ELSEVIER

Journal of Computational and Applied Mathematics 103 (1999) 77–92

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

Simulation of free surface flows in a distributed memory environment

José A. Cuminato[a,*], Antonio Castelo Filho[a], Maurilio Boaventura[b], Murilo F. Tomé[c]

[a] *ICMSC-USP, P.O. Box 668-13560-970, Sao Carlos, S.P., Brazil*

[b] *IBILCE-UNESP, P.O. Box 136-15054-000-S. J. do Rio Preto, S.P., Brazil*

[c] *Instituto Superior Técnico de Lisboa, Av. Rovisco Paes, 1096 Lisboa CODEX, Portugal*

Received 29 October 1997

Abstract

A parallel technique, for a distributed memory machine, based on domain decomposition for solving the Navier–Stokes equations in cartesian and cylindrical coordinates in two dimensions with free surfaces is described. It is based on the code by Tome and McKee (J. Comp. Phys. 110 (1994) 171–186) and Tome (Ph.D. Thesis, University of Strathclyde, Glasgow, 1993) which in turn is based on the SMAC method by Amsden and Harlow (Report LA-4370, Los Alamos Scientific Laboratory, 1971), which solves the Navier–Stokes equations in three steps: The momentum and Poisson equations and particle movement. These equations are discretized by explicit and 5-point finite differences. The parallelization is performed by splitting the computation domain into vertical panels and assigning each of these panels to a processor. All the computation can then be performed using nearest neighbour communication. Test runs comparing the performance of the parallel with the serial code, and a discussion of the load balancing question are presented. PVM is used for communication between processes.

Keywords: Free surface flow; Distributed memory; Parallel technique; Domain decomposition

1. Introduction

The marker-and-cell (MAC) method, introduced by Harlow and Welch [8] has been particularly designed for the modelling of fluid flows with free surfaces. Some of its important features are the superposition of a mesh of rectangular cells onto the flow domain and the use of virtual particles to represent the existing fluid. The particles move from one cell to another as a function of flow velocities, keeping track of the fluid location. If a cell contains a number of particles it is considered to contain fluid, providing the localization of the free surface. So in the MAC method the flow domain

* Corresponding author.

0377-0427/99/$ – see front matter
PII: S0377-0427(98)00242-8

is split into a number of cells which are labelled to be either empty, full, surface or boundary cells, and one has to update the labels as time evolves.

Amsden and Harlow [1] developed a simplified MAC method (SMAC), where the calculation cycle is split into two parts: a provisional velocity field calculation followed by a velocity update using a potential function which is the solution of Poisson's equation and ensures mass conservation. Tome and McKee [10] developed this technique further by adding a number of specially designed devices to deal with general domains and multiple inlets/outlets. They called it GENSMAC. This code has been successfully used to simulate different types of two-dimensional free surface flows, such as: jet filling of complex shaped cavities, penstock head-gate closure, jet buckling and die swell problems, see [10].

One of the main drawbacks in the implementation of either SMAC or GENSMAC methods is the storage space needed for the virtual particles, which can be quite large for very fine grids or problems requiring a fair amount of particles. Furthermore, moving those particles around at every computing cycle can take more than half the total computing time. In order to get round this difficulty, Tomé et al. [11] presented a new version of the GENSMAC method in which only the storage and movement of the particles sitting on the free surface is required. This brought a substantial improvement in both the storage space used by the code and the total computing time. This new version of GENSMAC includes also an extension which permits the simulation of three-dimensional axisymmetric flows. Axisymmetric problems are quite common in practical applications mainly in the chemical and food industries for the filling of molds.

As already mentioned, cutting down the number of particles representing the fluid was a big improvement on the previous methodologies, but for some problems it is not yet enough to bring the computing times within reasonable bounds. This may be achieved by employing parallel computation. The parallel algorithm to be described in this work is based on domain decomposition techniques and it will be implemented for both: the case of particles on the whole domain and on the free surface only. The reason for implementing both techniques lies in the fact that the free surface version is so involved with regard to the use of complex data structures, that we need to make sure its parallel version would keep the computing times gain presented by the serial version.

In this work we describe the method GENSMAC and the parallel algorithm which implements the domain decomposition technique. Both versions of the GENSMAC code with particles on the entire domain and on the free surface, will be considered. Issues like load balancing and efficiency will be addressed. Two numerical tests will be presented and comparison of the computing times of the parallel code with that of the serial code will be given.

2. The SMAC method in two dimensions

The SMAC method is based on a staggered grid finite differences discretization of the equations arising from the following procedure used to solve the Navier–Stokes equations.

Assume that at time t_0 the velocity field $\boldsymbol{u}(\boldsymbol{x}, t_0)$ be given. Calculate $\boldsymbol{u}(\boldsymbol{x}, t)$, $t = t_0 + \delta t$ from the following:

(1) Let $\tilde{p}(\boldsymbol{x}, t_0)$ be an arbitrary pressure field satisfying the correct pressure conditions on the free surface, see Eqs. (6) and (7).

(2) Calculate the intermediate velocity field $\tilde{u}(x,t)$ from

$$\frac{\partial \tilde{u}}{\partial t} = -\frac{\partial u^2}{\partial x} - \frac{\partial uv}{\partial y} - \frac{\partial \tilde{p}}{\partial x} + \frac{1}{\mathrm{Re}}\nabla^2 u + \frac{1}{\mathrm{Fr}^2} g_x, \tag{1}$$

$$\frac{\partial \tilde{v}}{\partial t} = -\frac{\partial uv}{\partial x} - \frac{\partial v^2}{\partial y} - \frac{\partial \tilde{p}}{\partial y} + \frac{1}{\mathrm{Re}}\nabla^2 v + \frac{1}{\mathrm{Fr}^2} g_y, \tag{2}$$

where $\mathrm{Re} = UL/v$ and $\mathrm{Fr} = U/\sqrt{Lg}$, with U and L typical velocity and length scales, g gravity and v viscosity.

(3) Solve Poisson's equation

$$\nabla^2 \psi = \nabla \cdot \tilde{u} \quad \text{in } \Omega.$$

(4) Update the velocities and pressure from

$$u(x,t) = \tilde{u}(x,t) - \nabla\psi(x,t), \tag{3}$$

$$p(x,t) = \tilde{p}(x,t) + \psi(x,t)/\delta t. \tag{4}$$

(5) Update the particles position by solving the ODEs

$$\mathrm{d}x/\mathrm{d}t = u, \qquad \mathrm{d}y/\mathrm{d}t = v. \tag{5}$$

A number of boundary conditions can be applied at the rigid boundaries, namely: no-slip, free-slip, prescribed inflow, prescribed outflow and continuative outflow. At the free surface the usual free stress conditions apply:

$$p - \frac{2}{\mathrm{Re}}\left[n_x n_x \frac{\partial u}{\partial x} + n_x n_y \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) + n_x n_y \frac{\partial v}{\partial y}\right] = 0, \tag{6}$$

$$n_x m_y \frac{\partial u}{\partial x} + n_x m_y \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) + n_x m_y \frac{\partial v}{\partial y} = 0, \tag{7}$$

where $\boldsymbol{n} = (n_x, n_y)$ is the outward unit normal vector to the surface and $\boldsymbol{m} = (m_x, m_y)$ is the tangential vector. For the Poisson equation the boundary conditions are of mixed type, at the free surface a homogeneous Dirichlet boundary condition applies and at the rigid boundary the condition is of Neuman type.

The grid is staggered with the subscripts i and j denoting the cell centre, i counts columns in the x-direction and j counts rows in the y-direction. The unknowns are located as displayed in Fig. 1.

The implementation of the SMAC method effected in GENSMAC has four main parts:

(1) The velocity solver based on explicit finite differences:

$$\begin{aligned}\frac{\tilde{u}^{n+1}_{i+1/2,j} - u^n_{i+1/2,j}}{\delta t} = {} & \frac{\tilde{p}_{i,j} - \tilde{p}_{i+1,j}}{\delta x} + \frac{u^n_{i+1/2,j-1/2} v^n_{i+1/2,j-1/2} - u^n_{i+1/2,j+1/2} v^n_{i+1/2,j+1/2}}{\delta y} \\ & + \frac{u^n_{i+1/2,j} u^n_{i-1/2,j} - u^n_{i+3/2,j} u^n_{i+1/2,j}}{\delta x} + \frac{1}{\mathrm{Re}}\left[\frac{u^n_{i+1/2,j+1} + u^n_{i+1/2,j-1} - 2u^n_{i+1/2,j}}{\delta y^2}\right. \\ & \left. - \frac{v^n_{i+1,j+1/2} - v^n_{i+1,j-1/2} - v^n_{i,j+1/2} + v^n_{i,j-1/2}}{\delta x \delta y}\right] + \frac{1}{\mathrm{Fr}^2} g_x,\end{aligned} \tag{8}$$

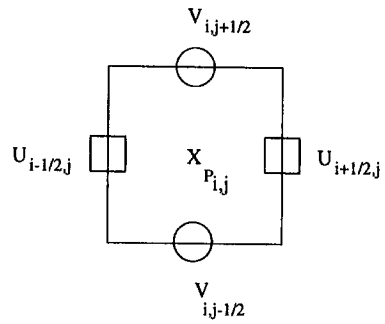

Fig. 1. Variable location in the staggered grid.

$$\frac{\tilde{v}_{i,j+1/2}^{n+1} - v_{i,j+1/2}^{n}}{\delta t} = \frac{\tilde{p}_{i,j} - \tilde{p}_{i,j+1}}{\delta y} + \frac{u_{i-1/2,j+1/2}^{n} v_{i-1/2,j+1/2}^{n} - u_{i+1/2,j+1/2}^{n} v_{i+1/2,j+1/2}^{n}}{\delta x} + \frac{v_{i,j+1/2}^{n} v_{i,j-1/2}^{n} - v_{i,j+3/2}^{n} v_{i,j+1/2}^{n}}{\delta y} - \frac{1}{\mathrm{Re}} \left[\frac{2v_{i,j+1/2}^{n} - v_{i+1,j+1/2}^{n} - v_{i-1,j+1/2}^{n}}{\delta x^2} + \frac{u_{i+1/2,j+1}^{n} - u_{i+1/2,j}^{n} - u_{i-1/2,j}^{n} + u_{i-1/2,j}^{n}}{\delta y \delta y} \right] + \frac{1}{\mathrm{Fr}^2} g_y. \tag{9}$$

(2) The Poisson solver Based on 5-point FD:

$$4\psi_{i,j} - \psi_{i+1,j} - \psi_{i-1,j} - \psi_{i,j+1} - \psi_{i,j-1} = -h^2 \tilde{D}_{i,j} \tag{10}$$

where $h = \delta x$ (assuming $\delta x = \delta y$) and

$$\tilde{D}_{i,j} = \frac{\tilde{u}_{i+1/2,j} - \tilde{u}_{i-1/2,j}}{\delta x} + \frac{\tilde{v}_{i,j+1/2} - \tilde{v}_{i,j-1/2}}{\delta y}.$$

(3) Update the velocity and pressure fields:

$$u(x,t) = \tilde{u}(x,t) - \nabla\psi(x,t), \tag{11}$$

$$p(x,t) = \tilde{p}(x,t) + \psi(x,t)/\delta t. \tag{12}$$

(4) Particle movement:

$$x^{n+1} = x^n + u\,\delta t, \tag{13}$$

$$y^{n+1} = y^n + v\,\delta t. \tag{14}$$

3. The SMAC method in cylindrical coordinates

The SMAC method in cylindrical coordinates for the solution of three-dimensional axisymmetric problems follows the same development of the above section. The equations are:

(1) The tilde velocity $\tilde{\boldsymbol{u}}(r,z,t)$ is calculated from:

$$\frac{\partial u}{\partial t} = \left[-\frac{1}{r}\frac{\partial (ru^2)}{\partial r} - \frac{\partial (uv)}{\partial z} - \frac{\partial \tilde{p}}{\partial r} + \frac{1}{\mathrm{Re}}\frac{\partial}{\partial z}\left(\frac{\partial u}{\partial z} - \frac{\partial v}{\partial r}\right) + \frac{1}{F_r^2} g_r \right]_{t=t_0}, \tag{15}$$

$$\frac{\partial v}{\partial t} = \left[-\frac{1}{r}\frac{\partial (ruv)}{\partial r} - \frac{\partial (v^2)}{\partial z} - \frac{\partial \tilde{p}}{\partial z} + \frac{1}{\mathrm{Re}}\frac{1}{r}\frac{\partial}{\partial r}\left(r\left(\frac{\partial u}{\partial z} - \frac{\partial v}{\partial r}\right)\right) + \frac{1}{F_r^2} g_z \right]_{t=t_0} \tag{16}$$

(2) The Poisson equation:

$$\nabla^2 \psi(r,z,t) = \nabla \cdot \tilde{\boldsymbol{u}}(r,z,t). \tag{17}$$

(3) The boundary conditions on the free surface:

$$p - \frac{2}{\mathrm{Re}}\left[\frac{\partial u}{\partial r} n_r^2 + \frac{\partial v}{\partial z} n_z^2 + \left(\frac{\partial u}{\partial z} + \frac{\partial v}{\partial r}\right) n_r n_z\right] = 0, \tag{18}$$

$$\frac{2}{\mathrm{Re}}\left[\left(\frac{\partial u}{\partial r} - \frac{\partial v}{\partial z}\right) n_r n_z + \left(\frac{\partial u}{\partial z} + \frac{\partial v}{\partial r}\right)(n_r^2 - n_z^2)\right] = 0. \tag{19}$$

4. The parallel technique

The basic idea is the use of parallel programming by message passing, that is, a number of serial processes exchanging information (data) through messages. The public domain package PVM, see [7], will be used for message passing. At a given time t the solution domain will resemble the one depicted in Fig. 2 below, where S denotes the cells forming the free surface, B denotes the rigid boundary and F denotes the fluid. The algorithm described in steps 1, 2 and 3 is to be applied to the cells marked with F. On the cells marked with S the velocities are updated using the free surface stress free conditions (6) and (7). On the rigid boundaries, inlet and outlet, the velocities are calculated from the appropriate boundary conditions applicable to each case.

The region containing all the cells is split into a number of vertical panels. These panels are then assigned to each of the available processors, which goes on to apply the velocity solver

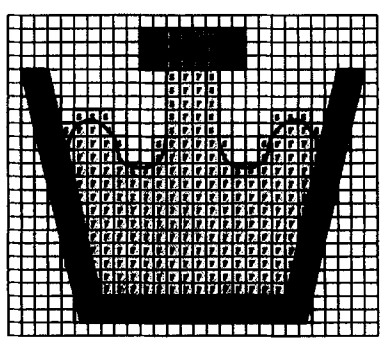

Fig. 2. Domain Ω and cells types as in GENSMAC.

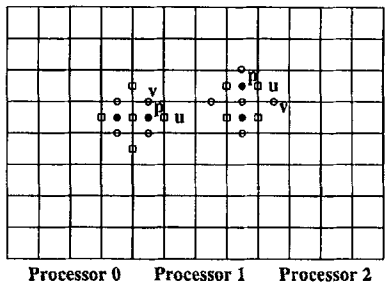

Fig. 3. Stencil for the momentum equations.

to its grid points which are full cells, see Fig. 2. For the explicit finite differences scheme this step presents no difficulties for the parallel algorithm. Fig. 3 shows the stencil for the momentum equations.

After having calculated the tilde velocities from Eqs. (8) and (9), each processor communicates the first and last columns of both u and v, see Fig. 3, to the right and left neighbour processors, respectively. Obviously the first and last processors communicate to only the last and the first columns, respectively. The next step is the solution of Poisson's equation through the 5-point finite difference scheme of step 2. This step is effected using the same processor topology used for the momentum equations. We use a parallel implementation of the conjugate gradient method presented in [3, 4] distributed in the package Parallel Iterative Methods (PIM), see [5]. PIM implements all the code necessary for the iteration of the CG method, apart from the routines for matrix–vector products and inner products, which are supplied by the user. In this context this is one of the major difficulties, because the matrix–vector products are quite involved and require careful coding. One step of the conjugate gradient solver requires the computation of the matrix–vector product $y = Ax$, where for a grid point (i,j) the corresponding $y_{i,j}$ is calculated from

$$y_{ij} = \alpha_{ij}x_{ij} + \beta_{ij}x_{i+1,j} + \gamma_{ij}x_{i-1,j} + \delta_{ij}x_{i,j+1} + \varepsilon_{ij}x_{i,j-1} \tag{20}$$

and $\alpha, \beta, \gamma, \delta$ and ε are the coefficients corresponding to the centre, north, south, east and west locations, respectively. These coefficients are derived from the PDE and boundary conditions.

The main difficulty in performing this matrix–vector product is associated with the domain being irregular. This is done in such a way that only points falling inside the region Ω will be included in the calculations. To achieve this, indirect addressing has to be used in order to cope with the contribution of the farthest points (east and west in our case because we are numbering the unknowns by columns) and to minimize communication, as we do not wish to communicate the coefficients of points which are not going to be used to form the product. For points inside Ω with neighbours not in Ω, the corresponding coefficient is set to zero. To implement the correct boundary conditions it is necessary at this stage to modify the coefficients and/or the right-hand side vector as appropriate. Note that the coefficients are formed only once, and are kept fixed during the iterations of the iterative method. This arrangement permits the use of nonuniform meshes. The generality of the geometry makes the next neighbour communication pattern much more involved than the case of a square region, because the length and positions of the vectors to be exchanged depend now on the geometry.

The parallel implementation of the particle movement is done likewise. Each processor having upgraded the velocity and pressure fields applies the ODE (5) to find the position of its particles at that time. Particle movement across processor boundaries will be necessary and will only involve next neighbour communication. The position of the fluid is updated with each processor communicating the possible particles which are to be moved across the boundaries. At the end of a computing cycle all processors hold the correct information for performing the next step. The algorithm for the cylindrical coordinates is exactly the same as that for the cartesian coordinates, considering the appropriate equation for each case.

5. A new technique for representing the free surface

The most time consuming parts of the GENSMAC code are the routines implementing virtual particle movement. They account for about 50% of the total work load. In addition, for relatively fine meshes the number of particles needed for representing the fluid properly is very large, requiring a large memory storage space. These two facts present a serious restriction in the use of GENSMAC for large problems and/or fine meshes. Recently, Tome et al. [11] developed a new version of GENSMAC, in which a new technique for representing the free surface is implemented. We shall call this version of the code *the free surface particle*. It permits the storage and movement of particles on the free surface only, as opposed to the previous version of GENSMAC (which we call here *the full particle*), where particles were used in the bulk of the fluid including the free surface.

In the sequential version of the free surface particle implementation, each body of fluid is represented as a oriented closed list, which records the position of the particles on the free surface. The orientation is crucial for determining where the fluid lies. Fig. 4 illustrates one of such lists.

As can be noted from the picture the initial point PI coincides with the last one PL. The arrows give the orientation of the list which in turn determines the position of the fluid. Fluid lies always on the left of the arrow's direction.

This data structure, unfortunately, does not carry over for the parallel implementation, for the domain decomposition may split each body of fluid into smaller pieces which will be assigned to different processors. Each of these pieces will possibly have to be continued in a neighbour

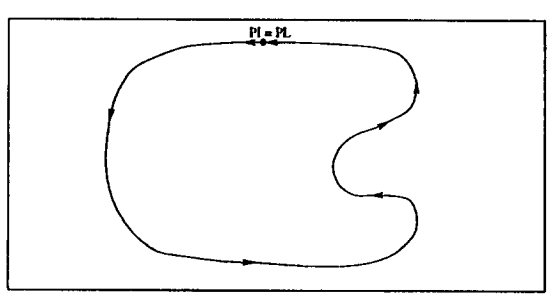

Fig. 4. Free surface representation as closed oriented list.

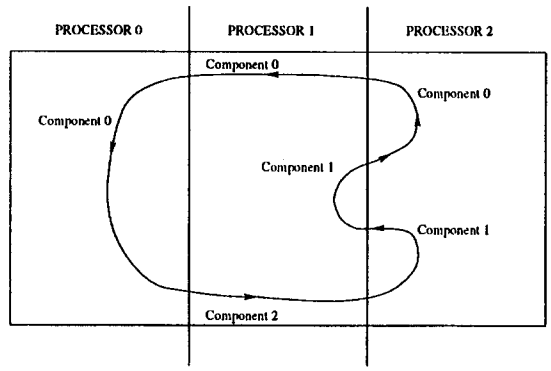

Fig. 5. Components of a list.

processor, so it cannot be a closed list for a start. We used an open oriented list instead which is composed of several components, each processor may hold one, more than one or even none of them. The following picture shows an example of such arrangement.

Fig. 5 illustrates the situation of a flow with one body of fluid separated in three processors. Note that processor 1 holds information on three components of the fluid: component 0 – starts at component 0 of processor 2 and ends at component 0 of processor 0; component 1 – starts at component 1 and ends at component 1 both in processor 2; and component 2 – starts at component 0 of processor 0 and ends at component 1 of processor 2. As time evolves the fluid geometry can vary considerably, so that the start, the end, a part or even the whole of a component can transfer from one processor to another. When this happens, information is exchanged between the involved processors so that the new situation is accommodated within the data structure and a new time step can begin. For instance, when the start of a component, together with some part of it, moves to a neighbouring processor, they are appended at the end of a component there, which corresponds to its continuation in the destination processor. When part of a component, which does not contain either the start or end of it, needs to be transferred to another processor, a new component is created at the destination, while in the original processor the donor component is broken into two new ones. It can also happen the situation where several components group together to form a unique new one. Figs. 6 and 7 illustrate some of those situations.

Consider the situation where on a given time instant the fluid has the shape depicted in Fig. 5, and as time evolves the fluid assumes the form shown in Fig. 6, that is, the start, the end and a central part of components 0 and 1 in processor 2 must be transferred to processor 1. In this case components 0 and 1 of processor 2, turn into components 0–2 and 1–3, respectively, of processor 2; while in processor 1 two new components are created, 3 and 4. The blank spaces shown in the pictures, between the components, are left purposely in order to make clear which components belonged to which processors. The next figure illustrates the situation where all components of a processor migrate to another. Processor 2 ceased holding fluid so that it holds no components of the list. On the other hand processor 1 used to hold 5 components which collapsed into only one.

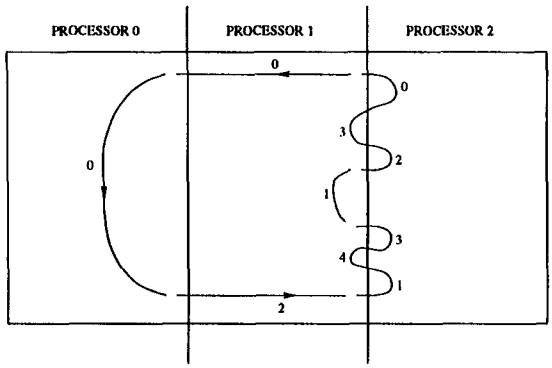

Fig. 6. Fluid movement.

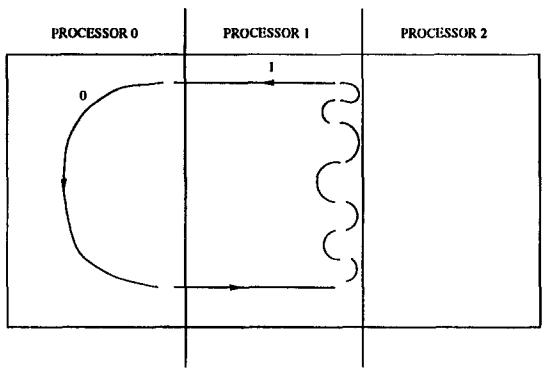

Fig. 7. Fluid movement.

The computer implementation of the data structure storing the free surface and its migration from one processor to another is relatively involved because the fluid can take almost any shape which is being modified at each time step. It has been implemented for the code GENSMAC in cartesian coordinates as well as in cylindrical coordinates.

6. Test problems

The sequential version of GENSMAC has been throughly tested throughout these years of use, including its validation against experiments [9]. The parallel version was tested in several examples, and two of them were chosen to be presented here. They were run in a parallel machine using a minimum of two processors and a maximum of eight.

Problem 1 (*Waterfall onto a step*). In this example the simulation of the flow of a water falling onto a round obstacle is presented. The fluid enters the container through an inflow starting at time t_0 falls onto the round obstacle under gravity and leaves through a continuative outflow. The boundary conditions on the container and obstacle walls are no-slip. The plots presented show

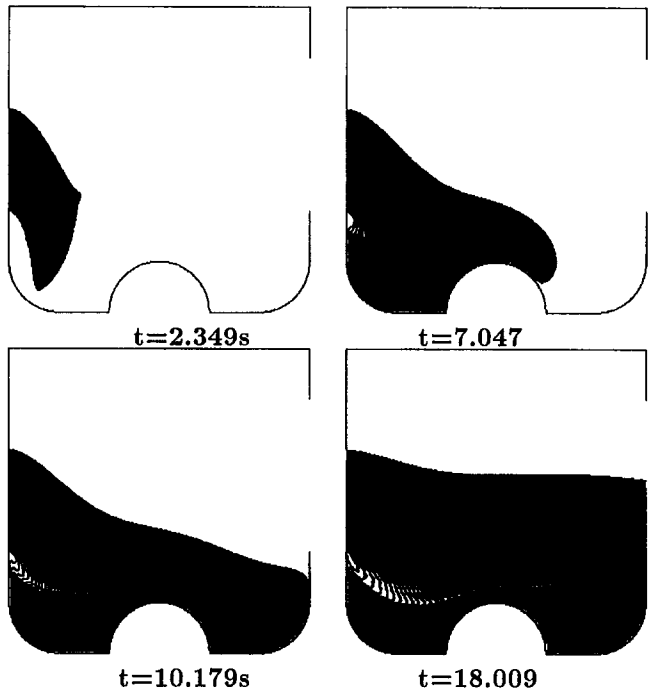

Fig. 8. Particle plots for the waterfall problem at different times.

particle configuration at different times. The Reynolds and Froude numbers for this example are: (Re = 10.0) and (Fr = 1.0).

Figs. 8 and 9 presents the numerical results for the same problem as above using the free surface particles version of the code, so that the fluid is represented by its free surface only.

Problem 2. This problem is concerned with an axisymmetric flow into a square box of side 5 cm which contains fluid. An inlet of diameter 5 mm is placed above this box and a coloured jet flows from it into the fluid in the box with velocity of 1 m/s. The fluid viscosity is $\nu = 0.005$, giving (Re = 10.0) and ($Fr^2 = 4.52$). The results shown below were produced by the code implementing the use of particles on the free surface only.

The examples presented above are classical test problems in the literature. The comparison between the numerical solution obtained by the methods used in this work with those in the literature were given in [9,10] and we have no intention of discussing them here again. The sole purpose of Fig. 10 is to illustrate the problems we are solving and not to discuss how good the solutions are. We shall concentrate the discussion on the performance of the parallel code instead.

7. Load balancing

An important aspect of parallel computing which needs consideration is load balancing. Balancing the work load fairly among the available processors is crucial for the good performance of the

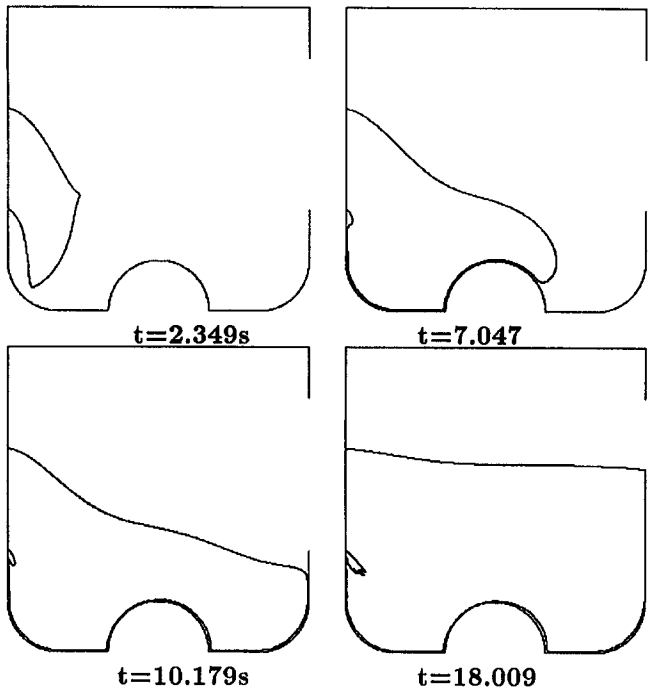

Fig. 9. Particle plots for the waterfall problem at different times.

parallel code. The ideal load balancing is achieved when all the available processors are assigned an equal share of the work. Nevertheless, this is not always easily accomplished, specially in the case of problems with free surfaces. The dynamics of load balancing presents serious difficulties in this case as the domain varies with time making the total work-load time dependent. The problem geometry can vary extensively with time and the region occupied by the fluid can be very small for a substantial time interval so that distributing the work-load will not be efficient. As time evolves this region increases and so does problem size so that it becomes worth having more processors cooperating.

The technique we have been using of splitting the domain into vertical strips of approximately the same width and assigning each of these strips to a processor, which will be responsible for the calculations on that strip throughout the whole computation, forces those processors holding parts of the domain where fluid has not yet arrived remain idle until fluid arrives to them. This is an inherent weakness of free surface problems and no parallel technique can avoid it.

8. Performance validation

Throughout this section we shall denote by T_p the time the parallel code using p processors takes to solve a given problem and by T_s the time the sequential code takes to solve the same problem.

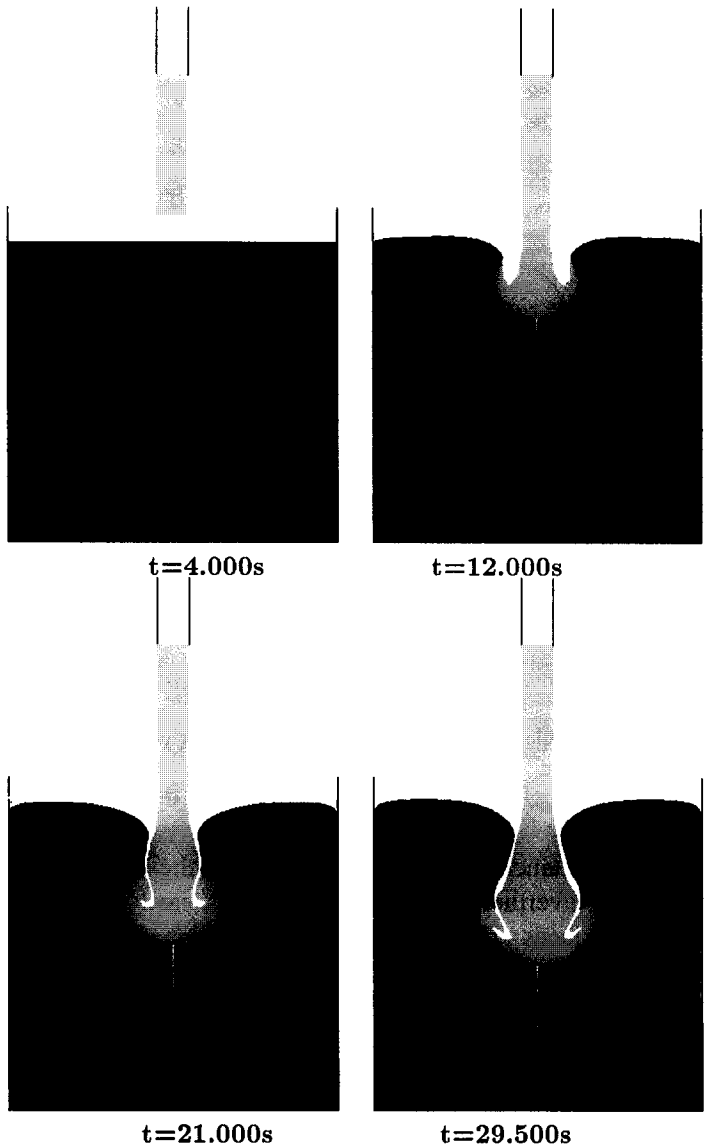

Fig. 10. Particle plots for jet flow experiment at different times.

The *Efficiency* of the parallel code is then defined by

$$E = T_p/pT_s.$$

Note that by this definition when the problem is *embarrassingly parallel* $T_p = T_s/p$ and then the efficiency is 1, this is the best one can hope for.

In this section we present the results of applying the parallel code for solving the problems of the previous section.

Table 1
Time in seconds

NEQ	T_s	T_2	T_4	T_6	T_8
500	226	290	297	272	262
2000	2570	2189	1750	1795	1593
3000	4976	3880	2839	2725	2597
8000	28 621	21 594	12 499	11 449	10 328
12 000	65 708	44 505	25 886	19 585	17 119

Table 2
Efficiency

NEQ	$T_s/2T_2$	$T_s/4T_4$	$T_s/6T_6$	$T_s/8T_8$
500	0.39	0.19	0.12	0.08
2000	0.58	0.38	0.24	0.20
3000	0.64	0.44	0.30	0.24
8000	0.66	0.57	0.42	0.35
12 000	0.73	0.63	0.56	0.48

Table 1 displays the time for the solution of Problem 1 using a serial code and a parallel code with 2, 4, 6 and 8 processors. The version of the code used is the free surface particles. This is a typical example of a transient problem, because in the beginning there is hardly any fluid in the container, and the work load is very low, making the use of several processors unjustified. As more fluid enters the container more calculation needs to be performed and the cooperation of several processors makes sense. The overall performance of the parallel code, however, is greatly affected by this initial phase . The purpose of the example is mainly to give the reader an idea of the sort of times we are talking about.

Table 2 displays the variation of code efficiency with problem size. The purpose of this example is to produce some evidence that the code becomes more efficient for large problems. It can be noted in Table 2 that efficiency is growing with problem size; that is, as problem size increases the work load becomes more evenly balanced among the processors. This brings the possibility of solving really large problems, provided that sufficient number of processors are available.

Tables 1 and 2 display the time (in seconds) and the efficiency for problem 1, in the particular situation when the maximum number of cells containing fluid, i.e., 12 000, is achieved. NPROCS is the number of processors and the full particles version of the code is being used. Note from Tables 3 and 4 that for this problem we have better efficiency. This happens because the number of particles to be moved around is quite large and this improves the computation to communication ratio. Nevertheless, the total time taken by both the sequential and parallel codes is around three times that of the same codes for the free surface particles version, showing that the free surface particles code is much faster.

The efficiency of the code is studied for Problem 2 using the free surface particles version of the code with two problem sizes: NEQ = 4000 and NEQ = 16 000. Note that for this problem the work load is well balanced among the processors from the start of the computation: all the processors have approximately the same work load. If the number of processors is increased, the work load of each

Table 3
Time in seconds and efficiency

NPROC	Time	Efficiency
1	198 018	1.00
2	121 740	0.81
4	75 009	0.66
6	55 302	0.60
8	41 555	0.60

Table 4
Efficiency for problem 2

NEQ	$T_s/2T_2$	$T_s/4T_4$	$T_s/6T_6$	$T_s/8T_8$
4000	0.78	0.58	0.48	0.45
16 000	0.83	0.73	0.70	0.63

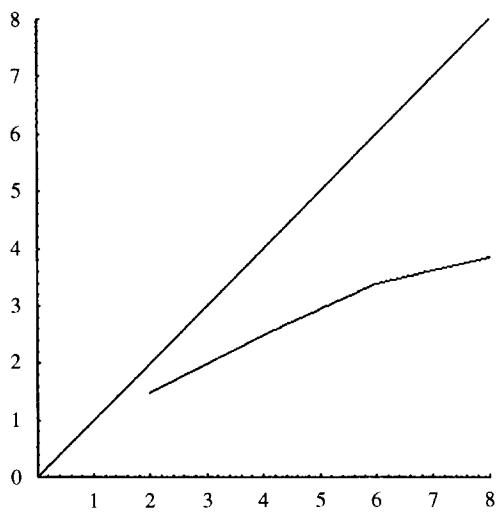

Fig. 11. Speed-up for Problem 1.

processor decreases, so that the computation to communication ratio decreases and consequently so does the efficiency. On the other hand if problem size is increased the opposite situation occurs and so efficiency is increased.

Figs. 11 and 12 illustrate the speed-up obtained by the parallel code in the solution of Problems 1 and 2 above with the maximum number of cells allowed (12 000 for Problem 1 and 16 000 for Problem 2). It can be observed that again the performance of the parallel code is rather good for truly large problems, where the use of parallel processing can be justified. The decrease in efficiency with increasing number of processors is expected due to the very nature of the parallel technique being employed.

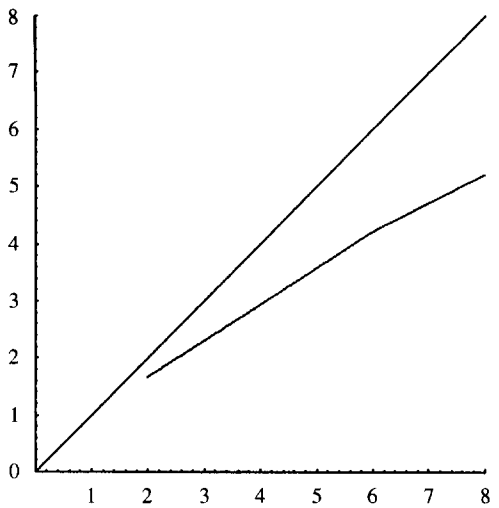

Fig. 12. Speed-up for Problem 2.

9. Conclusions

A parallel implementation of the SMAC method is presented. This implementation is based on the code GENSMAC which has been successfully tested with a variety of problems. The parallel code is based on domain decomposition and is implemented using PVM, a public domain software that permits the use of cluster of workstations as a parallel computer. The tests showed that the parallel implementation can harness the power of the cluster cutting running times to at least half that of the serial code. However, as the tests showed, one cannot hope to increase the number of processes indefinitely, for a fixed sized problem. This is a characteristic inherent to the type of problem we are dealing with, where the solution domain varies with time, starting from a very small region and growing as time evolves.

Acknowledgements

This work is part of an on-going project supported by FAPESP and CNPq under grants 94/6019-2 and 523141/94-7, respectively. The author Maurílio Boaventura is supported by CAPES under a Ph.D. grant. The code is implemented in C using PVM for message passing and the run tests were performed on a IBM/SP2 with eight processors, available at CENAPAD-SP, at Campinas University (UNICAMP). The computer was not dedicated to our use and the network was shared with other applications.

References

[1] A.A. Amsden, F.H. Harlow, The SMAC method: a numerical technique for calculating incompressible fluid flows, Los Alamos Scientific Laboratory, Report LA-4370, 1971.

[2] J.A. Cuminato et al., Parallel solution of free surface flows, in: VI Congresso Brasileiro de Engenharia e Ciências Térmicas, VI Congreso Latinoamericano de Tranferencia de Calor y Materia, Proceedings, Florianópolis, vol. 6, 1996, pp. 583–588.
[3] R.D. Cunha, A study of iterative methods for the solution of systems of linear equations on transputer networks, Ph.D. Thesis, University of Kent at Canterbury, UK, 1992.
[4] R.D. Cunha, T. Hopkins, The parallel solution of linear equations using iterative methods on transputer networks, Technical Report 16/92, University of Kent at Canterbury, UK, 1992a.
[5] R.D. Cunha, T. Hopkins, PIM 1.1, The parallel iterative package for systems of linear equation, User's guide, Technical Report, University of Kent at Canterbury, UK, 1993.
[6] M.O. Deville, Numerical experiments on then MAC Code for a slow flow, J. Comput. Phys. 15, 13 (1974) 362–374.
[7] A. Geist et al., PVM3 user's Guide and Reference Manual, Oak Ridge, National Laboratory, Oak Ridge, TN, 1994.
[8] F. Harlow, J.E. Welch, Numerical calculation of time-dependent viscous incompressible flow of fluid with free surfaces, Phys. Fluids 8 (1965) 2182–2189.
[9] M.F. Tome, GENSMAC: A multiple free surface fluid flow solver, Ph.D. Thesis, University of Strathclyde, Glasgow, Scotland, 1993.
[10] M.F. Tome, S. McKee, GENSMAC: A computational marker and cell method for free surface flows in general domains, J. Comput. Phys. 110 (1994) 171–186.
[11] M.F. Tomé et al., Numerical simulation of axisymmetric free surface flows, Notas do ICMSC-USP, No. 30, São Carlos, 1996.
[12] J.A. Viecelli, A method for including arbitrary external boundaries in the MAC incompressible fluid computing technique, J. Comput. Phys. 4 (1969) 543–551.