Simulation of Particulate Flows on Multi-Processor Machines with Distributed Memory

M. Uhlmann

Dept. Combustibles Fósiles, CIEMAT
Avda. Complutense 22, 28040 Madrid (Spain)
markus.uhlmann@ciemat.es

May 2003

Simulation of Particulate Flows on Multi-Processor Machines with Distributed Memory

Uhlmann, M. (CIEMAT)
32 pp. 12 figs. 9 refs.

Abstract:

We present a method for the parallelization of an immersed boundary algorithm for particulate flows using the MPI standard of communication. The treatment of the fluid phase uses the domain decomposition technique over a Cartesian processor grid. The solution of the Helmholtz problem is approximately factorized and relies upon a parallel tri-diagonal solver; the Poisson problem is solved by means of a parallel multi-grid technique similar to MUDPACK. For the solid phase we employ a master-slaves technique where one processor handles all the particles contained in its Eulerian fluid sub-domain and zero or more neighbor processors collaborate in the computation of particle-related quantities whenever a particle position over-laps the boundary of a sub-domain. The parallel efficiency for some preliminary computations is presented.

Simulación de Flujos con Partículas en Ordenadores Parallélos con Memoria Distribuida

Uhlmann, M. (CIEMAT)
32 pp. 12 figs. 9 refs.

Resumen:

Se propone un método para la computación paralela de un algoritmo de fronteras embebidas para flujos con partículas, utilizando el estandard de comunicación MPI. Se aplica la técnica de *domain decomposition* a la fase fluida: factorización aproximada para los problemas de Helmholtz, resolviendo las matrices tri-diagonales en paralelo; un esquema *multi-grid* paralelo para el problema de Poisson. Se utiliza una técnica maestro-esclavo para la fase solida: un procesador trata las partículas que están localizadas dentro de su sub-dominio y cero o más procesadores vecinos colaboran cuando la posición solapa la frontera de un sub-dominio. Se presenta la eficiencia paralela en algunos cálculos preliminares.

Contents

Chapter 1

Introduction

The study of particulate flow systems of scientific or engineering interest typically involves a large number of solid particles and a wide range of scales of the flow field. The combination of both factors means that feasible computations will be severely limited by current computing power. On the other hand, every effort should be made towards an efficient use of the available resources. This leads us immediately to the topic of parallel computing.

Virtually all of today's super-computers are parallel machines. In many cases, the memory is not shared but distributed across the processors or across various nodes of multi-processors. In order to obtain a portable code, it is therefore quite desirable to design a program for distributed-memory systems and, particularly, to resort to the widely available MPI standard of data communication.

In this framework, i.e. distributed memory multi-processor machines, there are quite a few open questions concerning the strategy to be employed for particle simulations. Most of them are related to issues of load-balancing due to unequal distribution of particles across processors as well as the communication overhead produced by shared processing of particles and hand-over of particles between different processing units. In the present study we try to gather some first experiences from the practical implementation of an algorithm for particulate flow simulation. We do not pretend to solve the above mentioned questions optimally. Rather will we report our chosen approach and the results concerning efficiency.

The algorithm which we consider pertains to two spatial dimensions and was previously detailed in [1]. It could be argued that the parallelization of a 2D code is not very useful as such. This is not necessarily true. Firstly, our results show that reasonable parallel efficiency can be obtained for typical computations and employing somewhat "coarse-grained" parallelism. Furthermore, this study should really be seen as a step towards three-dimensional computations. The latter will definitely benefit from parallelism in a more significant way than the present ones. The conclusions drawn from the present study should be widely valid in the 3D case which is not really much more than a straightforward extension of the present case, albeit adding a much higher computational cost.

Chapter 2

Fluid phase parallelism

2.1 Basic projection method and spatial discretization

The fractional-step method used for advancing the incompressible Navier-Stokes equations in time has been detailed in [1]; it shall be reproduced below for convenience. Using a semi-implicit scheme for the viscous terms and a three-step, low-storage, self-restarting Runge-Kutta method with explicit non-linear terms, the semi-discrete system can be written as follows:

$$\begin{aligned}
\frac{\mathbf{u}^* - \mathbf{u}^{k-1}}{\Delta t} &= -2\alpha_k \nabla p^{k-1} + \alpha_k \nu \nabla^2 \left(\mathbf{u}^* + \mathbf{u}^{k-1}\right) \\
&\quad -\gamma_k \left[(\mathbf{u}\cdot\nabla)\mathbf{u}\right]^{k-1} - \zeta_k \left[(\mathbf{u}\cdot\nabla)\mathbf{u}\right]^{k-2} + \gamma_k \mathbf{f}^k + \zeta_k \mathbf{f}^{k-1} && (2.1a) \\
\nabla^2 \phi^k &= \frac{\nabla\cdot\mathbf{u}^*}{2\alpha_k \Delta t} && (2.1b) \\
\mathbf{u}^k &= \mathbf{u}^* - 2\alpha_k \Delta t \nabla \phi^k && (2.1c) \\
p^k &= p^{k-1} + \phi^k - \alpha_k \Delta t\, \nu \nabla^2 \phi^k && (2.1d)
\end{aligned}$$

Here $\mathbf{u}$ is the fluid velocity vector, p the pressure divided by the fluid density and ν the kinematic viscosity. The volume force $\mathbf{f}$ regroups the body forces arising from the solid-fluid coupling. Furthermore, $k = 1, 2, 3$ is the Runge-Kutta step count (with the level $k = 3$ being equivalent to $n + 1$), u^* the predicted, intermediate velocity and the intermediate variable ϕ is sometimes called “pseudo-pressure”. The set of coefficients α_k, γ_k, ζ_k leading to overall second-order temporal accuracy for both velocity and pressure was given in [1].

A uniform staggered grid is defined and the equations are discretized in space by standard central second-order finite-differences.

2.2 Parallel data structure

We distribute the fluid data defined on the fixed Eulerian grid over a Cartesian processor grid. For this purpose we arrange the available processors in a two-dimensional array with `nxprocs` columns and `nyprocs` rows such that their total number is given by the product `nprocs` = `nxprocs`·`nyprocs`. Each processor is designated a rank `myid` from MPI with `myid` ∈ [0, `nprocs-1`]. Likewise, each processor carries a column- and a row-index, `my_col` ∈ [0, `nxprocs-1`] and `my_row` ∈ [0, `nyprocs-1`], respectively, which uniquely identifies its position within the processor grid.

The linear grid dimensions are then each distributed over the corresponding processor grid dimensions. Let n_{x1} be a number of grid points in the first coordinate direction. Then the following algorithm generates the pointers *ibeg*, *iend* which give the index range of nodes held locally by each processor in a given column `my_col`:

Require: n_{x1}, `my_col`, `nxprocs`

```
$n =$ int $(n_{x1}/\text{nxprocs})$
$d =$ mod $(n_{x1}, \text{nxprocs})$
$ibeg(\texttt{my\_col}) = \texttt{my\_col} \cdot n + 1 + \min(\texttt{my\_col}, d)$
**if** $\texttt{my\_col} < d$ **then**
  $n \leftarrow n + 1$
**end if**
$iend(\texttt{my\_col}) = ibeg(\texttt{my\_col}) + n - 1$
**if** $\texttt{my\_col} = \texttt{nxprocs} - 1 \quad || \quad iend(\texttt{my\_col}) > n_{x1}$ **then**
  $iend(\texttt{my\_col}) = n_{x1}$
**end if**
```

It should be noted that the above algorithm leads to a distribution as even as possible (with a difference of only one in the number of indices held by each processor). As an example, 25 data are distributed to 4 processors as $7:6:6:6$, whereas under the block distribution employed by ScaLAPACK [2] we would get $7:7:7:4$.

Due to the use of a staggered grid we have unequal global dimensions for the different variables:

$$\begin{aligned} u: &\quad (1:n_{xu}, 1:n_{yu}) \\ v: &\quad (1:n_{xv}, 1:n_{yv}) \\ p, \phi: &\quad (1:n_{xp}, 1:n_{yp}) \end{aligned} \tag{2.2}$$

with

$$n_{xu} = n_{xp} + 1,\ n_{yu} = n_{yp},\ n_{xv} = n_{xp},\ n_{yv} = n_{yp} + 1. \tag{2.3}$$

Therefore, we need a total of 12 pointer arrays to local data (3 grid types, 2 spatial directions, 2 for start/end):

$$\begin{array}{lll} ibegu(0:\texttt{nxprocs}-1), & ibegv(0:\texttt{nxprocs}-1), & ibegp(0:\texttt{nxprocs}-1), \\ iendu(0:\texttt{nxprocs}-1), & iendv(0:\texttt{nxprocs}-1), & iendp(0:\texttt{nxprocs}-1), \\ jbegu(0:\texttt{nyprocs}-1), & jbegv(0:\texttt{nyprocs}-1), & jbegp(0:\texttt{nyprocs}-1), \\ jendu(0:\texttt{nyprocs}-1), & iendv(0:\texttt{nyprocs}-1), & jendp(0:\texttt{nyprocs}-1). \end{array} \tag{2.4}$$

Finally, local arrays are dimensioned by adding one "ghost cell" at each extremity of the local index range (cf. figure 2.1 for a schematic representation of the data distribution), e.g.

$$u_{local}: \quad (ibegu(\texttt{my\_col})-1 : iendu(\texttt{my\_col})+1, jbegu(\texttt{my\_row})-1 : jendu(\texttt{my\_row})+1), \tag{2.5}$$

and analogously for the other variables. This single overlap of data is enough to evaluate all operators locally when central second-order finite-differences are used. The necessary communication then consists in exchanging values of the "ghost cells" between neighbor processors at various stages during the algorithm, particularly after any update (cf. § 2.6 below).

2.3 Communication of "ghost cell" data

The method of exchanging "ghost cell" data between neighbors of a Cartesian processor grid is taken from reference [3] and shall be explained below.

First, a pointer array $neighbor(1:8)$ is defined locally. It holds the ranks of the 8 neighbor processors in a clock-wise numbering scheme as shown in figure 2.2. For processors which touch the extremities of the processor grid, some (or all) of these ranks might be undefined (say: set to a negative value) except when periodicity is specifically imposed, in which case they wrap-around. Analogously, define a local array of flags $ibd(1:8)$ which, when true, allow a processor to communicate with the respective neighbor (when $ibd(i)$ true, $neighbor(i)$ needs to indicate a valid rank). Hereby, boundary conditions like Dirichlet or Neumann (no communication) and periodicity (communication) can be distinguished for "extreme" processors.

The data to be exchanged in order to fill one processor's "ghost cells" is the following:

(i) one contiguous line each with neighbors 3,7

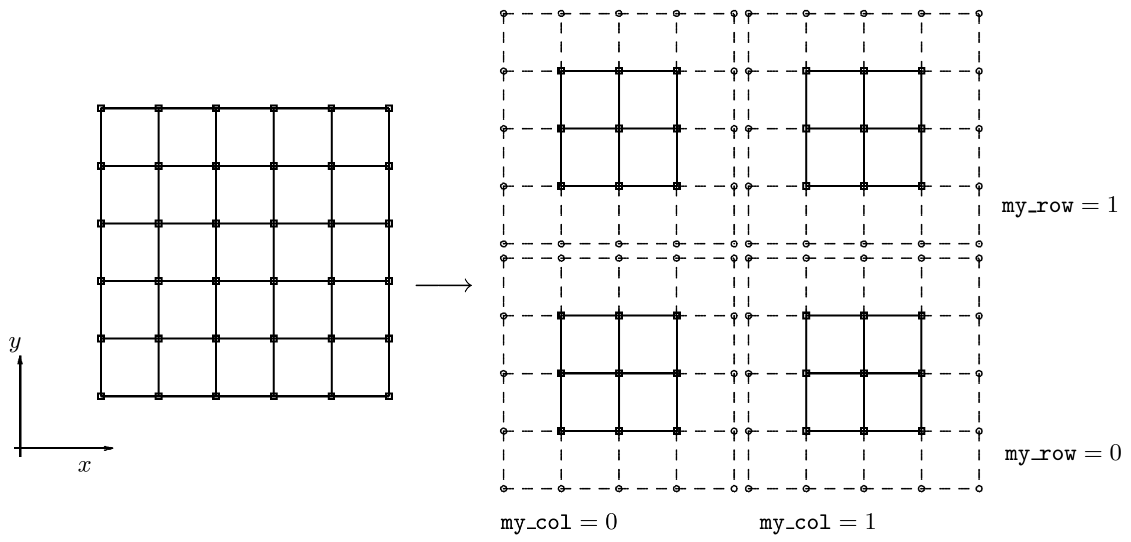


Figure 2.1: Parallel data structure: Schematic of the decomposition of a two-dimensional Cartesian *spatial* grid with dimension `nx` = 6, `ny` = 6 upon a two-dimensional Cartesian *processor* grid with `nxprocs` = 2 and `nyprocs` = 2, i.e. 4 processors in total. Node points are indicated by □ and "ghost" points by ∘. The pointers `my_col` and `my_row` indicate the position of each processor within the processor grid.

6	7	8
5	`myid`	1
4	3	2

Figure 2.2: Definition of neighbor processors in two-dimensional Cartesian processor grid.

(ii) one non-contiguous column each with neighbors 5,1

(iii) one datum each with neighbors 2,4,6,8

The global data volume to be communicated at one instance of an exchange of “ghost cell” data is the following:

$$\begin{aligned} D_V \;&=\; 2\left\{\texttt{nxprocs}\cdot\texttt{nyprocs}\,(2S_{LR}+2S_{TB}+4S_C)\right. \\ &\left. -4\,(S_C+S_{LR}+S_{TB}) - \max(\texttt{nxprocs}-2,0)\cdot 2S_{TB} - \max(\texttt{nyprocs}-2,0)\cdot 2S_{LR}\right\}, \end{aligned} \tag{2.6}$$

where

$$S_{LR} = iend(\texttt{my\_col}) - ibeg(\texttt{my\_col}) + 1, \quad S_{TB} = jend(\texttt{my\_row}) - jbeg(\texttt{my\_row}) + 1, \quad S_C = 1, \tag{2.7}$$

with *iend* etc. placeholders for the actual *iendu* etc. The second row in (2.6) is due to non-periodicity at the boundaries of the domain. For a case which is fully periodic and where the dimensions divide evenly by the number of processors we obtain:

$$D_V = 4\,\left(2\cdot\texttt{nxprocs}\cdot\texttt{nyprocs} + \texttt{nxprocs}\cdot n_y + \texttt{nyprocs}\cdot n_x\right). \tag{2.8}$$

Please note that the global factor of two in (2.8) reflects the send-and-receive character of the exchange. The global number of messages to be exchanged is the following:

$$N_M = 2\cdot\texttt{nxprocs}\cdot\texttt{nyprocs}\cdot 8 \tag{2.9}$$

in the fully periodic (worst) case.

For each of the three operations (i)-(iii) appropriate data types can be defined by means of calls to `MPI_TYPE_CONTIGUOUS` and `MPI_TYPE_VECTOR`, avoiding the need for temporary storage. The following algorithm then performs a non-blocking exchange of the ghost-cell data:

```
Require: $ibd$, $neighbor$, iseq_send, iseq_recv
  ireq=0
  for $ii = 1:8$ do
    $idest$ = iseq_send($ii$)
    if $ibd(idest)$ then
      $ireq \leftarrow ireq + 1$
      $rank\_dest = neighbor(idest)$
      call MPI_ISEND(...,$rank\_dest$,...)
    end if
    $iorig$ = iseq_recv($ii$)
    if $ibd(iorig)$ then
      $ireq \leftarrow ireq + 1$
      $rank\_orig = neighbor(iorig)$
      call MPI_IRECV(...,$rank\_orig$,...)
    end if
  end for
  call MPI_WAITALL(ireq,...)
```

Here the order of the exchange is determined through the following definition of sequences:

$$\texttt{iseq\_send} = \{1,3,2,4,5,7,6,8\}, \quad \texttt{iseq\_recv} = \{5,7,6,8,1,3,2,4\}. \tag{2.10}$$

The data types and the indices to the local array to be sent/received need to be adjusted in correspondence with the above sequences. These details were omitted from the above algorithm.

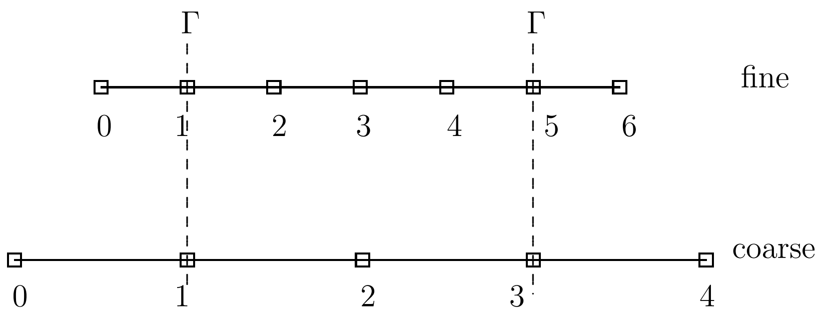


Figure 2.3: Two levels of refinement of the geometric multi-grid method with vertex-centered arrangement. Note that the two extreme points are ghost-cells, i.e. the proper fine grid consists of 2^2+1 nodes and the coarse one of 2^1+1 nodes.

2.4 Parallel Poisson solver: multi-grid

The inversion of the Poisson equation for pseudo-pressure (2.1b) was performed by means of a direct method in the single-processor version of the present code. More specifically, a cyclic reduction technique was used (cf. [1]). From a quick scan of the literature it seems that this type of algorithm is difficult to parallelize efficiently [4]. Instead we opted for an iterative multi-grid solution method for our present purpose. The desired accuracy of the solution (and consequently the accuracy with which zero-divergence is imposed) can then be chosen as a compromise with the numerical effort, as opposed to machine accuracy which is obtained by the direct method.

The present multi-grid variant is equivalent to the popular package MUDPACK [5], except that it is implemented for distributed multi-processor machines by means of MPI. The set of routines we use is actually based upon MGD [3]. The characteristics of the multi-grid algorithm can be summarized as follows:

- full-weighted residuals for restriction from fine to coarse grid;
- area-weighting of residuals for injection from coarse to fine grid;
- Gauss-Seidel point relaxation with red-black ordering for smoothing of residuals;
- use of V-cycles.

The parallelism was kept "as is", but the underlying solver was modified such as to allow for Neumann boundary conditions imposed at the node-points. Therefore, the discretization was changed to a vertex-centered arrangement as shown in figure 2.3. Since the pressure-nodes coincide with the boundary Γ in our present staggered grid arrangement, we are now able to handle the zero-gradient boundary condition for pseudo-pressure used for solving equation (2.1b).

When using this scheme, the global number of grid points for pressure is defined as follows:

$$n_{xp} = ixp \cdot 2^{(iex-1)} + 1, \tag{2.11a}$$

$$n_{yp} = jyq \cdot 2^{(jey-1)} + 1, \tag{2.11b}$$

and the dimensions for the two velocity components n_{xu},n_{yu}, n_{xv}, n_{yv} follow from (2.3). The exponents iex and jey should be maximized for a given problem since they determine how many levels of the grid hierarchy are used in each direction and therefore directly influence the convergence properties. On the other hand, the factors ixp and jyq must be chosen such that each processor will have at least a single grid point in each coordinate direction at the finest grid level, viz.

$$ixp \geq \texttt{nxprocs}, \quad jyq \geq \texttt{nyprocs}. \tag{2.12}$$

It should be noted that load-balancing is somewhat compromised in our present version due to the additional "singleton" dimension (i.e. the "+1" in equations 2.11). Thereby one processor always holds an additional grid point.

The necessary communication for a solution of a Poisson problem can be summed up as follows. Let us suppose for simplicity that $n_{xp} = n_{yp}$. The grid at refinement level k then has the size $ny(k) = nx(k) = ixp \cdot 2^{k-1} + 1$. One interchange of boundary data for the grid at level k consists of the data volume and number of messages of § 2.3 (i.e. formulas 2.6, 2.9), but calculated for the grid size $nx(k)$, $ny(k)$. Let us call these quantities $D_V(k)$ and $N_M(k)$. Inspecting the current multi-grid algorithm we find for the total data volume (D_{MGD}) and the total number of messages to be sent (N_{MGD}):

$$D_{MGD} = n_{it} \left(\sum_{k=iex}^{2} 5D_V(k) + 2D_V(1) + \sum_{k=1}^{iex} 2D_V(k) \right) , \tag{2.13a}$$

$$N_{MGD} = n_{it} \left(\sum_{k=iex}^{2} 5N_M(k) + 2N_M(1) + \sum_{k=1}^{iex} 2N_M(k) \right) , \tag{2.13b}$$

where n_{it} refers to the number of iterations needed to achieve the desired residual. In our experience, this number was around 6.

The performance of the multi-grid method is shown in figure 4.1 in form of the parallel efficiency:

$$E = \frac{T_1}{p \cdot T_p} , \tag{2.14}$$

where T_n is the execution time on n processors. As can be expected, the efficiency decreases for large numbers of processors at a given problem size.

2.5 Parallel Helmholtz solver: ADI

The Helmholtz problems for the predicted velocities (2.1a) could in principle be solved by a technique very similar to the multi-grid method above. However, due to the staggered grid, the required factorization of the dimensions (2.11) cannot be met simultaneously for the velocity components and pressure. Furthermore, it is not necessary to solve the Helmholtz equations up to very high accuracy due to the truncation error of the underlying scheme. Therefore, we opt for an approximate factorization method which has the advantage to decouple the spatial dimensions and further reduce the computational complexity. The present choice is a factorization with second order temporal accuracy, thereby keeping the overall order of the numerical method. Details are presented in appendix A.

In terms of parallelism, the present data structure implies that for an x (y) sweep, a row (column) of processors solves tri-diagonal linear systems in parallel, independently of the other rows (columns). More specifically, an x sweep means that each column of processors (i.e. the `nxprocs` processors with the same value of `my_row`) jointly solves a number of $jbeg$(`my_row`) : $jend$(`my_row`) tri-diagonal systems in the x direction. Communication is only needed within the respective column of processors. Therefore, MPI communicators for each column and each row are defined.

The task then is to parallelize the solution of a tri-diagonal linear system over a number of np processors, where np will in reality either be `nxprocs` or `nyprocs`. We employ the algorithm of Mattor *et al.* [6], which—in analogy with ordinary differential equations—splits up the problem into a homogeneous and a particular one. The homogeneous problem can be precomputed and the parallel over-head is restricted to the solution of a small algebraic system and a relatively limited amount of communication. Details of the algorithm are given in appendix B.

Keeping in mind the communication count of the parallel tri-diagonal solver and considering the structure of the factorization algorithm, the following data volume D_{ADI} and message count per processor N_{ADI} is obtained (using the notation of § 2.3):

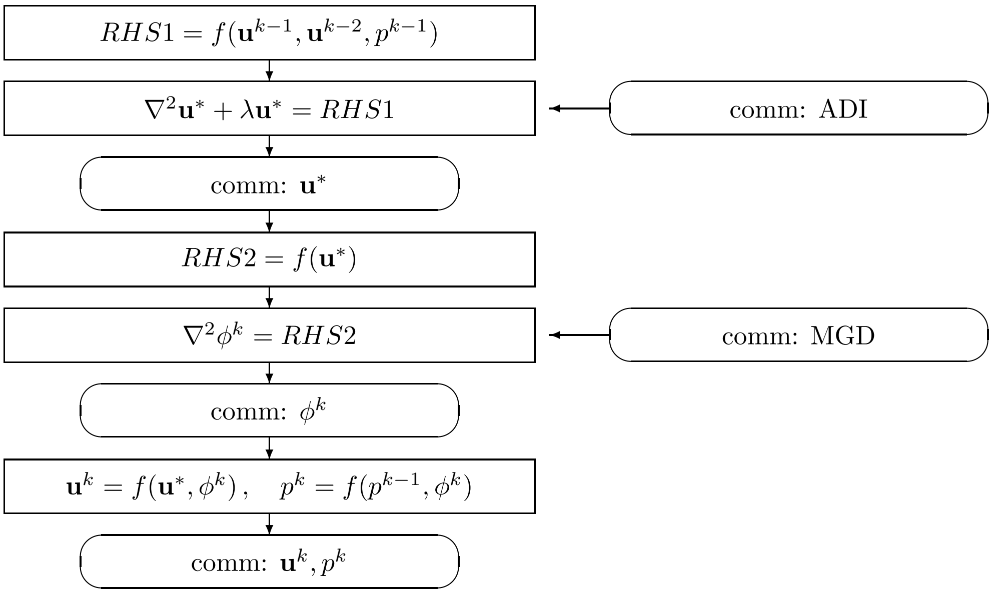
$$\begin{aligned} D_{ADI} &= S_{LR}\,\log_2(\texttt{nyprocs}) + S_{TB}\,\log_2(\texttt{nxprocs})\,, && \text{(2.15a)} \\ N_{ADI} &= 8S_{LR} \sum_{i=1}^{\log_2(\texttt{nyprocs})} 2^{i-1} + 8S_{TB} \sum_{j=1}^{\log_2(\texttt{nxprocs})} 2^{j-1}\,. && \text{(2.15b)} \end{aligned}$$

The above numbers refer to one solution of a Helmholtz problem with the ADI method; the algorithm (2.1) contains one such for each velocity component at each step.

The performance of the factorization method is shown in figure 4.2 in form of the parallel efficiency.

2.6 Instances of communication

The following flowchart shows at which point of a fluid step communication occurs.



$RHS1 = f(\mathbf{u}^{k-1}, \mathbf{u}^{k-2}, p^{k-1})$
$\nabla^2\mathbf{u}^* + \lambda\mathbf{u}^* = RHS1$
comm: ADI
comm: $\mathbf{u}^*$
$RHS2 = f(\mathbf{u}^*)$
$\nabla^2\phi^k = RHS2$
comm: MGD
comm: ϕ^k
$\mathbf{u}^k = f(\mathbf{u}^*, \phi^k)\,, \quad p^k = f(p^{k-1}, \phi^k)$
comm: $\mathbf{u}^k, p^k$

It can be seen that—besides the necessary communication during the solution of Helmholtz and Poisson problems—there are two exchanges of boundary data for each velocity component and two for pressure-size arrays (i.e. one for pseudo-pressure ϕ^k and one for the final pressure p^k).

2.7 Validation

2.7.1 Lid-driven cavity flow

The lid-driven cavity has been used extensively for validation of flow solvers in the literature. The flow develops inside a closed square cavity, $\Omega = [0,1] \times [0,1]$, with the top boundary moving at constant speed $g(x) = 1$, cf. figure 2.4. The boundary conditions are therefore:

$$\mathbf{u}(0,y) = 0,\ \mathbf{u}(1,y) = 0,\ \mathbf{u}(x,0) = 0,\ u(x,1) = 1,\ v(x,1) = 0, \tag{2.16}$$

while the adequate condition for pseudo-pressure is homogeneous Neumann over the entire boundary.

Figure 4.5 shows velocity profiles of our computations pertaining to a spatial grid with 513×513 pressure nodes. For this case, a cartesian processor grid with 4×4 processors was used. The steady state was reached by convergence of the maximum norm of the residual below 10^{-7}. The results correspond well with results from Ghia *et al.* [7] at the present Reynolds number of $Re = 400$.

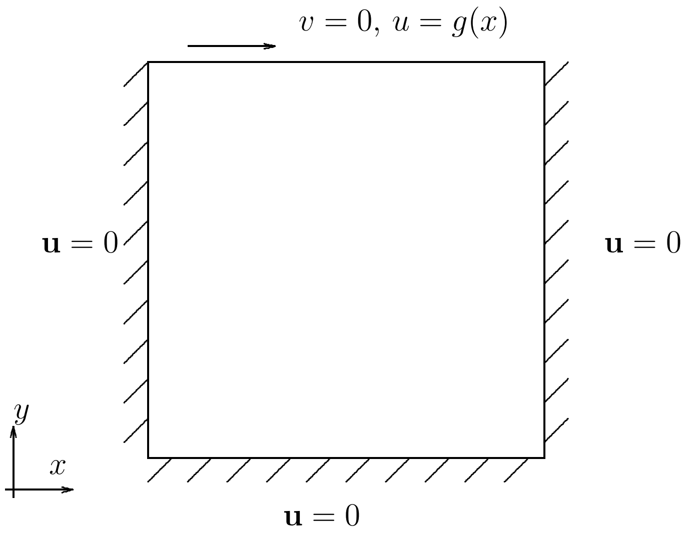


Figure 2.4: Schematic of the lid-driven cavity configuration.

2.8 Performance

The performance of the full fluid step is shown in figure 4.6 in form of the parallel efficiency. If we require a reasonable parallel efficiency of 50%, say, then with grids in the range of 512–2048, the number of processors should be chosen as 4, i.e. `nxprocs` = `nyprocs` = 2.

Chapter 3

Solid phase parallelism

Here we shall consider the parallelization of the immersed boundary method for the simulation of solid particles “on top” of the parallelization of the fluid phase which was presented in chapter 2.

3.1 Basic algorithm of the immersed boundary method

The full algorithm which needs to be re-implemented for multi-processor machines is the following (cf. [1]):

$$\begin{aligned}
\mathbf{X}^{(d)}(s_l,p) &= \mathbf{x}_{c,p}^{k-1} + r_{c,p}\left(\cos\left(\frac{2\pi(l-1)}{n_L}+\theta_{c,p}^{k-1}\right),\sin\left(\frac{2\pi(l-1)}{n_L}+\theta_{c,p}^{k-1}\right)\right) && \text{(3.1a)}\\
\mathbf{U}^{(d)}(s_l,p) &= \mathbf{u}_{c,p}^{k-1} + \boldsymbol{\omega}_{c,p}\times\left(\mathbf{X}^{(d)}(s_l,p)-\mathbf{x}_{c,p}^{k-1}\right) && \text{(3.1b)}\\
\mathbf{F}^k(s_l,p) &= \kappa\left(\mathbf{X}^{(d)}(s_l,p)-\mathbf{X}^{k-1}(s_l,p)\right) + 2\gamma\left(\mathbf{U}^{(d)}(s_l,p)-\mathbf{U}^{k-1}(s_l,p)\right)\,, && \text{(3.1c)}\\
\mathbf{f}^k(\mathbf{x}) &= \sum_p\sum_l \mathbf{F}^k(s_l,p)\,\delta_h\left(\mathbf{x}-\mathbf{X}^{k-1}(s_l,p)\right)\Delta s\,, && \text{(3.1d)}\\
\frac{\mathbf{u}^*-\mathbf{u}^{k-1}}{\Delta t} &= \alpha_k\nu\nabla^2(\mathbf{u}^{k-1}+\mathbf{u}^*) - 2\alpha_k\nabla p^{k-1} \\
&\quad -\gamma_k\left[(\mathbf{u}\cdot\nabla)\mathbf{u}\right]^{k-1} - \zeta_k\left[(\mathbf{u}\cdot\nabla)\mathbf{u}\right]^{k-2} + \gamma_k\mathbf{f}^k + \zeta_k\mathbf{f}^{k-1}\,, && \text{(3.1e)}\\
\nabla^2\phi^k &= \frac{\nabla\cdot\mathbf{u}^*}{2\alpha_k\Delta t}\,, && \text{(3.1f)}\\
\mathbf{u}^k &= \mathbf{u}^* - 2\alpha_k\Delta t\nabla\phi^k\,, && \text{(3.1g)}\\
p^k &= p^{k-1} + \phi^k - \alpha_k\Delta t\,\nu\nabla^2\phi^k\,, && \text{(3.1h)}\\
\mathbf{U}^k(s_l,p) &= \sum_{i,j}\mathbf{u}^k(\mathbf{x}_{i,j})\,\delta_h\left(\mathbf{x}_{i,j}-\mathbf{X}^{k-1}(s_l,p)\right)\Delta x\Delta y\,, && \text{(3.1i)}\\
\mathbf{X}^k(s_l,p) &= \mathbf{X}^{k-1}(s_l,p) + \alpha_k\Delta t\left(\mathbf{U}^k(s_l,p)+\mathbf{U}^{k-1}(s_l,p)\right)\,, && \text{(3.1j)}\\
\frac{\mathbf{u}_{c,p}^k-\mathbf{u}_{c,p}^{k-1}}{\Delta t} &= \frac{\rho_f}{V_{c,p}(\rho_{c,p}-\rho_f)}\left(-\gamma_k\int_{\mathcal{S}}\mathbf{f}^k\,\mathrm{d}V^{k-1} - \zeta_k\int_{\mathcal{S}}\mathbf{f}^{k-1}\,\mathrm{d}V^{k-2}\right) + 2\alpha_k\mathbf{g}\,, && \text{(3.1k)}\\
\frac{\mathbf{x}_{c,p}^k-\mathbf{x}_{c,p}^{k-1}}{\Delta t} &= \alpha_k\left(\mathbf{u}_{c,p}^k+\mathbf{u}_{c,p}^{k-1}\right)\,. && \text{(3.1l)}\\
\frac{\boldsymbol{\omega}_{c,p}^k-\boldsymbol{\omega}_{c,p}^{k-1}}{\Delta t} &= \frac{\rho_f}{I_{c,p}}\left(-\gamma_k\int_{\mathcal{S}}(\mathbf{r}^{k-1}\times\mathbf{f}^k)\,\mathrm{d}V^{k-1} - \zeta_k\int_{\mathcal{S}}(\mathbf{r}^{k-2}\times\mathbf{f}^{k-1})\,\mathrm{d}V^{k-2}\right.\\
&\quad\left. + \frac{\left(\int_{\mathcal{S}}\left(\mathbf{r}^{k-1}\times\mathbf{u}^k\right)\,\mathrm{d}V^{k-1}\right)-\left(\int_{\mathcal{S}}\left(\mathbf{r}^{k-2}\times\mathbf{u}^{k-1}\right)\,\mathrm{d}V^{k-2}\right)}{\Delta t}\right)\,. && \text{(3.1m)}\\
\frac{\boldsymbol{\theta}_{c,p}^k-\boldsymbol{\theta}_{c,p}^{k-1}}{\Delta t} &= \alpha_k\left(\boldsymbol{\omega}_{c,p}^k+\boldsymbol{\omega}_{c,p}^{k-1}\right)\,. && \text{(3.1n)}
\end{aligned}$$

The additional nomenclature is as follows: $\mathbf{X}(s_l,p)$ are the Lagrangian marker points of the pth particle, s_l the discrete coordinate on the pth solid-fluid interface with $1\le l\le n_L$; $\mathbf{X}^{(d)}(s_l,p)$ are the desired Lagrange marker locations; $\mathbf{F}^k(s_l,p)$ is the singular boundary force at the lth Lagrange location of the pth particle at Runge-Kutta sub-step k; similarly for $\mathbf{U}^k(s_l,p)$, $\mathbf{U}^{(d)}(s_l,p)$ which are the Lagrange velocities and desired velocities; κ and γ are the stiffness and damping constants of the virtual forces connecting actual Lagrange marker positions and desired positions; $\mathbf{x}_{c,p}^k$, $\mathbf{u}_{c,p}^k$, $\theta_{c,p}^k$, $\omega_{c,p}^k$, $\rho_{c,p}$, $r_{c,p}$ are the center position, linear velocity, angular position, angular velocity, density and radius, respectively, of the pth particle.

Steps (3.1e)–(3.1h) have already been dealt with in chapter 2.

3.2 Master-and-slaves strategy

Keeping in mind the domain decomposition method employed for the fluid parallelization (cf. § 2.2) it is most reasonable to have a given processor deal with those particles which are currently located within its local sub-domain. There are two catches to this strategy: (i) particles will in general be non-evenly distributed over the sub-domains, leading to problems of load-balancing; (ii) particles will naturally cross borders between sub-domains causing a variety of complications such as the necessity to hand-over the control to other processors and the over-lapping of particles between adjacent sub-domains. Since particle-related computations will need to be shared amongst multiple processors in some instances, we will adopt a master-and-slaves strategy. This means that at any time there will be one processor responsible for the general handling of each particle (that particle's "master") and there will be a number of additional processors which help the master in dealing with the specific particle (the "slaves" of that particle's master). The number of slave processors associated with each particle changes in time and can be any integer, including zero.

3.2.1 Accounting for particles

Accounting for local particles and association with their global numbering is done by defining a local list of pointers which will be called $indcloc(1:\texttt{nclmx})$. For a given local index i, $indcloc(i)$ gives the corresponding global identification (icg, say): if the value of $indcloc(i)$ is positive, then the present processor is the master for this particle; if the value is negative, the present processor is one of the slaves for this particle; if the value is zero, the slot i is free, i.e. there is currently no particle assigned to this location. The global numbering runs in ascending order as $1 \leq icg \leq \texttt{ncobj}$.

The cooperation between master and slave processors is accounted for in a logical (bit) array $lcoop(1:8, 1:\texttt{nclmx})$. Let us consider a particle with local index i. If the current processor is this particle's master, then $lcoop(j,i)$ determines whether the local neighbor j (cf. figure 2.2) acts as a slave or not; in the case where the processor is itself a slave for particle i, then exactly one bit out of $lcoop(1:8,i)$ is set to "true" and its index indicates which neighbor is the responsible master.

3.2.2 Determination of the master processor

For a particle with given center position $\mathbf{x}_{c,i}(t)$ we define as the master processor the one whose sub-domain includes the point under consideration. The sub-domain is defined as the union of the cells surrounding all pressure nodes held locally, i.e. the rectangular area limited by the four lines:

$$\begin{aligned} x &= xu(ibegp(\texttt{my\_col}))\,, \\ x &= xu(iendp(\texttt{my\_col})+1)\,, \\ y &= yv(jbegp(\texttt{my\_row}))\,, \\ y &= yv(jendp(\texttt{my\_row})+1)\,. \end{aligned} \tag{3.2}$$

Presently, it is supposed that the temporal resolution of the process is sufficient such that a particle's position can advance at most from one sub-domain to an adjacent one within one time-step. Therefore, we can simplify the determination of the successor to the present master in such a case by only checking the boundaries of the nearest neighbors.

3.2.3 Determination of the slave processors

In order to determine which processors need to contribute to a given particle's balances, we need to consider the various steps of the algorithm (3.1). Shared computations will only be those where Eulerian quantities are involved, fields being distributed as outlined in § 2.2. Therefore, the shared operations are the spreading of the Lagrangian forces (3.1d) and the interpolation of the Eulerian velocities (3.1i). Both operations involve essentially the same loops with the same index ranges, determined by the width of the support of the regularized delta function. What we need to do in order to find out if a particle with a given position and radius can be handled alone by its master is to compute the extreme indices swept over by the loops corresponding to (3.1d) and (3.1i) and compare them to the index range held by the master. This check needs to be done for both velocity components (staggered grid).

In the case where the loops exceed the local range, the corresponding neighbors are added to the local list of slaves for the particle in question (i.e. the cooperation bits in $lcoop$ are set accordingly) and data is prepared for communication (cf. § 3.2.5 below). Here we suppose that the particles are of such a dimension that they will not need cooperation beyond the nearest 8 neighbors, i.e. roughly that the radius is smaller than the linear dimension of a sub-domain.

3.2.4 Old master/new master communication

The data to be passed from the current master to its successor in the case of a particle leaving the present master's sub-domain is the following:

- particle-related quantities $\mathbf{u}^k_{c,p}$, $\mathbf{x}^k_{c,p}$, $\omega^k_{c,p}$, $\theta^k_{c,p}$, $r_{c,p}$, $\rho_{c,p}$, i.e. 8 real data per particle;
- particle-related integral quantities $\int_{\mathcal{S}} \mathbf{f}^k \,\mathrm{d}V^{k-1}$ $\int_{\mathcal{S}} (\mathbf{r}^{k-1} \times \mathbf{f}^k) \,\mathrm{d}V^{k-1}$, $\int_{\mathcal{S}} \left(\mathbf{r}^{k-1} \times \mathbf{u}^k\right) \,\mathrm{d}V^{k-1}$, i.e. 4 real data per particle;
- Lagrange-marker-related quantities $\mathbf{U}^k(s_l, p)$, $\mathbf{X}^k(s_l, p)$, $\mathbf{F}^k(s_l, p)$, i.e. $6n_L$ real data per particle.

The communication itself is carried out in two phases. The first consists of a mandatory exchange of a single integer between each processor and all of its 8 neighbors. The datum to be exchanged evidently is the number of particles having recently crossed from the local sub-domain to the respective neighbors sub-domain, say $nsend(1:8)$. The data in this phase is exchanged by the scheme of § 2.3 with the data type being `MPI_INTEGER`. The result is stored in $nrecv(1:8)$.

The second phase is the exchange of the actual data, carried out only for those processor pairs where communication occurs. Therefore, a slightly modified variant of the algorithm of § 2.3 applies here. Supposing that the data to be exchanged was previously assembled by the current master in the buffer $outbuf$ then the following scheme is used:

```
Require: $ibd$, $neighbor$, iseq_send, iseq_recv, outbuf
  ireq=0
  for $ii = 1:8$ do
    $idest$ = iseq_send($ii$)
    if $ibd(idest)\,\&\,nsend(idest) > 0$ then
      $ireq \leftarrow ireq + 1$
      $rank\_dest = neighbor(idest)$
      call MPI_ISEND($outbuf(1, idest)$,...,$rank\_dest$,...)
    end if
    $iorig$ = iseq_recv($ii$)
    if $ibd(iorig)\,\&\,nrecv(iorig) > 0$ then
      $ireq \leftarrow ireq + 1$
      $rank\_orig = neighbor(iorig)$
      call MPI_IRECV($inbuf(1, iorig)$,...,$rank\_orig$,...)
    end if
  end for
  call MPI_WAITALL(ireq,...)
```

The sequence of destinations/origins is the same as given in § 2.3.

3.2.5 Master/slaves communication

3.2.5.1 Initialization as slaves

After each update of the particle positions the slaves are completely re-determined and data is transmitted anew from the master to all contributing slaves of a given particle. The data volume consists in the following:

- Particle-related quantities $\mathbf{u}^k_{c,p}$, $\mathbf{x}^k_{c,p}$, $\omega^k_{c,p}$, $\theta^k_{c,p}$, $r_{c,p}$, i.e. 7 real data per particle.
- Lagrange-marker-related quantities $\mathbf{U}^k(s_l, p)$, $\mathbf{X}^k(s_l, p)$, $\mathbf{F}^k(s_l, p)$, i.e. $6n_L$ real data per particle.

- For reasons of accounting, we also transmit the local (to the master) index of the particle in question. This measure helps us to avoid the need for "double bookkeeping" of particle indices or a search through the local index list when the inverse communication is carried out subsequently (i.e. slave-to-master, cf. § 3.2.5.2).

Again, there are two phases of communication, the first being the exchange of the number of data packages to be exchanged between each pair of neighbors. After this step, each processors holds the arrays of counters $nsend(1:8)$ and $nrecv(1:8)$. These counters are preserved through the following time-step for subsequent utilization during shared operations (cf. § 3.2.5.2 below). The second phase is similar to the second phase described in § 3.2.4, except that the data volume per particle is slightly inferior here.

3.2.5.2 Shared computation of interpolation step and integrals

Here we will discuss how the operation

$$\mathbf{U}^k(s_l,p) = \sum_{i,j} \mathbf{u}^k(\mathbf{x}_{i,j})\,\delta_h\left(\mathbf{x}_{i,j} - \mathbf{X}^{k-1}(s_l,p)\right)\Delta x \Delta y\,, \tag{3.3}$$

is carried out following the domain decomposition technique of the fluid and our master-slaves strategy for the particles. Firstly, we suppose that both master and slaves hold *all* of the Lagrangian marker-point positions $\mathbf{X}^{k-1}(s_l,p)$. Then, let both master and slaves accumulate $\tilde{\mathbf{U}}^k(s_l,p)$ as far as their local index range of the Euler field $\mathbf{u}$ allows. The desired result is simply the sum of $\tilde{\mathbf{U}}^k(s_l,p)$ over all slaves and the master. Therefore, the master needs to perform a gathering operation of type on the intermediate result $\tilde{\mathbf{U}}^k(s_l,p)$. Furthermore, this communication is exactly the inverse of the initial exchange between master and slaves (§ 3.2.5.1) with respect to the number of packages of data, i.e. here a master receives $nsend(i)$ from the neighbor i and a slave needs to send $nrecv(i)$ packages to the neighbor i. The volume of each package is evidently n_L. The unpacking operation of the data received by the master involves a summation.

Similarly, the computation of the volume integral of the angular momentum within a particle domain $\int_{\mathcal{S}}\left(\mathbf{r}^{k-1}\times\mathbf{u}^k\right)\mathrm{d}V^{k-1}$ is carried out in a shared fashion with the same procedure of gathering by the master. The data volume is simply one real per particle and per slave.

It should be noted that the volume integrals over the singular force/torque can be carried out without communication since it is equal to the sum over the force at the Lagrangian marker points.

3.2.6 Information flow: who is holding what at what stage?

The flowchart in table 3.1 gives an overview of the flow of information between the various processors involved in the computation of quantities pertaining to a given particle. It can be seen that all the necessary particle-center-related information is passed on to slave processors during the reassignment step such that each processor can (redundantly) compute the Lagrangian forces $\mathbf{F}^k$ in local memory. Further communication is then only necessary during the Euler-to-Lagrange step (3.1i) and for the gathering of the results from integrations.

3.3 Performance

3.3.1 Intensity of communication due to particles

Let the total number of particles leaving their master's sub-domain at a given step be $n_{\mathcal{L}}$. Analogously, let $n_{\mathcal{O}}$ be the global sum (over all particles) of the number of slaves per particle. Inspecting the whole algorithm we find that the total data volume to be sent/received between processors due to the presence of particles is:

$$G_V = \underbrace{16}_{\text{integers}} + \underbrace{n_{\mathcal{O}}\cdot(13+7\,n_L) + n_{\mathcal{L}}\cdot(13+6\,n_L)}_{\text{reals}} \quad . \tag{3.4}$$

Table 3.1: Schematic flowchart of the present parallel immersed boundary algorithm. The letters "M" and "S" refer to master and slave processors w.r.t. a given particle.

result	arguments	equation	who executes?	M holds	S holds	comm
$\mathbf{X}^{(d)}$	$\mathbf{x}_c^{k-1}$, r_c, θ_c^{k-1}	3.1a	M & S (redundant)	all	all	—
$\mathbf{U}^{(d)}$	$\mathbf{u}_c^{k-1}$, $\mathbf{x}_c^{k-1}$, ω_c^{k-1}, $\mathbf{X}^{(d)}$	3.1b	M & S (redundant)	all	all	—
$\mathbf{F}^k$	$\mathbf{X}^{(d)}$, $\mathbf{X}^{k-1}$, $\mathbf{U}^{(d)}$, $\mathbf{U}^{k-1}$	3.1c	M & S (redundant)	all	all	—
$\mathbf{f}^k$	$\mathbf{F}^k$	3.1d	M & S (partially)	all	all	—
$\mathbf{f}^{k-1}$	$\mathbf{F}^{k-1}$	3.1d	M & S (partially)	all	all	—
$\mathbf{u}^k$	$\mathbf{f}^k$,$\mathbf{f}^{k-1}$		$\boxed{\text{fluid step}}$			
$\mathbf{U}^k$	$\mathbf{u}^k$	3.1i	M & S (partially)	local $\mathbf{u}^k$	local $\mathbf{u}^k$	$\times$
$\mathbf{X}^k$	$\mathbf{X}^{k-1}$, $\mathbf{U}^k$, $\mathbf{U}^{k-1}$	3.1j	M	all	—	—
$\mathbf{u}_c^k$, $\mathbf{x}_c^k$, ω_c^k, θ_c^k	$\mathbf{u}_c^{k-1}$, $\mathbf{x}_c^{k-1}$, ω_c^{k-1}, θ_c^{k-1}, ρ_c, r_c $\int_\mathcal{S} \mathbf{f}^k \, \mathrm{d}V^{k-1}$, $\int_\mathcal{S} \mathbf{f}^{k-1} \, \mathrm{d}V^{k-2}$, $\int_\mathcal{S} (\mathbf{r}^{k-1} \times \mathbf{f}^k) \, \mathrm{d}V^{k-1}$, $\int_\mathcal{S} (\mathbf{r}^{k-2} \times \mathbf{f}^{k-1}) \, \mathrm{d}V^{k-2}$, $\int_\mathcal{S} \left(\mathbf{r}^{k-1} \times \mathbf{u}^k\right) \, \mathrm{d}V^{k-1}$, $\int_\mathcal{S} \left(\mathbf{r}^{k-2} \times \mathbf{u}^{k-1}\right) \, \mathrm{d}V^{k-2}$	3.1l	M; integral over $\mathbf{r} \times \mathbf{u}$: M & S partially	all $\mathbf{F}$, $\mathbf{u}_c$, $\mathbf{x}_c$, ω_c, θ_c local $\mathbf{u}$	local $\mathbf{u}$	$\times$
	$\boxed{\text{re-assignment step: old master/new master \& master/slaves initialization}}$					$\times$

Please note that it is more convenient to express the data volume *globally* here whereas it was *local* to each processor in previous counts (cf. § 2.3, 2.4, 2.5).

The number of messages to be sent per processor is not constant, but depends as well upon the positions of the particles. It is bounded as follows:

$$16 \leq N_{SOLID} \leq 48\,. \tag{3.5}$$

The constant part stems from mandatory preliminary messages which determine the size of the actual data to be exchanged between each processor pair. The upper limit of 48 is obtained when a given processor needs to exchange slave and master data with all its eight direct neighbors.

3.3.2 Parallel efficiency

We compute a case where 48 particles are settling under the action of gravity in a closed ambient container. Each particle is represented by $n_L = 30$ Lagrangian marker points. Two Eulerian grids were considered: $n_{xp} = n_{yp} = 512$ (i.e. $ixp = jyq = 4$, $iex = jey = 8$) and $n_{xp} = n_{yp} = 2048$ (i.e. $ixp = jyq = 16$, $iex = jey = 8$). Therefore, both Eulerian grids allow for the same number of multi-grid levels. The distribution of the particles is un-balanced between processors, but the un-balancing was not maintained while the number of active processors was changed.

The parallel efficiency for this flow case is given in the figure 4.7. The accounting was done for a single time step only; it does change in time due to the varying distribution of the particles. The present numbers are however instructive and can be taken as a rough guide.

It is somewhat surprising to see an efficiency of over 80% with `nxprocs` = `nyprocs` = 2 in the case of the smaller grid whereas for the larger grid the efficiency is at only 50%. We believe that this effect is due to the above mentioned difference in the distribution of the particles. For the larger processor count of `nxprocs` = `nyprocs` = 4 we obtain the expected result that the parallel efficiency for the smaller grid has dropped to unacceptable 10% while the larger grid can still be computed at an efficiency of nearly 40%.

Chapter 4

Conclusions

We have described a method of parallelization of a particulate flow code of the immersed boundary type for execution on distributed memory machines. The parallelization of the fluid part of the algorithm employs standard domain decomposition techniques for Cartesian grids. The solutions of the linear systems stemming from the discretization of the Helmholtz and Poisson problems are obtained by means of an approximate factorization method and a multi-grid technique respectively. The kernel of the former scheme is a tri-diagonal solver for executing the one-dimensional sweeps in parallel, parallelized over one of the Cartesian processor grid dimensions. The multi-grid method uses the same parallelization as the basic domain decomposition technique, but applied to all grid refinement levels.

The parallelization of the solid part was implemented through a master-slaves technique. The processor whose sub-domain includes a particle's center point is designated the master for the particle. The master carries out the computations associated with the Lagrangian marker points and those due to the Newtonian particle tracking as far as possible, i.e. as much as the extension of its Eulerian sub-domain allows. When a particle overlaps several sub-domains, i.e. when its balances cannot be carried out by its master from local memory any more, the respective neighbor processors are designated as slave processors, contributing to the computation. Limited communication between masters and slaves is necessary at three points during the algorithm.

Preliminary computations of fluid-particle systems of low dimensionality demonstrate the feasibility of the current approach in terms of the parallel efficiency. This quantity is, however, somewhat difficult to define reliably in the context of freely-moving particles due to time-varying load distribution between processors. Therefore, future computations should provide long-time statistics of the parallel efficiency for various configurations, system sizes and resolutions.

Acknowledgements

The basic version of the parallel multi-grid code used for the solution of the Poisson problem was provided by B. Bunner.

Bibliography

[1] M. Uhlmann. First experiments with the simulation of particulate flows. Technical Report No. 1020, CIEMAT, Madrid, Spain, 2003. ISSN 1135-9420.

[2] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.

[3] B. Bunner. Code package MGD: an MPI-parallel replacement for MUDPACK. URL: `http://www.mgnet.org/mgnet-codes-bunner.html`.

[4] T. Rossi and J. Toivanen. A parallel fast direct solver for block tridiagonal systems with separable matrices of arbitrary dimension. *SIAM J. Sci. Comp.*, 20(5):1778–1793, 1999.

[5] J. Adams. Mudpack-2: Multigrid software for elliptic partial differential equations on uniform grids with any resolution. *Appl. Math. Comp.*, 53:235–249, 1993.

[6] N. Mattor, T.J. Williams, and D.W. Hewett. Algorithm for solving tridiagonal matrix problems in parallel. *Parallel Computing*, 21:1769–1782, 1995.

[7] V. Ghia, K.N. Ghia, and C.T. Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multi-grid method. *J. Comput. Phys.*, 48:387–411, 1982.

[8] C.G. Hirsch. *Numerical computation of internal and external flows*. J. Wiley, 1990.

[9] A.R. Mitchell and D.F. Griffiths. *The finite difference method in partial differential equations*. J. Wiley, 1980.

Figures

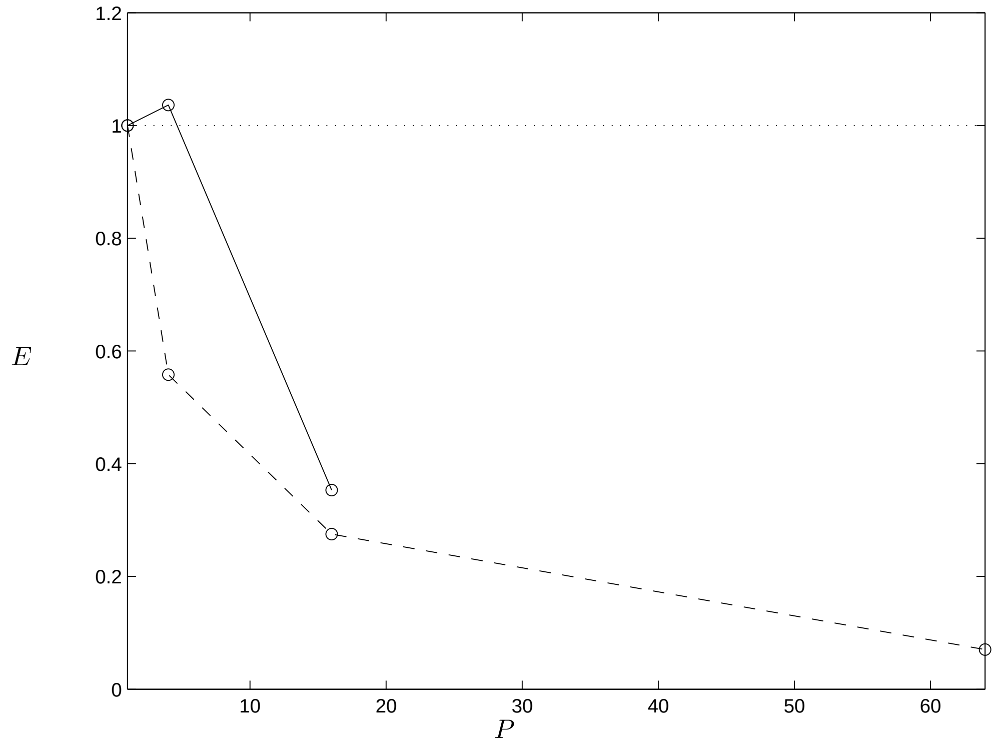


Figure 4.1: Parallel efficiency $E = T_P/(P \cdot T_1)$ of the parallel multi-grid algorithm executed with P processors on an SGI ORIGIN 3800. The two curves correspond to problem sizes of $n_{xp} = n_{yp} = 512$ (———) and $n_{xp} = n_{yp} = 2048$ (- - - -).

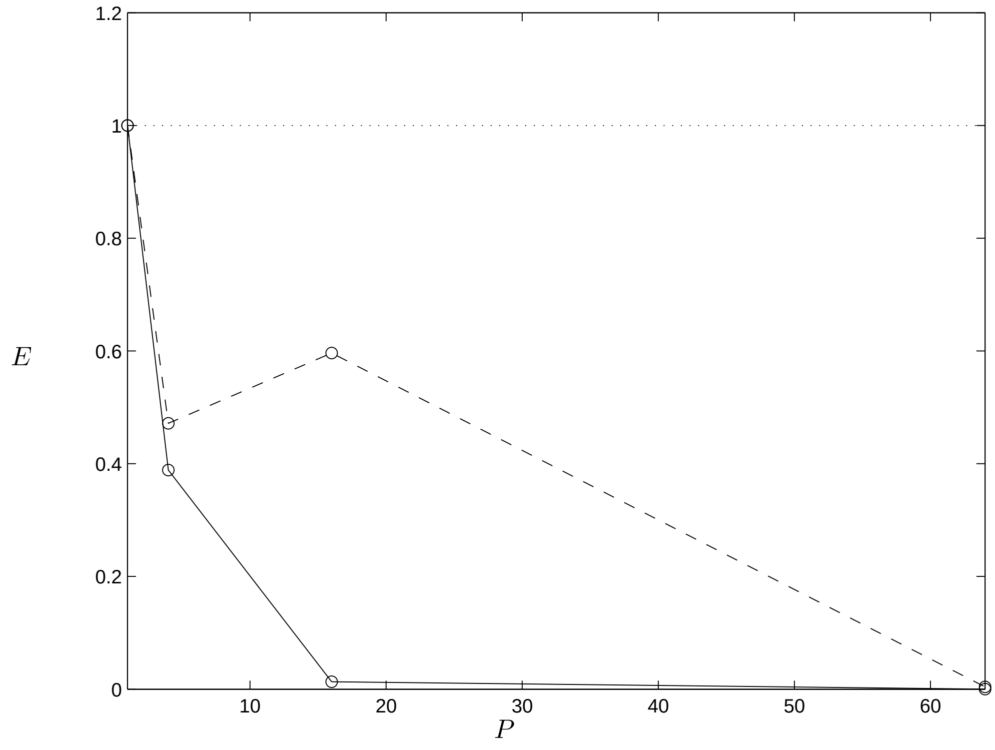


Figure 4.2: Parallel efficiency $E = T_P/(P \cdot T_1)$ of the parallel approximate factorization step executed with P processors on an SGI ORIGIN 3800. The two curves correspond to problem sizes of $n_{xp} = n_{yp} = 512$ (——) and $n_{xp} = n_{yp} = 2048$ (- - - -).

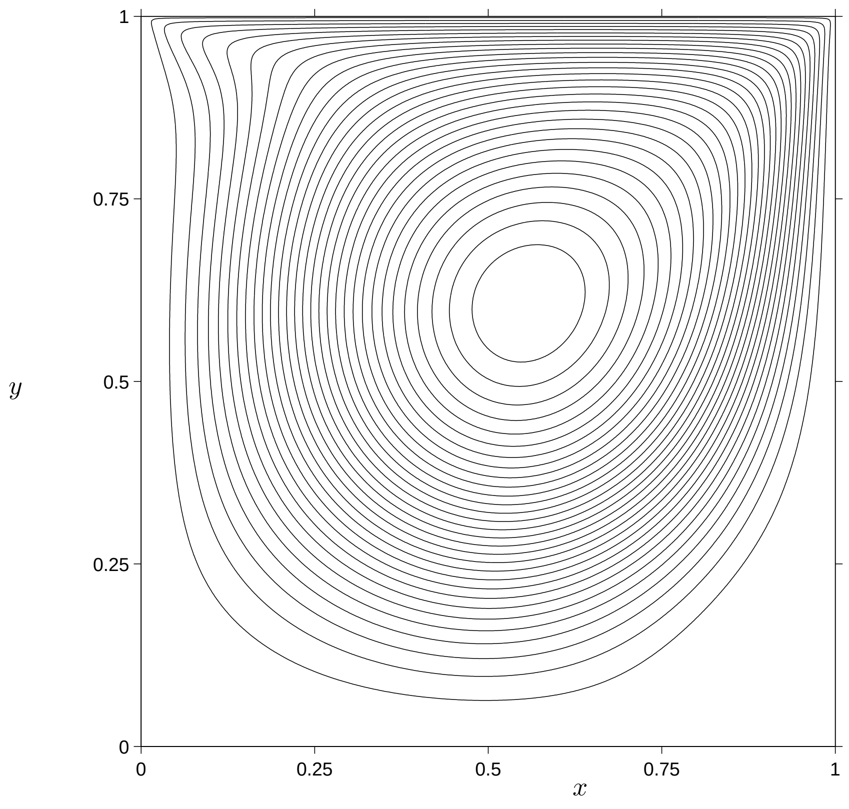
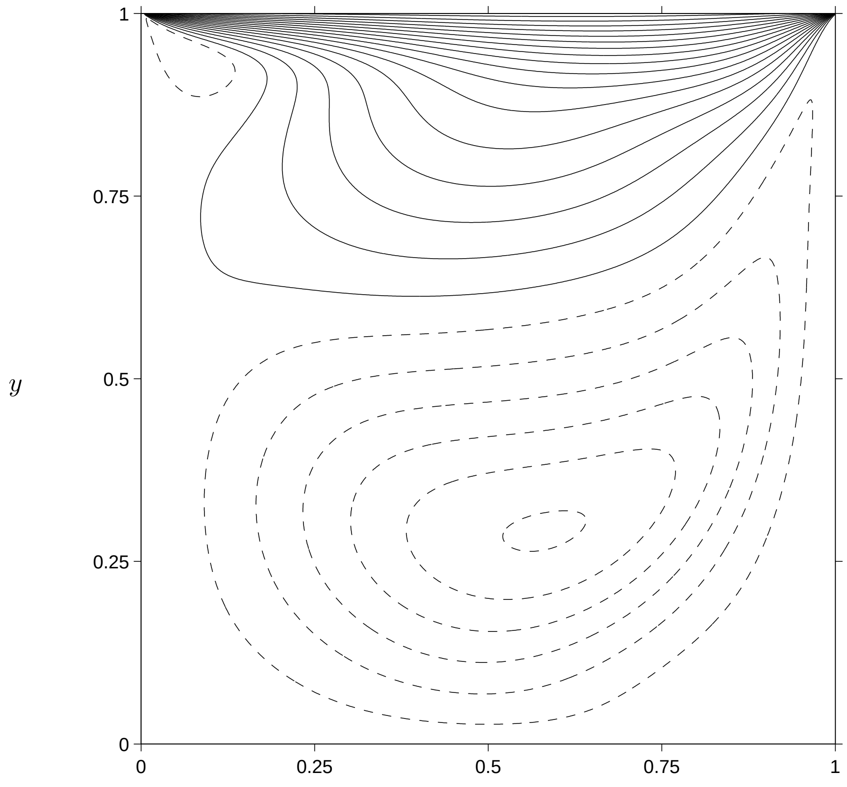


Figure 4.3: Contour lines of the streamfunction for the case of the lid-driven cavity at $Re = 400$.



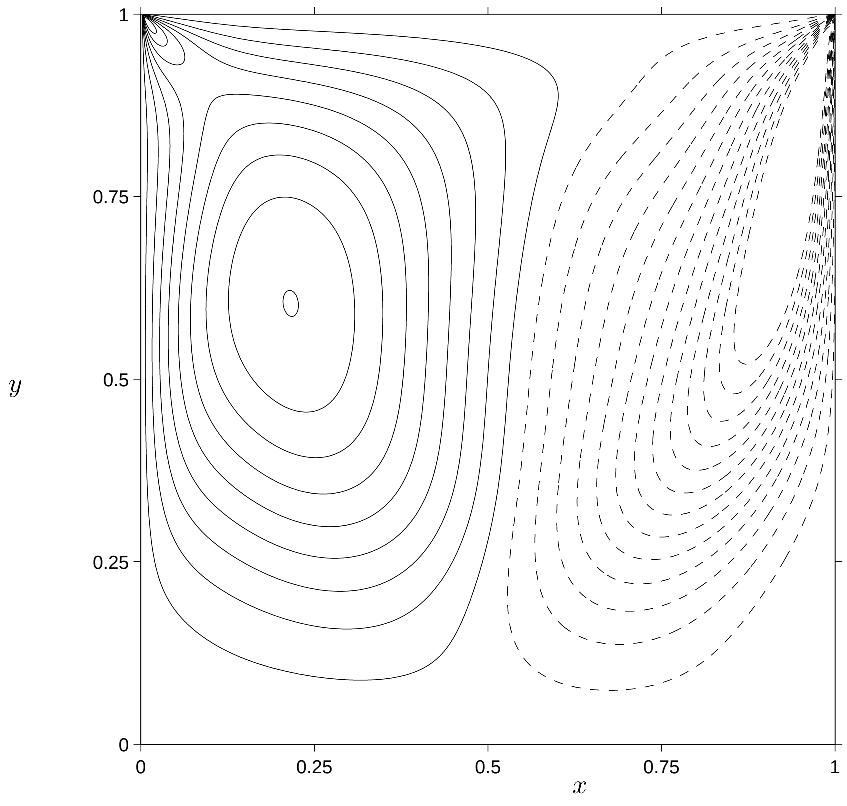


Figure 4.4: Contour lines of the x-component of velocity (top) and of the y-component of velocity (bottom) for the case of the lid-driven cavity at $Re = 400$. The solid lines correspond to positive values, the dashed lines to negative values.

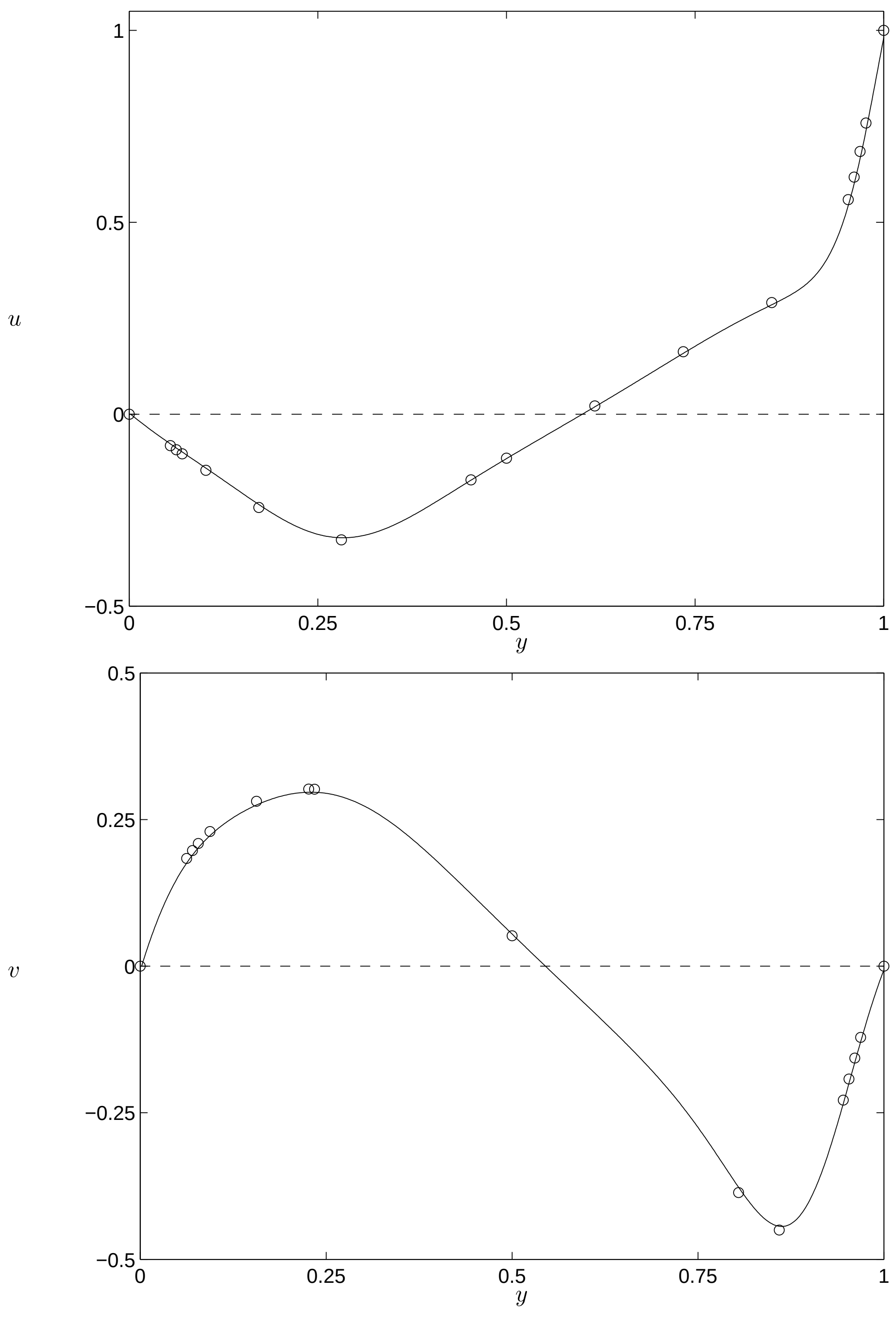


Figure 4.5: Profiles of the x-component of velocity along the y-direction (top) and of the y-component of velocity along the x-direction (bottom) for the case of the lid-driven cavity at $Re = 400$. The solid lines correspond to the present results obtained with 513×513 pressure node points; the symbols are taken from reference [7].

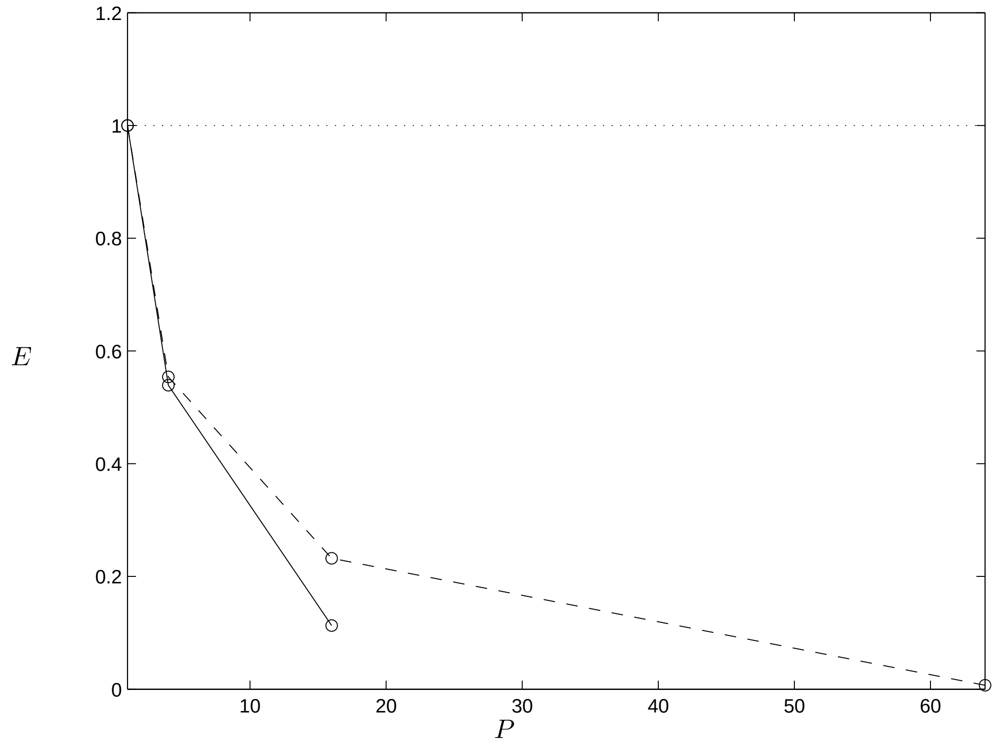


Figure 4.6: Parallel efficiency $E = T_P/(P \cdot T_1)$ of a full step of the parallel fluid solver executed with P processors on an SGI ORIGIN 3800. The two curves correspond to problem sizes of $n_{xp} = n_{yp} = 512$ (———) and $n_{xp} = n_{yp} = 2048$ (- - - -).

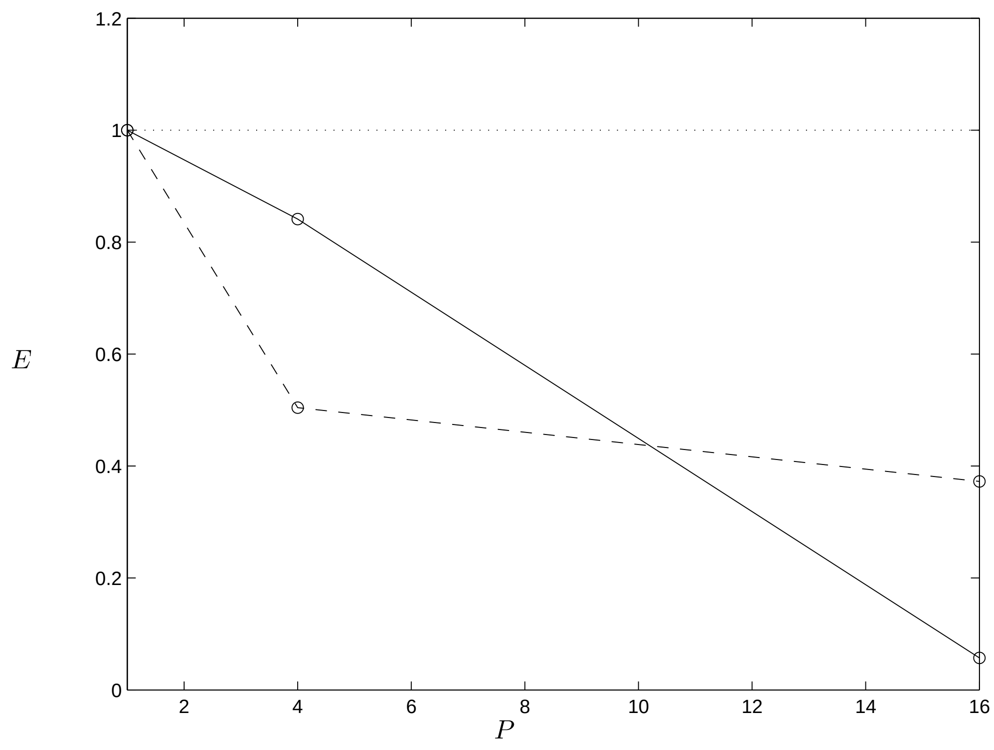


Figure 4.7: Parallel efficiency $E = T_P/(P \cdot T_1)$ of a full step of the parallel fluid & solid solver executed with P processors on an SGI ORIGIN 3800. Here, `nxprocs` = `nyprocs` = $\sqrt{P}$. The two curves correspond to problem sizes of $n_{xp} = n_{yp} = 512$ (———) and $n_{xp} = n_{yp} = 2048$ (----). In both cases, 48 particles with $n_L = 30$ Lagrange markers each were assigned and unequally distributed over the processors.

Appendix A

Approximate factorization method

The predictor step for velocity (2.1a) can be rewritten as

$$u_i^* - u_i^{k-1} = C(S_x + S_y)\left(u_i^* + u_i^{k-1}\right) + F_i \tag{A.1}$$

for convenience, defining:

$$\begin{aligned} C &= \nu\alpha_k\Delta t & \text{(A.2a)}\\ S_x &= \partial_{xx} & \text{(A.2b)}\\ S_y &= \partial_{yy} & \text{(A.2c)}\\ F_i &= \Delta t\left\{-\gamma_k\left(u_j u_{i,j}\right)^{k-1} - \zeta_k\left(u_j u_{i,j}\right)^{k-2} - 2\alpha_k p_{,i}^{k-1} + \gamma_k f_i^k + \zeta_k f_i^{k-1}\right\}. & \text{(A.2d)}\end{aligned}$$

The following factorization of equation (A.1) is carried out (cf. [8, p.440]):

$$(1 - CS_x)(1 - CS_y)u_i^* = (1 + CS_x)(1 + CS_y)u_i^{k-1} + F_i \quad , \tag{A.3}$$

which is equivalent to the original equation up to $\mathcal{O}(\Delta t^3)$. The splitting of the factorized scheme into two separate one-dimensional steps is done according to the method of D'Yakonov (cf. [9, p.61]):

$$\begin{aligned}(1 - CS_x)u_i' &= (1 + CS_y)(1 + CS_x)u_i^{k-1} + F_i\,, & \text{(A.4a)}\\ (1 - CS_y)u_i^* &= u_i'\,. & \text{(A.4b)}\end{aligned}$$

The intermediate solution u_i' does not have a physical meaning and is later discarded. However, one needs to apply consistent boundary conditions to the first sweep in order to obtain the expected solution after the second sweep. This value is simply given by the second formula, i.e. at the boundary we need to impose the following:

$$\left(u_i'\right)_{L,R} = (1 - CS_y)\left(u_i^*\right)_{L,R} \quad , \tag{A.5}$$

where $()_{L,R}$ refers to values at the left and right boundary of the computational domain. Therefore, the adequate boundary condition at the extrema of the first sweep is obtained by a combination of the operator along those boundaries applied to the desired boundary conditions u_i^* which is available in the case of Dirichlet boundary conditions and a grid which is centered w.r.t. the physical boundary. The conclusion for a staggered grid is that one needs to perform first a sweep in the non-staggered direction and subsequently in the staggered one, i.e. for the x-velocity sweep first in y then in x and vice versa for the y-velocity. In the case of a homogeneous Neumann condition, it should be chosen to lie along the staggered boundary (i.e. for the x-velocity: the lines $x = cst$). This means that we are presently neither capable to simulate two perpendicular Neumann conditions for a single velocity component nor Neumann conditions for both components at a single boundary. Fortunately, the present cases of interest do not fall into either class.

Appendix B

Parallel tri-diagonal solver

Let us consider the linear algebraic system

$$\mathbf{\Lambda}\,\mathbf{x} = \mathbf{r} \quad , \tag{B.1}$$

where $\mathbf{\Lambda}$ is a tri-diagonal matrix of size $N \times N$. We suppose that there are P processors available and, for simplicity, $N = MP$ with M integer.

The algorithm of Mattor *et al.* [6] essentially solves the following three reduced systems—local to each processor p—instead:

$$\begin{aligned}
\mathbf{L}_p\,\mathbf{x}_p^r &= \mathbf{r}_p, &\text{(B.2a)}\\
\mathbf{L}_p\,\mathbf{x}_p^{UH} &= \mathbf{b}_p^{UH}, &\text{(B.2b)}\\
\mathbf{L}_p\,\mathbf{x}_p^{LH} &= \mathbf{b}_p^{LH}, &\text{(B.2c)}
\end{aligned}$$

for the vectors $\mathbf{x}_p^r$, $\mathbf{x}_p^{UH}$, $\mathbf{x}_p^{LH}$. According to the analogy with inhomogeneous differential equations, these vectors correspond to the "particular", the "upper homogeneous" and the "lower homogeneous" solutions. $\mathbf{L}_p$ is a reduced system matrix, $\mathbf{r}_p$ the reduced right-hand-side vector and $\mathbf{b}_p^{UH}$, $\mathbf{b}_p^{LH}$ are vectors with coupling terms which depend only upon the members of the original matrix $\mathbf{\Lambda}$. Therefore, the "homogeneous" solutions $\mathbf{x}_p^{UH}$, $\mathbf{x}_p^{LH}$ can be pre-computed during an initialization phase, if the matrix $\mathbf{\Lambda}$ is going to be constant during the various solution steps to be carried out afterwards, i.e. in our terms, if the time step is going to be constant. One "particular" solution of (B.2a) by LU factorization takes $5M$ operations per processor.

The final solution $\mathbf{x}$ is assembled as follows:

$$\mathbf{x} = \mathbf{x}_p^r + \xi_p^{UH}\,\mathbf{x}_p^{UH} + \xi_p^{LH}\,\mathbf{x}_p^{LH}\,, \tag{B.3}$$

which takes $4M$ operations per processor. The scalar factors ξ_p^{UH}, ξ_p^{LH} are the solution of a tri-diagonal linear system where the right-hand-side is a function of the reduced "particular" vectors $\mathbf{x}_p^r$. Therefore, it needs communication before its assembly: there are $\log_2(P)$ communication steps, the ith one sending a data volume of $8 \cdot 2^{i-1}$ real numbers. The solution of (B.3) itself—which is redundantly executed by each processor—requires $8(2P-2)$ operations.

The parallel efficiency of this algorithm is shown in figure B.1.

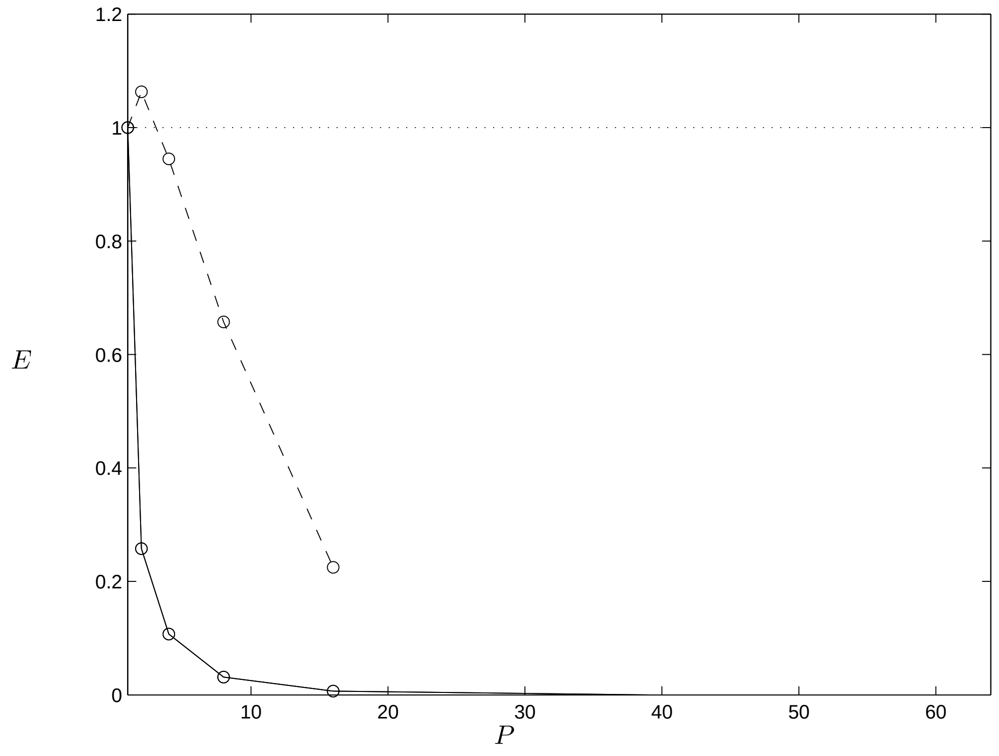


Figure B.1: Parallel efficiency $E = T_P/(P \cdot T_1)$ of the parallel tri-diagonal matrix algorithm executed with P processors on an SGI ORIGIN 3800. The two curves correspond to problem sizes of $N = 512$ (———) and $N = 50000$ (- - - -).