', *Int. J. Num. Methods Fluids*, **15**, 51–58 (1992).
26. J.Z. Lou and R. Ferraro, 'A parallel incompressible flow solver package with a parallel multigrid elliptic kernel', *J. Comput. Phys.*, **125**, 225–243 (1996).
27. B. Smith, P. Bjorstad, W. Gropp, *Domain Decomposition*, *Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, UK, 1996.

28. W. Gropp, E. Lusk, A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, publication TPD-0011, MIT Press, USA, 1994.
29. *Message Passing Toolkit: MPI programmer's Manual*, Cray publication SR-2197, 1996.
30. D.P. O'Leary and R.E. White, 'Multisplitting of matrices and parallel solution of linear systems', *SIAM J. Alg. Disc. Methods*, **6**, 630–640 (1985).
31. R.A. Renaut and H.D. Mittelmann, 'Parallel multisplitting for optimization', *J. Parallel Alg. Appl.*, **7**, 17–27 (1995).

