JOURNAL OF COMPUTATIONAL PHYSICS 97, 103–126 (1991)

Vectorising the Smooth Particle Hydrodynamics

B. HADDAD

DAEC and DEMIRM, Observatoire de Paris, Section de Meudon, F-92195 Meudon principal cedex, France

AND

F. CLAUSSET AND F. COMBES

DEMIRM, Observatoire de Paris, Section de Meudon, F-92195 Meudon principal cedex, France, and Radioastronomie Millimétrique, ENS, 24 Rue Lhomond, F-75231 Paris Cedex 05, France

Received February 27, 1989; revised December 5, 1989

A new method to vectorise the SPH (smooth particle hydrodynamics) code is presented that makes the CPU time grow linearly with the number of particles. This method is presented in 2D, but can be easily extended to 3D, with only $\approx 20\%$ increase in memory. One of the main advantages of this hydrodynamical code is that a variable particle size can be used. This implies a variable spatial resolution, particularly useful to sample high density regions, in density-contrasted physical problems. © 1991 Academic Press, Inc.

1. INTRODUCTION

Most hydrodynamical problems require numerical calculations because of their complexity and some of them, such as astrophysical collapses, bring into play steep density gradients. It may be interesting to have at one's disposal a code with variable resolution. A lot of methods are proposed to simulate the fluid equations, but one of them (i.e., SPH method [5]) calls on the Lagrangian description of a fluid and has appreciable advantages.

Thanks to the development of vectorial and parallel computers with more and more memory available, the number of particles may be large enough and the resolution sufficient for a lot of problems. To optimize the computing time we propose a new scheme to carry out the vectorising of the SPH. Our scheme is machine-independent, in the sense that it optimizes automatically the vector sizes in the function of the machine characteristics. This allows us to benefit from the advantages of a Lagrangian description having a variable resolution (unlike the Eulerian one), without being penalized by the CPU time.

In the first section we present the numerical technique that will be compared with other schemes, while in the second section our new vector 2D-scheme performed on the VP200 will be described. We explain the smoothing kernel, artificial viscosity,

0021-9991/91 $3.00

and the tests carried out with variable resolution according to the smoothing length in Sections 3 and 4. Results of isothermal collapses will be compared with analog data from other authors in Section 5.

2. Philosophy of the Code

The Numerical Technique and Comparison with Other Schemes

We use a Lagrangian description of a fluid by treating elements of fluid as finite-sized particles, the so-called smoothed particle hydrodynamics (SPH). This numerical technique was first introduced by Lucy [12], and since then well studied by Gingold and Monaghan [6, 7], Monaghan and Lattanzio [15]. It was shown to give very good results in many different applications [2, 11, 19].

Several methods can be used to simulate fluids dynamics: the Eulerian and Lagrangian schemes and intermediate ones as PIC (particle in cell) [8], which use a grid to facilitate the computations of the forces on the particles. However, the Lagrangian scheme is attractive and has important advantages over the other descriptions. First, it places the particles where the material is and should lead to a more economical calculation, especially if the code is vectorised, since the search of the neighbouring particles is the most penalizing for a scalar computation. Second, the main difficulty in the Eulerian description lies in the advective terms, while in the Lagrangian scheme the particles carry out the fluid quantities and the advective terms are intrinsically taken into account. Third, the PIC scheme requires interpolations, each time step, from grid to particles and from particles to grid, when the SPH code calculates the macroscopic quantities directly at the position of the fluid elements. Finally, owing to the variable smoothing length, the resolution is better in the high density regions.

Since the basic principles of the SPH method have been described in the references listed above, we will give only the main results.

Any macroscopic variable G can be evaluated as:

$$G(\mathbf{r}) = \sum_i m_i (G(\mathbf{r}_i)/\rho(\mathbf{r}_i))\, w(|\mathbf{r} - \mathbf{r}_i|, h_i), \tag{2.1}$$

where $\mathbf{r}_i$ are the vector positions of a set of N particles, w is the smoothing kernel, and h_i is the smoothing length; h_i is a function of $\mathbf{r}_i$ and then of the density. Also the derivatives of the fluid quantities are definable in terms of ∇w, so

$$\nabla(\rho G) = \sum_i m_i G(\mathbf{r}_i)\, \nabla w(|\mathbf{r} - \mathbf{r}_i|, h_i), \tag{2.2}$$

where ρ is the estimate of the density:

$$\rho(\mathbf{r}) = \sum m_i w(|\mathbf{r} - \mathbf{r}_i|, h_i). \tag{2.3}$$

Using this formalism we now write the equation of motion for the ith particle in the form:

$$d\mathbf{v}_i/dt = -(\nabla P/\rho)_i - \nabla_i \Phi + (\mathbf{F}_i)_{\text{visc}} \tag{2.4}$$

with $(\nabla P/\rho)_i = \sum_j m_j P_j/\rho_j \rho_i \nabla_i w(r_{ij}, h_j)$.

Here $r_{ij} = |\mathbf{r} - \mathbf{r}_i|$, Φ is the external potential (gravitational, for example) and $(\mathbf{F}_i)_{\text{visc}}$ is the term due to natural and artificial viscosity. To ensure better conservation of the momentum and energy, Gingold and Monaghan [6] have rewritten the pressure terms,

$$-\sum_j m_j (P_j/\rho_j^2 + P_i/\rho_i^2) \nabla_i w(r_{ij}, h_j) \tag{2.5}$$

which can be further symmetrised by taking $h_{ij} = (h_i + h_j)/2$ (see [4, 9]), to satisfy the action reciprocity principle.

However, in the problems that we have treated, the difference with the direct terms (more economic in CPU) was not significant, and we did not use this complex form. Indeed, we have tested the *local* conservation of momentum by checking that:

$$d\mathbf{v}/dt = -\nabla P/\rho - \nabla\Phi + \mathbf{F}_{\text{visc}}, \tag{2.6}$$

i.e., by checking the accuracy of the Eulerian equation of motion, when all terms are estimated by the SPH smoothing procedure, in other words by averaging all physical quantities over all neighbours. If momentum was not conserved because of asymmetries between h_i and h_j, Eq. (2.6) would not be satisfied, even if all particles were advanced according to the Lagrangian equation (2.4).

The tests have been performed at 1D on the isothermal shock problem and in 2D with the Gaussian density distribution (cf. discussion in Section 3). Equation (2.6) was verified within 1%. This can be explained by our determination of h_i (Section 4), which leads a slowly varying h over the spatial resolution of the simulations.

We will go back to the smoothing length and artificial viscosity in Section 4.

Algorithms of the Vectorisation

The equations of Section 2 show that the pressure and force calculations are completed by summations over neighbours, with weighting W. Since the average number of neighbours (around 21 with our kernel W_3, see Section 4) is small, it is not efficient to vectorise these summations directly. As the number of particles N is of the order of 10^4, the summations using vectors of size N will be much more gratifying. The direct method would consist of, for each given particle, a search for its neighbours and then calculation of the corresponding physical quantities. The method presented here proceeds in the reverse way: it is necessary to define in advance the neighbours of each particle and to construct arrays storing all physical quantities corresponding to these neighbours.

To optimise the search of neighbours, we proceed in several steps. In the first step we superpose a grid on the ensemble of particles, to locate each particle, and then to select its potential neighbours in the nearest cells. The number of potential neighbours is maintained at ≈ 50 by adjusting the number of surrounding cells and/or the cell size. On these potential neighbours we can then compute the inter-particle distances and determine the true neighbours. Once the neighbours have been determined for each particle, one simple method would be to sum over the same number of neighbours (i.e., $N_{\text{neighbours}}$ max). But the problem is the large range in the number of neighbours, which goes from 2 to 40. This implies a lot of unnecessary calculations, namely for the particles which have a few neighbours. Therefore we will sort particles in groups, according to their number of neighbours. The details of the group dispatching are described in Section 3 (steps 1, 2, 3).

3. Detailed Description of the Algorithm

To optimise the search for neighbours, a grid is superposed on the system, the cell of which is chosen to be the maximum size of a particle ($h_{\max}$). Pre-gathering particles in groups avoids the N^2 tests on the relative distance of each couple of particles. The principal stages of the algorithm are the search for the neighbouring particles (which decomposes in a search for every particle in a given cell, chained in a linked list, and the calculation of true neighbours) and the summation over these ones (with CPU time proportional to $N_{\text{neighbours}} \times N$).

The current scalar scheme is then:

— **The linked lists.**

```
C  Loop over the number of particles
         DO 1J = 1, N
                  LL = L(J)
                  MM = M(J)
C  if first particle found, particle J can be chained in 4
                  IF (NUM(LL, MM).NE.0) GO TO 4
C  if first particle not found, J is the first
                  NUM(LL, MM) = J
                  GO TO 1
C  search for the first null element in ICHAIN to place J
   4              K = NUM(LL, MM)
   6              KP = ICHAIN(K)
                  IF (KP.EQ.0) GO TO 8
                  K = KP
                  GO TO 6
   8              ICHAIN(K) = J
   1     CONTINUE
```

where L and M are the coordinates of the cell containing the particle J, NUM(L, M) the number of the first particle found in the cell (L, M), and ICHAIN the array of numbers of successive particles of the same chain (i.e., found in the same cell).

— **The search for the neighbours** in the nine surrounding cells including the cell of the particle and the calculation of the macroscopic quantities (here $\rho = \text{den}$):

```
          DO 23J = 1, N
   23     DEN(J) = 0
C  loop 2 is over the number of particles
          DO 2J = 1, N
                   LX = L(J) + 1
                   LN = L(J) - 1
                   MX = M(J) + 1
                   MN = M(J) - 1
C  loop 21 is over the surrounding cells
                   DO 21 LL = LN, LX
                   DO 21 MM = MN, MX
                          KT = NUM(LL, MM)
C  If no particle in the cell it is not necessary to calculate W and DEN
                          IF (KT.EQ.0) GO TO 21
   22                     CONTINUE
C  Calculation of W(KT, J)
                          ---------------
                          ---------------
C  calculation of the density
                          DEN(J) = DEN(J) + W(KT, J)
C  Use of the chaining to find the neighbours of J and sum
                          KP = ICHAIN(KT)
                          IF (KP.EQ.0) GO TO 21
                          KT = KP
                          GO TO 22
   21              CONTINUE
   2      CONTINUE
```

Another more time-effective scalar scheme is given in the Appendix, but this one is interesting because it needs less memory. These two loops are not vectorisable and our first goal is to suppress the GO TO instructions. The solution to vectorise lies in the calculation in advance of the neighbours of a given particle and then the second loop will be replaced by the scheme 1:

```
          DO 10J = 1, N
                   DO 20 KT = 1, NNEIB(J)
                   --physical values calculation----------
   20              CONTINUE
   10     CONTINUE
```

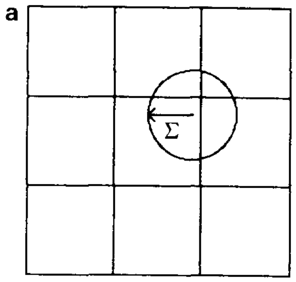
where NNEIB(J) is the number of neighbours of the particle J. Then our second goal is to reverse the loops in order that the inner loop be longer, because only the inner one is vectorised. We have to replace the natural scheme 1 by scheme 2, since N is much greater than NEIB, the maximal number of neighbours:

```
      DO 20 KT = 1, NEIB
                 DO 10J = 1, N
      -------------------------
10               CONTINUE
20    CONTINUE
```

But this scheme involves unnecessary calculations and a loss of CPU time of a factor ≈ 10. In the following we describe a method to circumvent the problem.

The solution is to put the particles in groups, each group being characterized by an approximate number of neighbours. We can also distinguish three steps:

1. the determination of the particles locations, a first sorting according to the number of particles per cell, and the search for the closest particles for a given one.



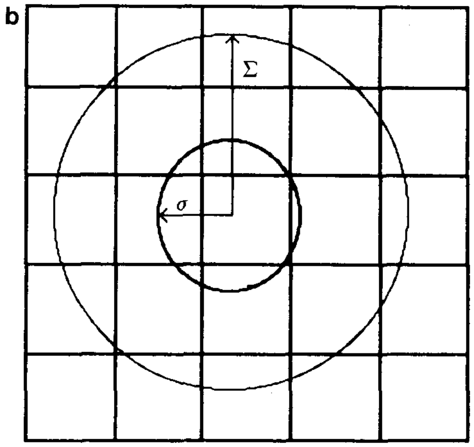


FIG. 1. (a) In case the cell size is $2h_{\max} = \Sigma$ the possible neighbours are located in the nine surrounding cells. (b) In case the search is performed per block the cell size is $\frac{1}{2}h_{\max} = \frac{1}{4}\Sigma$. For the given particle (size σ) the search in the 3×3 block is sufficient.

2. a second sorting according to the approximate number of neighbours

3. a third sorting according to the real number of neighbours, because only particles located at a distance smaller than h have a nonzero contribution to the macroscopic quantities calculated at the position of the ith particle.

Steps 1 *and* 2

To locate the particles we use a 2D grid whose cell size is the maximum diameter $2 * h_{\text{max}}$. In this case the possible neighbours are located in the nine surrounding cells of a given particle (see Fig. 1a). However, since one of the main advantages of this code lies in its variable resolution, the size h of the particles can be very different; one can therefore search for the neighbours in smaller areas such as 3×3 blocks and 5×5 blocks with a block size of $1/2 * h_{\text{max}}$, so that the searches in the 3×3 blocks and in the 5×5 blocks are sufficient in mean density regions, and a search in the 3×3 block only in high density regions (see Fig. 1b). For steep density gradients it can be better to use 7×7 blocks with the appropriate value of the size cell.

After having calculated the location of the jth particle $L(J)$ and $M(J)$, the number of particles per cell N_1 CEL(L, M), the particles are sorted in NGROUP groups, according to their values of N_1 CEL(L, M). For the understanding of the following it is necessary to detail the way the sorting is done. Let us define NEARM(II) as the maximum number of particles in one cell for the (II-1)th group. Let us define also NINTG(II) as the integrated number of particles over all groups preceding the II group, i.e., $\sum_k$ Number(k) for $k \leqslant$ II-1, where Number(k) is the number of particles in the kth group. Since particles are attributed to new numbers after the group sorting, the label and the rank of any particle in a group are the same. Then we start the search per block:

```
C  loop 1 is over the group number
   DO 1 II = 1, NGROUP
C  definition of the first and last particles in group II
              N1 = NINTG(II) + 1
              N2 = NINTG(II + 1)
C  maximum number of neighbours in IIth group
              M2 = NEARM(II + 1)
C  loop 2 is over the number of neighbours
              DO 2I = 1, M2
C  loop 3 is over the particles
                     DO 3J = N1, N2
                            LL = L(J)
                            MM = M(J)
                            K = NPBLOC(J)
C       BEGINNING OF THE SEQUENCE
```

```
C  study of the cell LL - 1, MM - 1
                              ------------------------
C  if there is a Ith particle (KT) in the cell, it is the (k + 1)th potential
C  neighbours of J
                              KT = ITPCEL(LL - 1, MM - 1, I)
                              IF (KT.NE.0) THEN
                              K = K + 1
                              ITBLOC(J, K) = KT
                              ENDIF
                              ------------------------
C       END OF THE SEQUENCE
                              ------------------------
C  new value of the number of particles in group J
                              NPBLOC(J) = K
   3                  CONTINUE
   2            CONTINUE
   1      CONTINUE
```

NPBLOC(J) and ITBLOC(J, K) are respectively the number of particles and the Kth particle number in the Jth particle block, and ITPCEL(L, M, K) the number of the Kth particle in (L, M) cell. The sequence shows the principle of the search for the cell (LL − 1, MM − 1), and it will be enough to repeat it over all the cells to unroll the loop. The inner loop is also longer and is completely vectorised.

The number of particles per block depends on the density distribution. To give an idea we fix the number of particles at $N \approx 3000$. For a constant density NBLOC lies between 7 and 50 with a mean value of 29, and for a density decreasing exponentially from the center, NBLOC lies between 2 and 200 with a mean value of 80.

The particles are then arranged in 9 or 10 groups according to the numbers of neighbours in the blocks, with 300 to 1000 particles per group.

Step 3

After having found the possible neighbours we have to search more exactly for the true neighbours, by carrying out the inter-particle distance $R(I, J)$ calculations inside each group. The second sorting groups have the same order as the first sorting ones and use the NINTG2 and NEARM2 arrays, but NEARM2 here is the maximum number of particles in the blocks occupied by the particles of the (II − 1)th group:

```
C  loop 1 is over the group number
   DO 1 II = 1, NGROUP
C  definition of the first and last particles in group II
               N1 = NINTG2(II) + 1
               N2 = NINTG2(II + 1)
```

```
C  maximum number of neighbours in group II
             M2 = NEARM2(II + 1)
C  loop 2 is over the number of neighbours
             DO 2I = 1, M2
C  loop 3 is over the particles
                    DO 3J = N1, N2
C  if the Ith particle in the bloc of J is close enough to J then it is the
C  (NNEIB + 1)th true neighbour
                           KT = ITBLOC(J, I)
                           IF (R(J, I).LT.H(KT)) THEN
                           NNEIB(J) = NNEIB(J) + 1
                           K = NNEIB(J)
                           ITNEIB(J, K) = KT
                           ENDIF
   3                CONTINUE
   2         CONTINUE
   1      CONTINUE
```

NNEIB(*J*) is the number of neighbours of particle *J*, and ITNEIB(*J*, *K*) is the label of the *K*th neighbour of particle *J*.

The particles are then sorted into new groups according to the real number of neighbours. After this the number of particles per group lies between 6 and 40 with a mean value of 21 for the two density distributions.

Discussion

The three sorting steps are necessary bnt play different roles. Without the first ordering the external loop in the blocks calculation would be performed over the maximum number of particles in a cell. Both other sortings are used because the calculations of the inter-particle distances and the comparison with the particle size can be penalizing if the calculations are carried out over too many particles. The second sorting also reduces the number of particles used to carry out the third ordering. The third sorting groups have the same order as the first sorting ones and use the NINTG3 and NEARM3 arrays, but NEARM3 here is the maximum number of neighbours in the (II − 1)th group. The structure of a vectorisable and optimized loop for the density calculation is then:

```
C  loop 10 is over the group number
   DO 10II = 1, NGROUP
C  definition of the first and last particles in the group
              N1 = NINTG3(II) + 1
              N2 = NINTG3(II + 1)
C  maximum number of real neighbours in the group
              M2 = NEARM3(II + 1)
C  loop 20 is over the number of real neighbours
              DO 20I = 1, M2, 2
```

```
C  loop 30 is over the particles
                        DO 30 J = N1, N2
                             K = I + 1
C  summation to calculate the density
                             DEN(J) = DEN(J) + W(J, I) + W(J, K)
   30                   CONTINUE
   20             CONTINUE
   10       CONTINUE
```

NGROUP lies between 9 and 10, and the inner loop is the longest. We have unrolled by a factor 2 ($K = I + 1$) the loop 20 to optimize the calculations.

Between scalar and vectorial runs of the vectorized code we have observer a factor 11 in the CPU time, and a factor 6 between the vectorised and scalar codes. Figure 2 gives the variation of the CPU time versus the number of particles for the three cases (vectorised scheme with vectorisation ON and OFF and scalar scheme). Note that the N-body subroutine is not included in the evaluation of the total CPU time, but it is only of the order of 3% (therefore its dependence on N would have been washed out by that of the hydrodynamic code). As regards CPU time variation versus the particles number, a proportionality has been found (i.e., CPU time αN). This is due to the constant number of neighbours necessary to ensure good accuracy and to the method that only makes useful calculations. Indeed, we expect the CPU time to vary as $\alpha N \cdot \langle \text{NNEIB} \rangle$, this is only an order of magnitude since the distribution of potential neighbours may depend on N even if $\langle \text{NNEIB} \rangle$ is fixed.

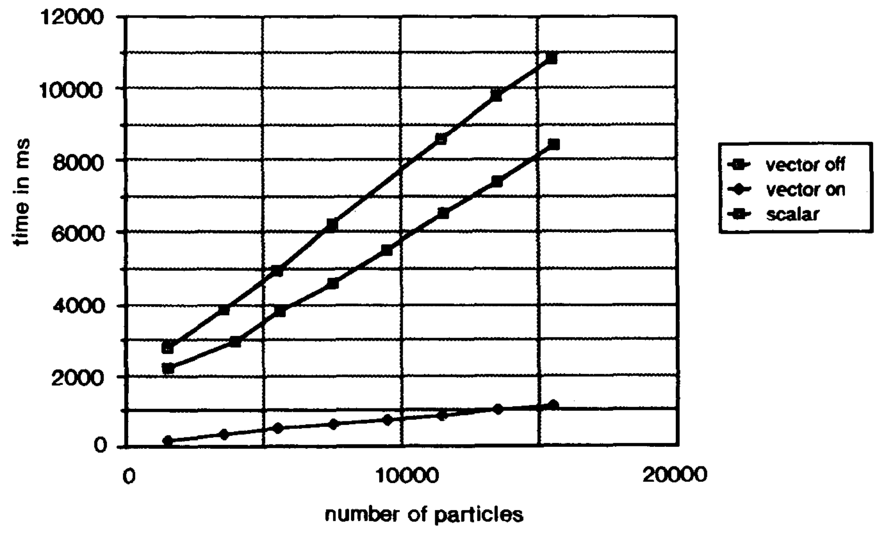


FIG. 2. CPU time per time step versus the number of particles for several schemes: vectorised scheme with vectorisation ON and OFF and scalar scheme. The calculations are performed over the 3×3, 5×5, and 7×7 blocks in the case of the collapse simulation (see Section 5). Note that the N-body subroutine is not included in the evaluation of the total CPU time, but it is only of the order of 3%.

In the vectorised case the slope of the CPU time vs N is significantly lower than one. This can be explained by the vectorised gain at large vector size.

The calculations of the macroscopic variables and the gradients are more efficient (factor 20), while for the search of the neighbours only a factor 5 is observed. This method will be also more economical when the physical model includes a lot of phenomena as magnetic field, radiation transfer, etc.

During the run, the number of groups is adjusted by an automatic procedure. It calculates over three time-steps the number of groups that optimizes the CPU time. The organization of the code is given in Table I. It is interesting to compare the performances of the code for two initial distributions of density. We present in Table II the CPU times per step for calculations performed over 3×3 and 5×5 blocks with uniform and gaussian distributions. For the first one, since the number of neighbours is about the same for all particles it is preferable to use only one

TABLE I

Organisation of the Program

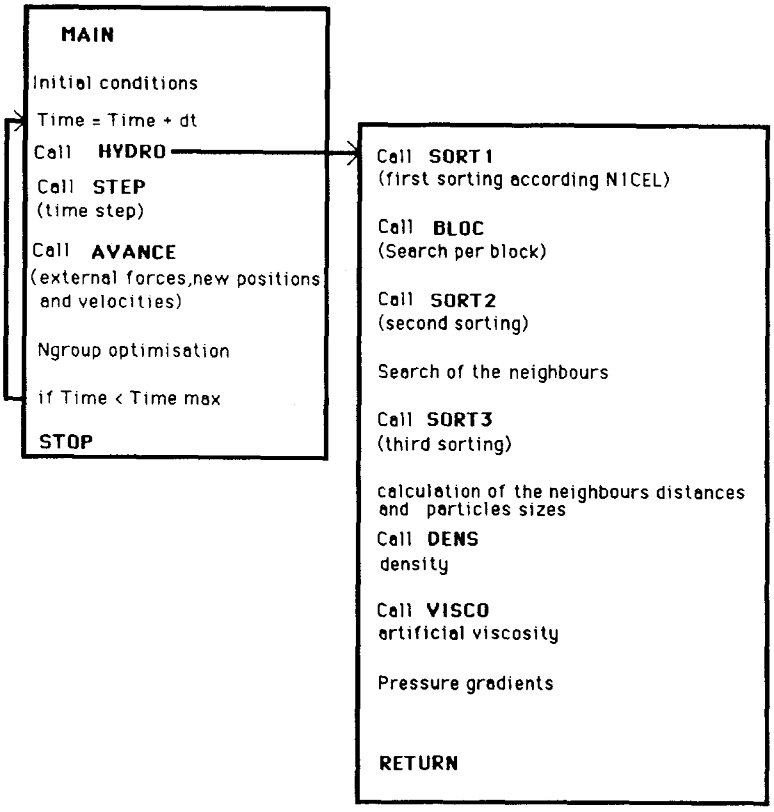


TABLE II

Time (in ms) for Each Step of the Hydrodynamical Part of the Code

	Uniform density			$r = r_0 \exp(-x^2/a^2)$		
	1 group		optimisation	1 group		optimisation
SORT1	13		13	9		9
BLOCK	67		67	41		30
SORT2	30		23	21		11
NEIGHBOURS	66		64	43		26
SORT3	12		11	13		7
DISTANCES	34		32	41		13
DENS	12		12	13		8
VISCO	93		81	105		55
GRADIENT	2		1	1		1
Total	329		304	287		160
Ratio		1.1			1.8	
Total without sorting	274			244		
Ratio		0.9			1.5	

Note. The tests are performed over two density distributions using 3980 particles, a search over the 3×3 and 5×5 blocks and with the Gingold–Monaghan viscous tensor. For the exponential function the simulation window is $[-7, +7]$ while the half intensity width is 3.5. We give the total CPU time with and without sorting for the case where the group's number is 1, for which the sortings are useless.

group, while for steeper density gradient the gain is more important with the optimised number of groups. Moreover, the program is more efficient for the gaussian distribution.

4. Interpolation Kernels

The Kernel

Each element of fluid is described by a smoothed out distribution of density, by using an interpolation function $w(r, h)$. The construction of the interpolation kernel $w(\mathbf{r}, h)$ is guided by the requirements of accuracy, smoothness and computational efficiency [15] and then different functions have been used. Unlike the exponential function $w(r, h) = 1/8\Pi h^3 \exp(-r/h)$ chosen by Wood [19] that has nonzero derivatives in $r = 0$ and then overestimates the self-contribution of a particle, we prefer to build a kernel by

$$(\partial \mathbf{w}(r, h)/\partial r)_{r=0} = 0 \tag{4.1}$$

$$(\partial \mathbf{w}(r, h)/\partial r)_{r=h} = 0 \tag{4.2}$$

$$\mathbf{w}(r, h)_{r=h} = 0 \tag{4.3}$$

and, of course,

$$\int^{h} 2\pi w(r, h) r\, dr = 1. \tag{4.4}$$

Several authors use a Gaussian function $\exp(-r^2/h^2)$ or a modified Gaussian function $\exp(-r^2/h^2) * (3/2 - r^2/h^2)$ [7, 15], but they are very smooth and they require a neighbours search over $3h$, so that we have chosen polynomial functions that are nonzero only in a finite domain. Then the summation over all particles is done only over the neighbouring particles. To reach a good interpolation accuracy, a minimum number of neighbours have to be taken into account, depending on the interpolation function. For the second and third degree polynomials w_2 and w_3 given below, a search over h is sufficient with respectively 25 and 21 neighbours, unlike the Gaussian function that requires 50 neighbours over $3h$:

$$\begin{array}{lll} W_2 = 18/(7\pi h^2) & (1-3u^2) & \text{for} \quad u = r/h \leqslant \frac{1}{3} \\ & \frac{3}{4}(1-u^2) & \text{for} \quad \frac{1}{3} \leqslant u \leqslant 1 \\ & 0 & \text{for} \quad u > 1 \end{array} \tag{4.5}$$

$$\begin{array}{lll} W_3 = 40/(7\pi h^2) & (1-6u^2+6u^3) & \text{for} \quad u = r/h \leqslant \frac{1}{2} \\ & 2(1-u)^3 & \text{for} \quad \frac{1}{2} \leqslant u \leqslant 1 \\ & 0 & \text{for} \quad u \geqslant 1 \end{array} \tag{4.6}$$

After testing these functions we have chosen W[illegible] which minimizes the number of neighbours and interpolates with errors not exceedig $O(h^2)$.

The Variable Kernel Size

One of the important points of our scheme lies in the variable kernel size h. Some authors [19, 20, 1, 14, 4] have used a variable h that allows a variable resolution, without the complex use of windows with varying cell size as in grid schemes. Moreover, the size of a fluid element adapts itself automatically according to the local density. The relationship between h and ρ is then [6]:

$$h^2 = K/\rho \qquad \text{where} \quad K \text{ is a constant.} \tag{4.7}$$

To optimize the accuracy of the calculations the following iterative procedure can be used:

$$\begin{tikzcd} h^2 = K/\rho \arrow[r] & \text{calculation of } w \arrow[r] & \text{calculation of } \rho \arrow[ll, bend left] \end{tikzcd}$$

It converge for the K values that optimize the calculations. Another method can be used, such as the determination of the variance $(h^2 = \langle \Delta \mathbf{r}^2 \rangle - \langle \Delta \mathbf{r} \rangle^2)$ over the particles in a block. However, when the density changes rapidly in time, the h value can be extrapolated from the previous time steps. In particular, a too rapid change in the h values may be a source of instabilities so that it is better to limit the variation to a few percent of the previous value at each time step. However, this temporal condition does not guarantee smooth spatial variations of h.

Therefore, we use a better method which suppresses the instabilities due to the variable h. This method consists in calculating the average density at the position of a particle and then taking $h^2 = K/\rho_{\text{mean}} \cdot \rho_{\text{mean}}$ is computed by averaging ρ^2 over neighbours with the SPH algorithm, i.e., according to $\rho_{\text{mean}} = (1/\rho) \sum_i m_i \rho(\mathbf{r}_i)\, w(|\mathbf{r} - \mathbf{r}_i|, h_i) = \langle \rho^2 \rangle / \rho$.

This method limits the variation of h, such that the equations of motion (2.4) are still valid in their simple form, without any inclusion of terms proportional to the derivatives of h (∇h and $\partial h/\partial t$ [6]). Evrard [4] has detailed analytically the required conditions for these added terms to be negligible, with a kernel size depending on the local density and concluded that h must vary on scales "a" much larger than h: grad h/grad $r \alpha h^2/ra \ll 1$, where r is the mean interparticle distance.

The Artificial Viscosity

For all the schemes used to simulate a fluid a crucial test lies in the shock simulation. Indeed a real shock occurs on the scale of the order of the mean free path of the fluid, which is much smaller than the resolution of the simulation. It is then necessary to smooth the density distribution over larger scales, such that the shock extends over a few resolution lengths. When applying the particle method it is convenient to have recourse to artificial viscosities that can be easily included in the equations of fluid. Using our method and our kernel we have tested several artificial viscosities proposed by Monaghan and Gingold [7] (1983) and a linear combination (CL) of the first such problems introduced:

The Neumann–Ritchmyer (NR) viscous pressure:

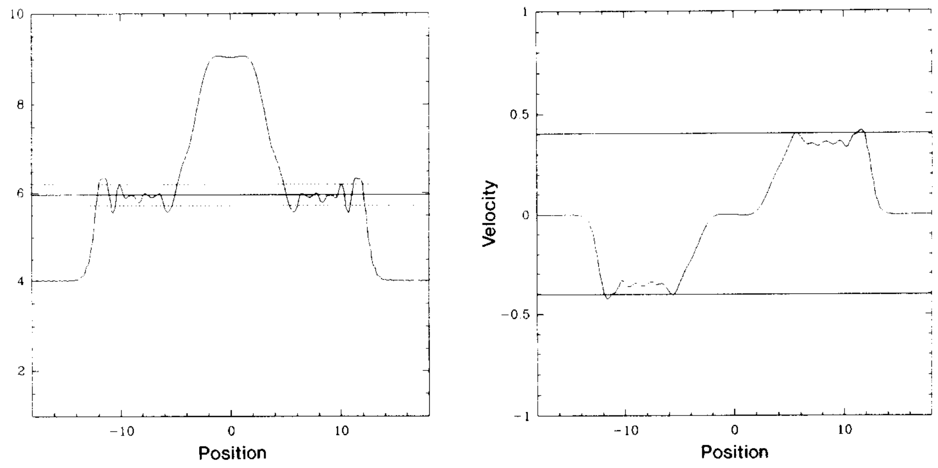
$$\begin{aligned} q &= \alpha\rho h^2(\nabla\cdot\mathbf{v})^2 && \text{for } \nabla\cdot\mathbf{v}<0 \\ &0 && \text{for } \nabla\cdot\mathbf{v}>0 \end{aligned} \tag{4.8}$$

The Bulk (B) viscosity:

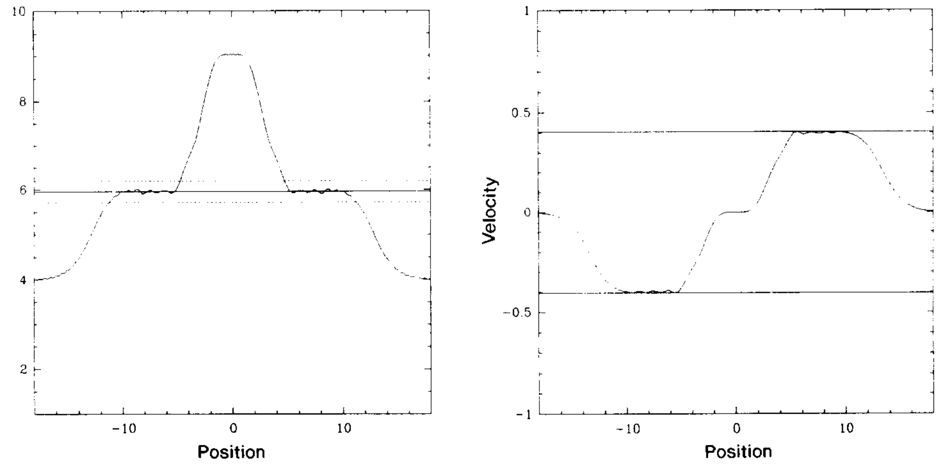
$$\begin{aligned} q &= -\alpha\rho h c(\nabla\cdot\mathbf{v}) && \text{for } \nabla\cdot\mathbf{v}<0 \\ &0 && \text{for } \nabla\cdot\mathbf{v}>0 \end{aligned} \tag{4.9}$$

FIG. 3. (a–f) Density and velocity profiles of an isothermal shock at the time $5C_s^{-1}$ (where C_s is the sound speed) with respectively no viscosity, Bulk viscosity ($\alpha = 0.8$), Neumann–Richtmeyer viscosity ($\alpha = 0.8$), Gingold–Monaghan viscous tensor ($\alpha_1 = 0.8$, $\alpha_2 = 0$, and $\beta = 0.1$), ($\alpha_1 = 0.8$, $\alpha_2 = 0.8$, and $\beta = 0.1$), linear combination.

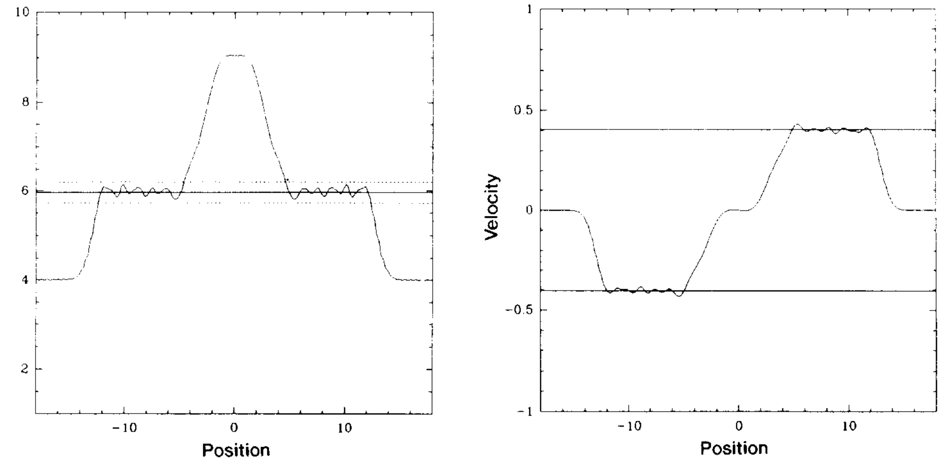
a


Density
Velocity
Position
Position
10
8
6
4
2
1
0.5
0
−0.5
−1
−10
0
10

b


Density
Velocity
Position
Position
10
8
6
4
2
1
0.5
0
−0.5
−1
−10
0
10

c


Density
Velocity
Position
Position
10
8
6
4
2
1
0.5
0
−0.5
−1
−10
0
10

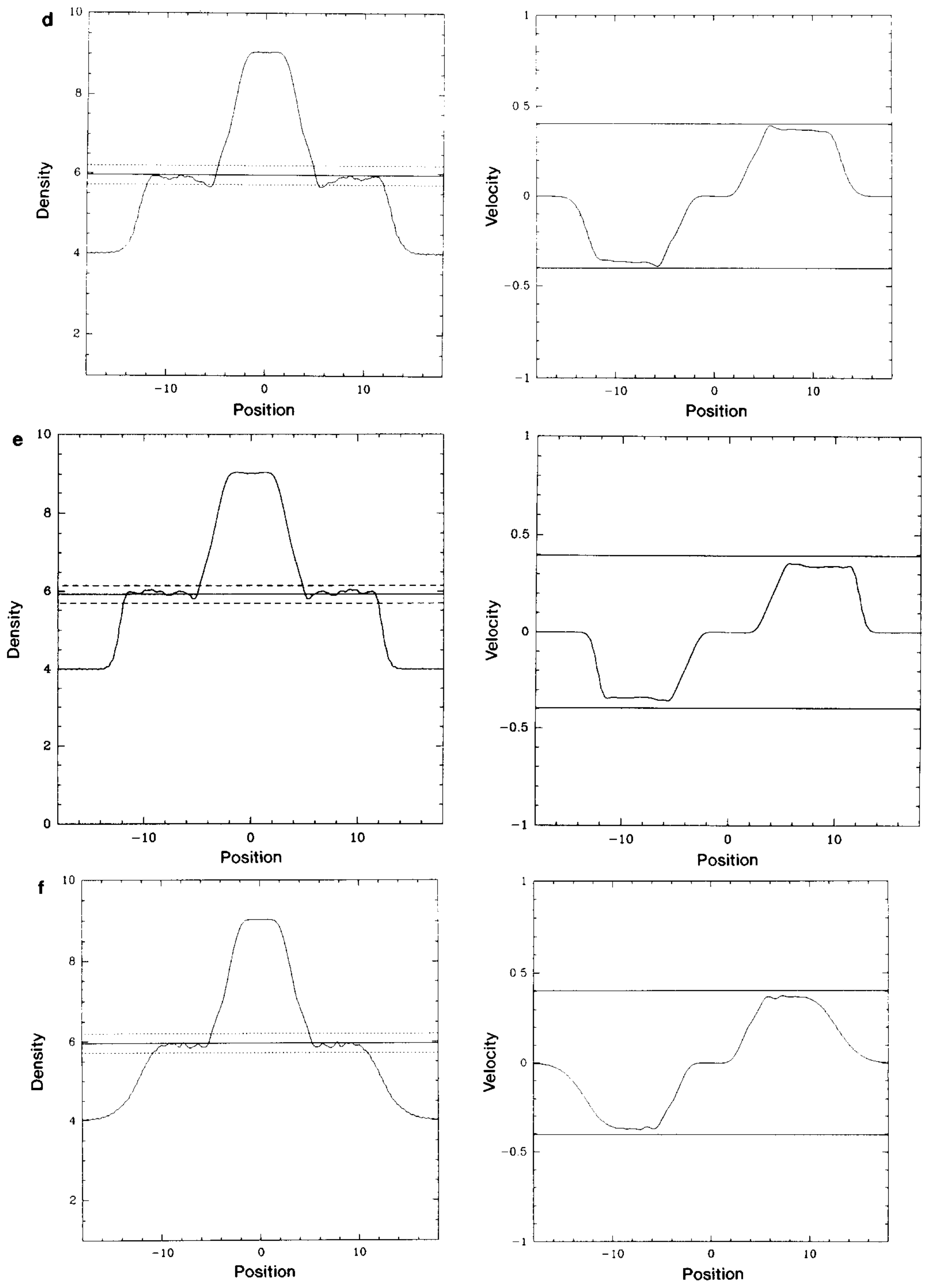


FIG. 3—*Continued*

where α is a constant, c is the sound speed, $\mathbf{v}$ is the fluid velocity, and ρ is the density.

The Gingold–Monaghan (GM) viscous tensor: (4.10)

$$\Pi_{ij} = -c_{ij}^2/\rho_{ij}(\alpha_1 q_{ij} + \alpha_2 q_{ij}^2)$$

with

$$\begin{aligned} q_{ij} &= r_{ij}v_{ij}/(h_{ij}c_{ij})/(r_{ij}^2/h_{ij}^2 + \beta) && \text{for} \quad v_{ij}r_{ij} < 0 \\ q_{ij} &= 0 && \text{for} \quad v_{ij}r_{ij} > 0, \end{aligned}$$

where $0 < \beta \ll 1$ is a constant and for any function G; we define

$$G_{ij} = (G_i + G_j)/2. \tag{4.11}$$

Π_{ij} is such that

$$(\mathbf{F}_i)_{\text{visc}} = -\sum_j m_j \Pi_{ij} \nabla_i w(r_{ij}, h_j). \tag{4.12}$$

We then consider an isothermal shock problem for a perfect gas analogous to that considered by Leboeuf *et al.* [11]. The initial state consists of a high density plateau surrounded by a region where the density is lower. The analytical solutions can be calculated only for 1D shocks so that we choose an initial symmetry—the density depends only on the x variable—that leads to an 1D solution, although the calculations are carried with a 2D code:

$$\begin{aligned} &x < -7 \text{ or } x > 7 && \rho = 4 \\ &-7 < x < 7 && \rho = 9. \end{aligned}$$

The results for each artificial viscosity are compared in Figs. 3a to 3f. The conclusion regarding the advantages of a given viscosity presents several differences from those with the shock tube problem (adiabatic) considered by Gingold and Monaghan [7] and Sod [18]. The first representation shows, for the case without artificial viscosity, the amplification of the oscillations that can lead to numerical divergences. The problem is solved by artificial viscosity as is shown by the other figures. For this type of shock the NR viscosity does not suppress all the oscillations. The density and velocity profiles are more smoothed by the B than by the GM, but the GM leads systematically to underestimated velocity values. The linear combination is comparable to the GM for the smoothing and to the B for the accuracy of the density values. All tests are carried out with variable size kernels. The various profiles are given for $y = 0$, but for finite y the results are comparable; i.e., there is no movement in the y-direction except in the neighbourhood of the boundaries. Moreover, the same calculation was performed with inversion of the x and y variables and the equivalence of the two directions was checked. We conclude that for isothermal shocks the CL or B viscosities can be used although the CL underestimates the velocity, and with an important smoothing for the B, while for adiabatic shocks the GM leads to better results.

3D Extension

It is straightforward to extend the method in three dimensions, but this will be done at the expense of memory and time. This is mainly due to the necessary increase of the number N of particles, to keep a performant spatial resolution. Within the same N, we can, however, estimate the necessary increase of memory for

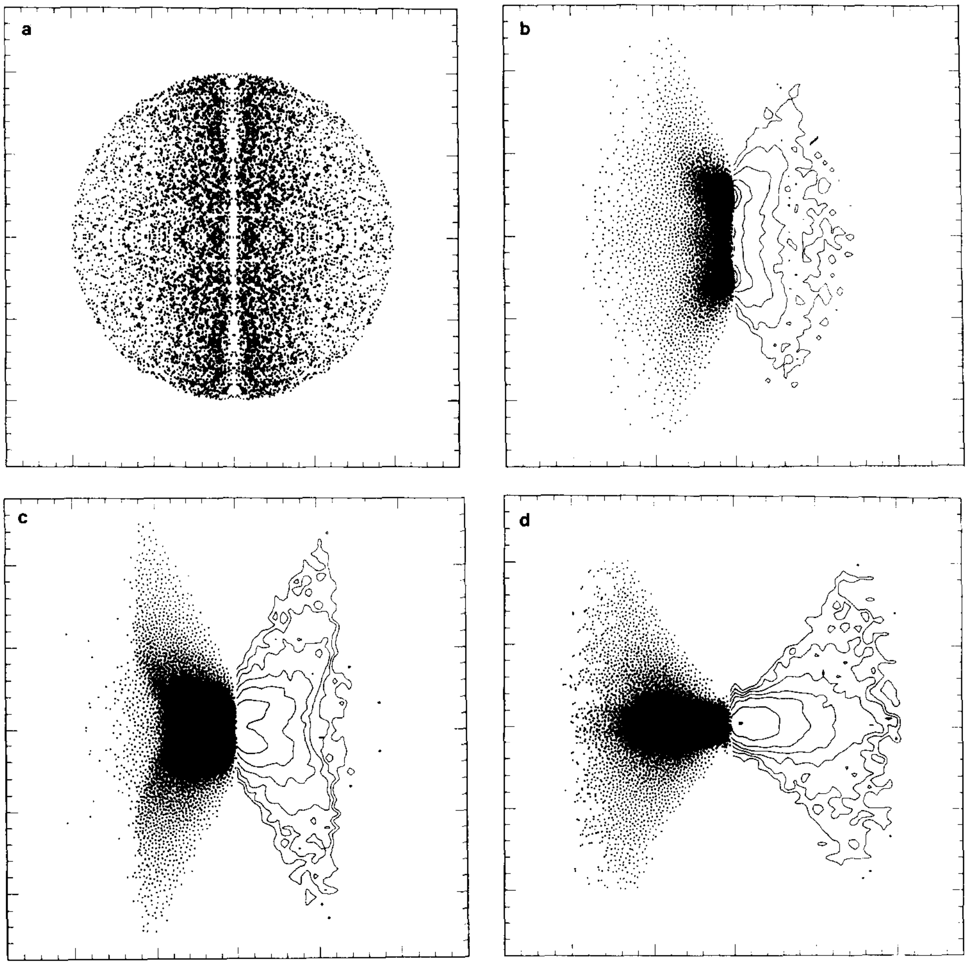


FIG. 4. (a–d) Non-rotating cylindrical cloud [10]. (a) shows the initial distribution of particles. Figures 4b–d on the left side show the positions of the fluid particles and, on the right side, the integrated density levels (i.e., $2\pi\rho R$) with 5000 particles, at time $t = 2.3 \times 10^{12}$ s and for temperatures equal respectively to 7.5K, 10K, and 13.5K.

a 3D simulation. The array that would be considerably extended is ITPCEL, which would gain one dimension: the necessary memory is multiplied by the number of cells in the grid (here ≈ 20–40), but this array occupied less than $\frac{1}{40}$th of the total, in 2D, with $N \approx 10^4$. One of the biggest array is ITBLOC, which has no reason to vary. Therefore, the memory increase in 3D, due only to physical vectors (like the positions of the particles) is not a major penalizing factor.

5. Isothermal Collapse of an Axially Symmetric Cloud

To compare our method with other hydrodynamic codes we have simulated isothermal collapes of axially symmetric clouds. The calculations are performed in a meridian plane using cylindrical coordinates r, θ, z, where the azimuthal coordinate θ does not appear explicitly because of the assumption of axial symmetry: each particle represents a small toroidal element. Accordingly, the Lagrangian equations are given below in cylindrical coordinates.

$$D\rho/Dt + \rho\, \nabla \cdot \mathbf{v} = 0 \tag{5.1}$$

$$Dv_r/Dt + \nabla_r P/\rho + \nabla_r \phi - A^2/(r^3\rho^2) = 0 \tag{5.2}$$

$$Dv_z/Dt + \nabla_z P/\rho + \nabla_z \phi = 0 \tag{5.3}$$

$$D(\rho A)/Dt + \rho A\, \nabla \cdot \mathbf{v} = 0. \tag{5.4}$$

Where ρ, P, A, ϕ are respectively the density, pressure (thermal and viscous), specific angular momentum, and gravitational potential. The Poisson equation for the gravity is solved by a FFT method. Of course this method is not optimal for variable resolution calculations, since gravity is computed on a grid, which is fixed spatially for all the fluid. In the course of the collapse, however, the total grid has been adjusted to the total extension of the fluid, i.e., the spatial resolution evolves toward smaller grid size with time. We think that the spatially variable resolution is much more important for the fluid hydrodynamics, since it is based on calculations over the sole neighbours, than for the long-range gravitation forces. We use this method as a comparative test of the program, and we confer the reader to gridless methods such as the tree code to solve this problem [9] . The latter, however, consume much more computing time.

More precisely, we solve the N-body interactions in 2D, since the 3D problem is axisymmetric. The particles represent tori all with the same vertical axis Oz, and the same mass. The convolution of density and gravitational potential is transformed in a simple product in the z-direction only. A direct convolution is made in the r-direction.

The potential between particles of coordinates (r, z) and (r', z') is that between two tori of radii r and r' and altitude z and z':

$$\Phi(r, r', z, z') = G \int d\theta/\pi[(z-z')^2 + r^2 + r'^2 - 2rr' \cos\theta + a^2]^{-1/2}, \tag{5.5}$$

where a is the softening length (the gravitational potential is softened from the Newtonian law in $1/r$, by $1/\sqrt{r^2+a^2}$, to compensate for the restricted number of particles; a is of the order of the cell size). The potential law (5.5) is computed and tabulated once at the beginning of the simulation.

The dimension of the grid is 64 in radius, and $Nz = 256$ in z: indeed in the FFT method, images must be avoided and the useful grid is only half (Nz useful $= 128$) of the total grid. The linear dimensions of one cell in z and r are therefore equal. The CPU time for the FFT varies as $N_c \log N_c$, where N_c is the cell number. This has not been varied here. Besides, the forces computation varies linearly with N, the

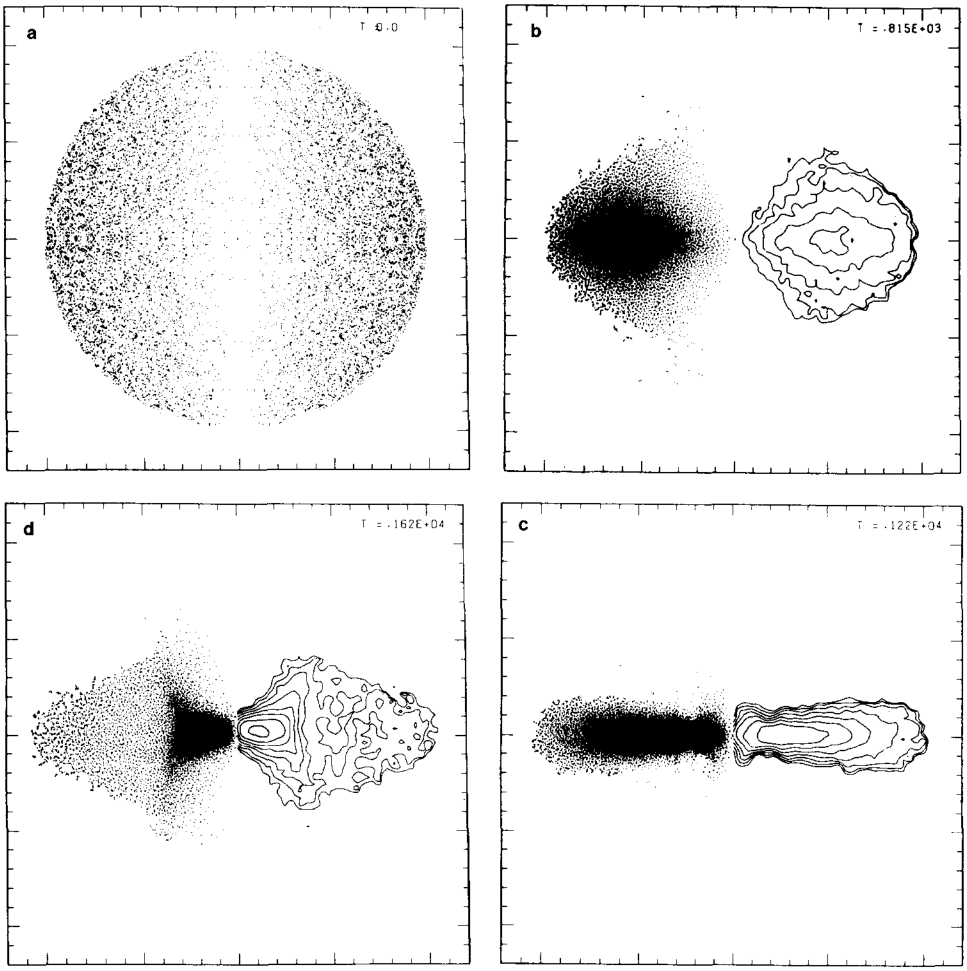
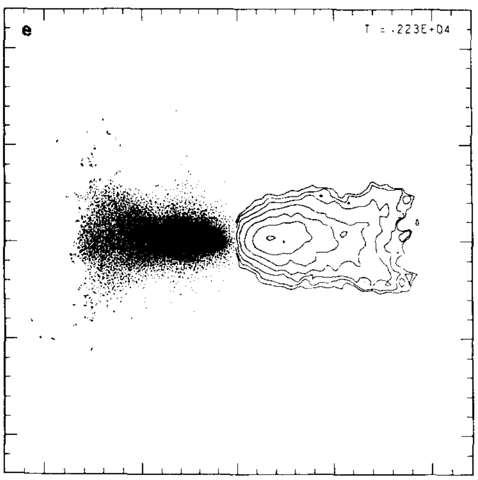
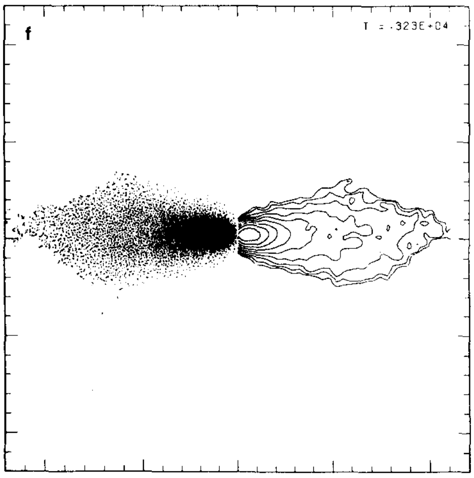


FIG. 5. (a–g) Rotating spherical cloud with uniform initial density. (a) shows the initial distribution of 6000 particles; Figs. 5b–g show the evolution of the collapsing cloud. The times are given in program units (i.e., 127 years). It is possible to distinguish the several steps: (b)–(c), contraction along the Z axis; then collapse in the radial direction and formation of a ring.





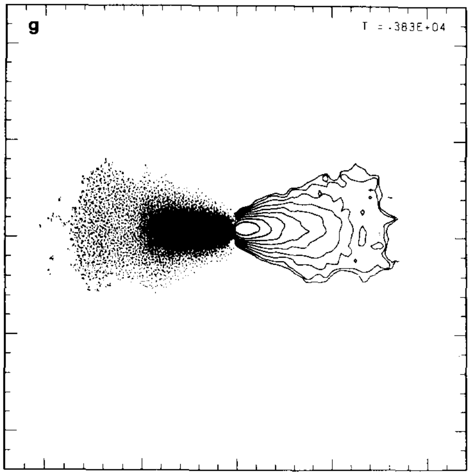


FIG. 5—*Continued*

number of particles. For $N = 15{,}000$, the contribution of the CPU time is 34 ms (3%).

Two types of initial conditions have been taken. The first one consists in a non-rotating cylindrical cloud with the same initial conditions as Larson [10] as illustrated in Fig. 4a. (In Figs. 4a and 5a the central hole surrounding the axis is due to the 2D density distribution, because for uniform ρ the 2D density is equal to $2\pi\rho R$, where R is from the axis.) The initial mass density then is

$$\rho(r, z) = 7.8 \times 10^{-18}(1 + (10r/R)^2)^{-1}\ \mathrm{gcm}^{-3}, \tag{5.6}$$

where $R = 10^{17}$ cm is the radius of the cloud, so that the initial 2D density distribution is

$$\mu(r, z) = 2\pi\rho(r, z). \tag{5.7}$$

According to Ostriker [16] and Mestel [13] a cylindrical equilibrium configuration is possible only if the mass per unit length along the cylinder has the value:

$$M/L = 2\mathcal{R}T/G, \tag{5.8}$$

where $\mathcal{R}$ is the perfect gas constant. With the assumed initial conditions for $T = 10\text{K}$, the gas pressure and the gravity should nearly balance. Figure 4b shows the density distribution resulting at a typical time in the collapse ($t = 2.3 \times 10^{12}$ s) when a temperature of 7.5K is assumed. In this case the horizontal pressure gradient is insufficient to prevent the horizontal collapse. On the contrary, when a temperature of 13K is assumed (Fig. 4d), the pressure gradient leads to a horizontal expansion and, for the critical temperature $T = 10\text{K}$, an equilibrium is observed (Fig. 4c). But at greater time, the simulation leads in all cases to a vertical collapse.

The second simulation consists in the collapse of a spherical rotating cloud with uniform initial density $\rho = 4.7 \times 10^{-19}\,\text{g cm}^{-3}$ and 6000 particles (Fig. 5a). The same initial conditions as Larson [10] and Black and Bodenheimer [3] have been taken, i.e., a rotation velocity of $0.9\omega_c$, where ω_c is the critical rotation velocity for the collapse given by

$$\omega_c^2 = 5/R^2(0.42GM/R - c_s^2), \tag{5.9}$$

where c_s is the sound speed. Initially the collapse is mainly in the Z-direction, as the cloud is centrifugally supported in the $X-Y$ plane (Figs. 5b–c). Then a strong shock forms in the central region and the cloud collapses in the $X-Y$ plane (Figs. 5d–g). At the end, the formation of a ring is observed (Fig. 5g) at a position comparable to that found by Black and Bodenheimer [3] but the ring is gravitationally unstable and collapses in the end. These results can be compared with Wood's 3D SPH simulations [19] with only 500 particles and therefore low resolution. He found that, finally, a non-axisymmetric mode becomes unstable, which justifies the 3D simulations.

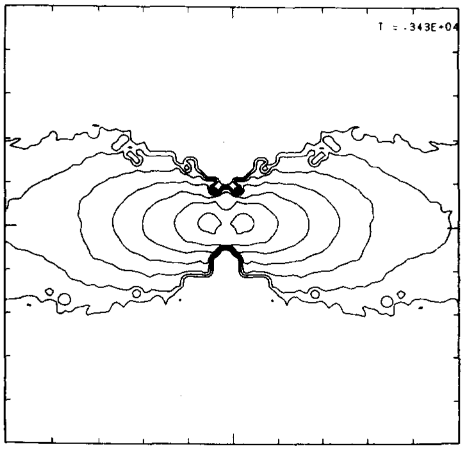


FIG. 6. Density levels resulting from the collapse of the rotating cloud.

6. Conclusion

We have vectorised the SPH algorithm, which now is quite time-performant with respect to other concurrent hydrodynamical codes. The algorithm has the great advantages of being Lagrangian and of simple conception. Also the code introduces only little dissipation, in contrast to grid-dependent algorithms. In our vectorised code, the CPU time grows linearly with the number of particles. This is due to the constant number of neighbours necessary to ensure good accuracy and to the algorithm that suppresses all unnecessary calculations. This enables us to handle many more particles and then to improve significantly the dynamics, which was a weak point of the SPH algorithm, although somewhat cured by the variable resolution. The larger number of particles also allows a correct treatment of the low-density regions, such as the boundaries, which was a second difficulty of the method.

In our code, it is to be noted that the calculations of the macroscopic variables and the gradients are much more efficiently vectorised (factor 20), while for the search of the neighbours only a factor 5 is observed between vectorial and scalar runs. For the whole program, the vectorisation gain is then 6 (≈ 1 for 15,000 particles). Therefore, the code will be much more interesting when the physical model includes a lot of phenomena such as magnetic field, radiative transfer, etc. Our tests and comparisons with other works have proved that the variable resolution gives good results, particulary for the collapse, where the resolution is a crucial problem. Our adjustment of the kernel size h suppresses the instabilities due to its variation and is more fit to density distributions with steep gradients.

Appendix

We present in this appendix a scheme that will be performant on scalar processors.

Calculation of the ITNEIB and NNEIB arrays.

```
C  loop 1 is over the particles
      DO 1J = 1, N
                 L = LL(J)
                 M = MM(J)
C  loop 2 is over the cells of the block
                 DO 2MR = M − NC, M − NC
                 DO 2LR = L − NC, L + NC
                           KT = N1CEL(LR, MR)
C  if no particle in the cell, it is no necessary to make any
C     calculation
                           IF (KT.EQ.0) GO TO 2
```

```
C  calculation of the interparticle distance
3                            DX=X(J)-X(KT)
                             DY=Y(J)-Y(KT)
                             R2=DX*DX+DY*DY
                             H2=H(KT)*H(KT)
C  if KT is close enough to J then KT is the (NNEIB+1)th neighbour of J
                             IF (R2.LT.H2) THEN
                             K=NNEIB(J)+1
                             NNEIB(J)=K
                             ITNEIB(J, K)=KT
                             ENDIF
C  use of the chaining to find the next neighbour
                             KT=ICHAIN(KT)
                             IF (KT.NE.0) GO TO 3
2                   CONTINUE
1    CONTINUE
```

where NC is 1 for the 3×3 block and 2 for the 5×5 one.

Acknowledgment

The calculations have been performed on the VP 200 of the CIRCE (Centre Inter Regional de Calcul Electronique, Orsay).

References

1. W. Benz, *Astron. Astrophys.* **139**, 378 (1984).
2. W. Benz and J. G. Hills, *Astrophys. J.* **323**, 614 (1987).
3. D. C. Black and P. Bodenheimer, *Astrophys. J.* **206**, 138 (1976).
4. A. E. Evrard, *Mon. Not. R. Astron. Soc.* **235**, 911 (1988).
5. R. A. Gingold and J. J. Monaghan, *Mon. Not. R. Astron. Soc.* **181**, 375 (1977).
6. R. A. Gingold and J. J. Monaghan, *J. Comput. Phys.* **46**, 429 (1982).
7. R. A. Gingold and J. J. Monaghan, *Mon. Not. R. Astron. Soc.* **204**, 115 (1983).
8. F. H. Harlow, *Comput. Phys. Commun.* **48**, 1 (1988).
9. L. Hernquist and N. Katz, *Astrophys. J. Suppl.* **70**, 419 (1989).
10. R. B. Larson, *Mon. Not. R. Astron. Soc.* **156**, 437 (1972).
11. J. N. Leboeuf, T. Tajima, and J. M. Dawson, *J. Comput. Phys.* **31**, 379 (1979).
12. L. B. Lucy, *Astron. J.* **82**, 1013 (1977).
13. L. Mestel, *Q. J. R. Astron. Soc.* **6**, 161 (1965).
14. S. C. Miyama, C. Hayashi, and S. Narita, *Astrophys. J.* **279**, 621 (1984).
15. J. J. Monaghan and J. C. Lattanzio, *Astron. Astrophys.* **149**, 135 (1985).
16. J. Ostriker, *Astrophys. J.* **140**, 1056 (1964).
17. R. H. Sanders and K. H. Prendergast, *Astrophys. J.* **188**, 489 (1974).
18. G. A. Sod, *J. Comput. Phys.* **27**, 1 (1978).
19. D. Wood, *Mon. Not. R. Astron. Soc.* **194**, 201 (1981).
20. D. Wood, *Mon. Not. R. Astron. Soc.* **199**, 331 (1982).