GRADSPH: A parallel smoothed particle hydrodynamics code for self-gravitating astrophysical fluid dynamics

S. Vanaverbeke\textsuperscript{a,b,*}, R. Keppens\textsuperscript{b,c,d}, S. Poedts\textsuperscript{b}, H. Boffin\textsuperscript{e}

\textsuperscript{a} KU Leuven Campus Kortrijk, E.-Sabbelaan 53, 8500 Kortrijk, Belgium
\textsuperscript{b} Centre for Plasma-Astrophysics, KU Leuven, Celestijnenlaan 200B, 3001 Leuven, Belgium
\textsuperscript{c} FOM – Institute for Plasma Physics Rijnhuizen, PO Box 1207, 3430 BE Nieuwegein, The Netherlands
\textsuperscript{d} Astronomical Institute, Utrecht University, PO Box 80000, 3508 TA Utrecht, The Netherlands
\textsuperscript{e} European Southern Observatory, Karl-Schwarzschild-Strasse 2, D-85748 Garching, Germany

We describe the algorithms implemented in the first version of GRADSPH, a parallel, tree-based, smoothed particle hydrodynamics code for simulating self-gravitating astrophysical systems written in FORTRAN 90. The paper presents details on the implementation of the Smoothed Particle Hydrodynamics (SPH) description, where a gridless approach is used to model compressible gas dynamics. This is done in the conventional SPH way by means of ‘particles’ which sample fluid properties, exploiting interpolating kernels. The equations of self-gravitating hydrodynamics in the SPH framework are derived self-consistently from a Lagrangian and account for variable smoothing lengths (‘GRAD-h’) terms in both the hydrodynamic and gravitational acceleration equations. A Barnes–Hut tree is used for treating self-gravity and updating the neighbour list of the particles. In addition, the code updates particle properties on their own individual timesteps and uses a basic parallelisation strategy to speed up calculations on a parallel computer system with distributed memory architecture. Extensive tests of the code in one and three dimensions are presented. Finally, we describe the program organisation of the publicly available 3D version of the code, as well as details concerning the structure of the input and output files and the execution of the program.

Program summary

Program title: GRADSPH 1.0
Catalogue identifier: AECX_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AECX_v1_0.html
Program obtainable from: CPC Program Library, Queen’s University, Belfast, N. Ireland
No. of lines in distributed program, including test data, etc.: 11 123
No. of bytes in distributed program, including test data, etc.: 1 561 909
Distribution format: tar.gz
Programming language: Fortran 90/MPI
Computer: HPC cluster
Operating system: Unix
Has the code been vectorised or parallelised?: Yes
RAM: 56 Mwords with 1.2 million particles on 1 CPU
Word size: 32 bits
Classification: 12
Solution method: Hydrodynamics is described using SPH, self-gravity using the Barnes–Hut tree method.
1. Introduction

The basic Smoothed Particle Hydrodynamics (SPH) method was invented in 1977 in two seminal papers by Lucy [1] and Gingold and Monaghan [2] in order to study the fission of rotating stars. At the heart of SPH is an interpolation method which allows any function to be expressed by its values at a set of disordered points, which are referred to as particles. As opposed to more traditional finite difference and finite volume codes, SPH is Lagrangian in nature, and does not need to make use of a fixed grid. In addition, the method concentrates the numerical elements (the particles) where the fluid is at all times, so that it does not waste computational time in empty regions of the computational domain. For given computational resources, SPH is therefore more efficient than grid-based hydrodynamics codes when applied to astrophysical problems involving huge density contrasts and many spatial scales. Because of its Lagrangian nature, SPH can also be combined effectively with tree methods to include self-gravity [3,4]. Although the resolution of shock waves achieved with SPH is still below the performance of advanced shock-capturing methods like PPM [5], the clear advantages of SPH allowed a variety of applications to astrophysical problems, including star and planet formation [6,7], and stellar collisions and mergers [8]. Detailed review articles on SPH can be found in [11,12], and an excellent introductory description is given in the recent book by Bodenheimer and Laughlin [13].

In this paper, we describe our implementation of a parallel SPH code in FORTRAN 90 combined with a Tree Code Gravity (TCG) method for evolving three-dimensional self-gravitating astrophysical fluids. The plan of the paper is as follows. In Section 2, we describe the fundamental principles of Smoothed Particle Hydrodynamics. In Section 3, we discuss the derivation of the SPH equations from a Lagrangian principle, which leads to the introduction of correction terms (‘GRAD-h’) terms in both the hydrodynamic and gravitational acceleration equations. Section 4 describes the treatment of shock waves using time-dependent dissipative terms. The implementation of the neighbour search and the computation of the gravitational acceleration of the particles with a Barnes–Hut method for evolving three-dimensional self-gravitating astrophysical problems, including star and planet formation [6,7], and stellar collisions and mergers [8]. Detailed review articles on SPH can be found in [11,12], and an excellent introductory description is given in the recent book by Bodenheimer and Laughlin [13].

In this paper, we describe our implementation of a parallel SPH code in FORTRAN 90 combined with a Tree Code Gravity (TCG) method for evolving three-dimensional self-gravitating astrophysical fluids. The plan of the paper is as follows. In Section 2, we describe the fundamental principles of Smoothed Particle Hydrodynamics. In Section 3, we discuss the derivation of the SPH equations from a Lagrangian principle, which leads to the introduction of correction terms (‘GRAD-h’) terms in both the hydrodynamic and gravitational acceleration equations. Section 4 describes the treatment of shock waves using time-dependent dissipative terms. The implementation of the neighbour search and the computation of the gravitational acceleration of the particles with a Barnes–Hut tree algorithm are described in Section 5. We then discuss the step-by-step algorithm in Section 6 and details of the parallelisation strategy of the program in Section 7. Extensive numerical tests of the algorithms in one and three dimensions are reported in Section 8. Section 9 provides a reference guide on how to compile and use the public version of the program. Scalability and performance tests are reported in Section 10. Finally, our conclusions are presented in Section 11.

2. Basic equations of Smoothed Particle Hydrodynamics

The evolution of a self-gravitating, inviscid, non-radiating fluid is governed by equations embodying the conservation of mass, momentum and energy:

\[
\frac{d\rho}{dt} = -\rho \nabla \cdot \vec{v}, \quad (1a)
\]

\[
\frac{d\vec{v}}{dt} = -\nabla P + \vec{g}, \quad (1b)
\]

\[
\frac{du}{dt} = -\frac{P}{\rho} \nabla \cdot \vec{v}, \quad (1c)
\]

in which \(d/dt = \partial/\partial t + \vec{v} \cdot \nabla\) is the Lagrangian time derivative, \((\rho, P, \vec{v}, u)\) are the density, pressure, velocity and internal energy per unit mass, respectively, \\(\vec{g}\) denotes the force per unit mass due to the self-gravity of the fluid, and \(\vec{r}\) is the position vector of a fluid element. This system of equations has to be closed by specifying an equation of state \(P = P(\rho, u)\), which relates the pressure to the density and the internal energy.

In SPH, the fluid is described by a collection of \(N\) discrete, but extended and possibly overlapping, spherically symmetric particles which act as sampling points of the local properties of the fluid. Each particle has associated with it a mass \(m_i\), a velocity vector \(\vec{v}_i\) and also values for thermodynamic variables which describe the state of the fluid, such as the pressure \(P_i\), the density \(\rho_i\) and the specific internal energy \(u_i\). The sum of the mass of all particles must be equal to the total mass within the computational domain \(\Omega\). The evolution of the fluid is determined by following the motion of the particles, under the action of interparticle forces which simulate the effects of pressure, viscosity and self-gravity.

The density \(\rho(\vec{r})\) at a particular point in the fluid is obtained by summing the contribution from all the particles which overlap that point, using a weighting function \(W(\vec{r} - \vec{r}_j, h)\):

\[
\rho(\vec{r}) = \sum_{j=1}^{N} m_j W(\vec{r} - \vec{r}_j, h). \quad (2)
\]

\(W(\vec{r} - \vec{r}_j, h)\) is a smooth differentiable function, referred to as the smoothing kernel or the interpolating kernel. This kernel must satisfy the three following properties [11]: (i) it converges to the Dirac-\(\delta\)-function in the limit where the smoothing length \(h\) → 0, (ii) its integral over the domain \(\Omega\) equals unity, and (iii) \(W(\vec{r} - \vec{r}_j, h) = W(\vec{r} - \vec{r}_j, h)\) if \(h\) is the smoothing kernel has compact support, \(W(\vec{r} - \vec{r}_j, h) = 0\) when \(|\vec{r} - \vec{r}_j| > nh\), where \(n\) denotes the effective radius of the kernel in units of the smoothing length, the number of neighbours contributing to the sum in Eq. (2) is over a finite number of neighbouring particles. Eq. (2) replaces the continuity equation (1a). Any other variable \(f(\vec{r})\) can be evaluated at the general position \(\vec{r}\) using:

\[
f(\vec{r}) = \sum_{j=1}^{N} m_j \frac{f(\vec{r}_j)}{\rho_j} W(\vec{r} - \vec{r}_j, h), \quad (3)
\]

where \(f(\vec{r}_j)\) denotes the value of the variable \(f\) evaluated at the position of the particle with index \(j\). In addition, the gradient of the function \(f(\vec{r})\) can be obtained as follows:

\[
\nabla f(\vec{r}) = \sum_{j=1}^{N} m_j \frac{f(\vec{r}_j)}{\rho_j} \nabla W(\vec{r} - \vec{r}_j, h). \quad (4)
\]

Note that the SPH interpolation of the gradient of a function is obtained by differentiating the kernel with respect to \(\vec{r}\). Our implementation of SPH uses the standard M4-kernel or cubic spline kernel [25], which is defined as follows in three space dimensions:

\[
W(r_{ij}, h) = \frac{1}{\pi h^3} \left[ 1 - \frac{3}{2} u^2 + \frac{3}{4} u^4 \right], \quad 0 \leq u \leq 1.
\]

\[
W(r_{ij}, h) = \frac{1}{4\pi h^3} (2 - u)^3, \quad 1 \leq u \leq 2.
\]

\[
W(r_{ij}, h) = 0, \quad u \geq 2.
\]
where $u = r_{ij}/h$ and $r_{ij} = \|\mathbf{r}_i - \mathbf{r}_j\|$ is the distance between two particles. This smoothing kernel goes to zero at a distance of $2h$ of a given particle and can be readily differentiated in the $x$-, $y$-, and $z$-directions to build $\nabla W$. For the moment, we assume that the smoothing length $h$ in the above equations is constant in both time and space for all particles involved in the simulation. The procedure to approximate the equations of motion (1b) and the energy equation (1c) is dictated by the need to ensure the conservation of linear momentum, angular momentum and energy. We follow Gingold and Monaghan [2] and rewrite the pressure gradient in the equation of motion by making use of the equality

$$
\nabla \frac{P}{\rho} = \nabla \left( \frac{P}{\rho} \right) + \frac{P}{\rho^2} \nabla \rho.
$$

(6)

With the help of Eq. (4), and assuming vanishing body forces, the equation of motion becomes

$$
d\tilde{v}_i = -\sum_{j=1}^{N} \frac{m_j}{m_i} (\frac{P_i}{\rho_i} - \frac{P_j}{\rho_j}) \nabla W(r_{ij}, h),
$$

(7)

where $\tilde{v}$ denotes the gradient of $W(r_{ij}, h)$ with respect to the coordinates of particle $i$. Because $\nabla W(r_{ij}, h) = -\nabla W(r_{ij}, h)$ and the pressure part of the above expression is symmetric with respect to the particle indices, the total linear momentum is conserved exactly by construction: $\sum_{i=1}^{N} m_i \tilde{v}_i = 0$. Because the force acting between each pair of particles is symmetric and directed along the line joining the particles, the sum of all torques also vanishes and the total angular momentum is therefore conserved exactly as well.

A suitable expression for the energy equation can be derived in a straightforward way by starting from the first law of thermodynamics in the adiabatic regime:

$$
du_i = \frac{P_i}{\rho_i^2} dt,
$$

(11)

in which we substitute, with the help of Eq. (2).

$$
d\rho_i = \frac{d}{dt} \left( \sum_{j=1}^{N} m_j W(r_{ij}, h) \right) = \sum_{j=1}^{N} m_j v_{ij} \cdot \nabla W(r_{ij}, h),
$$

(12)

in which $v_{ij} = \tilde{v}_i - \tilde{v}_j$, so that one finally obtains

$$
du_i = \frac{P_i}{\rho_i^2} \sum_{j=1}^{N} m_j v_{ij} \cdot \nabla W(r_{ij}, h).
$$

(13)

Eqs. (2), (7) and (10), together with an equation of state, constitute a complete set of SPH equations, except for additional dissipative terms which are necessary to capture shock waves and contact discontinuities. Our treatment of these terms will be discussed in more detail in Section 4. The above equations have been derived for a constant smoothing length. The smoothing length determines the local resolution and the number of neighbours at a given point. The efficiency and accuracy of the code improves dramatically when $h$ depends on the local particle number density. Many standard SPH codes [3,4] therefore use a spatially variable smoothing length so that the number of neighbours of a particle is approximately constant, with an optimal value in the range $N_{opt} = 30–100$ for three-dimensional simulations. If each particle has its own smoothing length, momentum can only be conserved if the kernel depends on a symmetric combination of the smoothing lengths $h_i$ and $h_j$ of any pair of particles. One way to achieve this is to use the average of the two kernels, one with $h_j$ and one with $h_i$, [3]. The system of equations to be solved with variable $h$ then becomes

$$
\rho_i = \sum_{j=1}^{N} m_j W_{ij},
$$

(14)

$$
d\tilde{v}_i = -\sum_{j=1}^{N} \frac{m_j}{m_i} (\frac{P_i}{\rho_i} + \frac{P_j}{\rho_j}) \nabla W_{ij},
$$

(15)

$$
d\tilde{u}_i = \frac{P_i}{\rho_i^2} \sum_{j=1}^{N} m_j v_{ij} \cdot \nabla W_{ij},
$$

(16)

where $W_{ij}$ is a shorthand notation for the arithmetic average of $W(r_{ij}, h_i)$ and $W(r_{ij}, h_j)$. Although the above equations conserve momentum exactly, there is still an error in the conservation of energy because terms involving the temporal and spatial derivatives of the smoothing lengths have been neglected. The neglect of these terms can lead to relative errors in the order of 10% in the conservation of energy for some applications [14]. Monaghan [15], Monaghan and Price [16] and Springel and Hernquist [17] recently demonstrated, however, that the system of SPH equations including fully adaptive smoothing lengths can be derived from a variational principle based on nothing more than the same expression for the Lagrangian, the first law of thermodynamics and a prescription how to compute the density from summation. The main advantage of using a Lagrangian to derive the equation of motion is that the conservation of momentum and energy is “hard-wired” into the resulting system of equations. The derivation of the resulting set of equations is explained in the next section.

3. Adaptive smoothing length formalism derived from a variational principle

Following Monaghan and Price [16], we start with the Lagrangian for an ideal fluid [18]:

$$
L = \int \rho \left( \frac{v^2}{2} - u(\rho, s) \right) dV,
$$

(19)

where $s$ denotes the specific entropy. The discretised form of the above expression becomes

$$
L = \sum_{i=1}^{N} m_i \left( \frac{v_i^2}{2} - u(\rho_i, s_i) \right).
$$

(20)

from which the discretized SPH equations can be derived by means of the Euler-Lagrange equations

$$
\frac{d}{dt} \left( \frac{\partial L}{\partial \tilde{v}_i} \right) - \frac{\partial L}{\partial \tilde{u}_i} = 0.
$$

(21)

Substituting the Lagrangian into the above equations, we can write down an expression for the equation of motion which involves the derivatives of the thermal energy with respect to density, taken at constant entropy,

$$
m_i \frac{d\tilde{v}_i}{dt} = -\sum_j \frac{\partial u_{ij}}{\partial \rho_j} \frac{\partial \rho_j}{\partial \tilde{u}_i},
$$

(22)

which can be evaluated using the first law of thermodynamics, $du = (P/\rho^2) d\rho$, so that one obtains

$$
m_i \frac{d\tilde{v}_i}{dt} = -\sum_j \frac{P_j}{\rho_j^2} \frac{\partial \rho_j}{\partial \tilde{u}_i},
$$

(23)

The method we use for determining the smoothing length of a particle is similar to the one adopted by Price and Monaghan [19]. A similar procedure has been used by Springel and Hernquist [17]. The idea is to let the smoothing length evolve as a function of the density through a relation of the form

$$
h_i = n \left( \frac{m_i}{\rho_i} \right)^{1/3},
$$

(24)
in which \( \eta \) is a dimensionless parameter which specifies the size of the smoothing length in terms of the average particle spacing. An alternative interpretation of Eq. (19) is that the mass contained within the smoothing sphere of each particle should be held constant:

\[
\frac{4\pi}{3} (\sigma h_i)^3 \rho_i = m_i N_{\text{opt}} = \text{constant}, \tag{20}
\]

where \( \sigma \) denotes the compact support radius of the smoothing kernel, which equals 2 for the cubic kernel used in this paper, and \( N_{\text{opt}} \) is an approximate measure of the number of neighbours within the smoothing sphere of each particle, which is given by

\[
N_{\text{opt}} = \frac{4\pi}{3} (\sigma \eta)^3. \tag{21}
\]

We determine \( \eta \) by fixing the number of neighbours in the above expression to a predefined value, which we take to be typically 50 in 3D simulations. The density is then computed through direct summation by means of Eq. (2), which, through the relation (19), becomes a nonlinear relation for both the density and the smoothing length. To obtain a self-consistent solution, we iterate (19), becomes a nonlinear relation for both the density and the derivative of the smoothing lengths with respect to the density and the derivative of the smoothing lengths can then be derived as follows. Differentiation of the density sum, Eq. (2), yields the following expressions for the total time derivative and the gradient of the density:

\[
\frac{d\rho_i}{dt} = \frac{1}{\Omega_i} \sum_{j=1}^{N} m_j \bar{v}_{ij} \cdot \nabla_i \bar{W}(r_{ij}, h_i), \tag{23}
\]

\[
\frac{\partial \rho_i}{\partial r_j} = \frac{1}{\Omega_i} \sum_{k=1}^{N} m_k \bar{v}_j \bar{W}(r_{ik}, h_i), \tag{24}
\]

where the quantities \( \Omega_i \) are defined as

\[
\Omega_i = 1 - \frac{\partial h_i}{\partial \rho_i} \sum_{j=1}^{N} m_j \frac{\partial W(r_{ij}, h_i)}{\partial h_j}. \tag{25}
\]

The quantities \( \Omega_i \) depend on the partial derivatives of the smoothing lengths with respect to the density and the derivative of the kernel function with respect to the smoothing length. It should be emphasised that in Eqs. (23) and substituting the above relations in Eqs. (8) and (17), the final expressions for the energy equation and the equation of motion can be written as

\[
\frac{d\bar{v}_i}{dt} = \frac{p_i}{\rho_i^2 \Omega_i} \sum_{j=1}^{N} m_j \bar{v}_{ij} \cdot \nabla_i \bar{W}(r_{ij}, h_i), \tag{26}
\]

\[
\frac{d\bar{v}_i}{dt} = \sum_{j=1}^{N} m_j \left( \frac{p_i}{\rho_i^2 \Omega_i} \nabla_i W(r_{ij}, h_i) + \frac{p_j}{\rho_j^2 \Omega_j} \nabla_j W(r_{ij}, h_j) \right). \tag{27}
\]

In our implementation, the \( \Omega_i \) are computed in the same subroutine which also determines the densities and the smoothing lengths. Generally we find that storing complete neighbour lists for all particles in a simulation is not efficient in terms of memory requirements. We therefore compute the neighbour list for the particle currently being processed on the fly whenever it is necessary, both in the subroutine which updates the densities and smoothing lengths of the particles, as well as in the subroutines which update the position, velocity and energy of the particles. It is important to note also that, while the summation in Eq. (2) includes all particles within the smoothing sphere with radius \( 2h_i \) surrounding the particle \( i \), the summation which appears in the equation of motion (Eq. (27)) must include all interacting neighbours for which \( r_{ij} < 2 \max(h_i, h_j) \) holds for a particle \( i \) and its neighbours \( j \), so that the reciprocity principle is always satisfied, even if \( r_{ij} > 2h_i \).

This is implemented in GRADSPH with an additional call to the neighbour search routine when calculating the hydrodynamical acceleration of the particles.

4. Dissipative terms and treatment of shock waves

The above derivation of the “grad-h” SPH equations from a variational principle implies differentiability of all physical variables. However, real hydrodynamic flows can develop flow discontinuities, especially shock waves and contact discontinuities, where the derivative of one or more hydrodynamic variable can become infinite. By the assumption of differentiability, flow discontinuities are inherently excluded in the above formulation, and thus need to be recovered in the numerical solution by means of an adequate procedure. The standard procedure in SPH is to add dissipative terms to the equations which diffuse discontinuities on the smoothing scale so that they again become resolved by the numerical method. Based on an analogy with grid-based methods employing Riemann solvers and using the notion of a signal velocity in between particle pairs, Monaghan [22] derived a general form of these dissipative terms. Although previous implementations of SPH (for example, Monaghan [11], Balsara [26]) only added a viscosity term in the equation of motion in order to capture shock waves, the analogy with Riemann solvers suggests that, as noted by Monaghan [22], the evolution equation for all conservative variables should contain a dissipative term to handle jumps in that particular conservative variable. In particular, for the conserved momentum and total energy per unit mass, \( e = u + \frac{1}{2}v^2 \), the additional dissipative terms take the form

\[
\left( \frac{dv_i}{dt} \right)_{\text{diss}} = -\sum_{j=1}^{N} m_j \frac{\alpha v_{\text{sig}}(\bar{v}_i - \bar{v}_j) \cdot \bar{e}_{ij}}{\rho_{ij}} \bar{W}_{ij}, \tag{28}
\]

\[
\left( \frac{d e_i}{dt} \right)_{\text{diss}} = -\sum_{j=1}^{N} m_j \frac{\alpha v_{\text{sig}}(e_i^\text{r} - e_j^\text{r})}{\rho_{ij}} \bar{e}_{ij} \cdot \bar{W}_{ij}, \tag{29}
\]

in which \( \bar{e}_{ij} \) is the unit vector in the direction joining any two particles, \( \alpha \) is the dissipation parameter, \( \rho_{ij} \) is the average density of any two particles, and \( e_i^\text{r} = \frac{1}{2}(|\bar{v}_i| - |\bar{e}_{ij}|)^2 + u_i \) is an energy which includes only components along the line joining the particles (for more details, we refer to Monaghan (1997) [22]). In addition, \( v_{\text{sig}} \) is an estimate of the signal velocity, which in its simplest form is given by

\[
v_{\text{sig}} = c_i + c_j - 3\bar{e}_{ij} \cdot \bar{v}_{ij}, \tag{30}
\]

in which \( c_i \) and \( c_j \) denote the sound velocities associated with particles \( i \) and \( j \), respectively. If we consider the evolution of the thermal energy,

\[
\frac{du_i}{dt} = \frac{d e_i}{dt} - v_i \cdot \frac{d \bar{v}_i}{dt}, \tag{31}
\]

we obtain, with the help of Eqs. (28) and (29),

\[
\left( \frac{du_i}{dt} \right)_{\text{diss}} = -\sum_{j=1}^{N} m_j \frac{\alpha v_{\text{sig}}(\bar{v}_i - \bar{v}_j)^2}{\rho_{ij}} \bar{e}_{ij} \cdot \bar{W}_{ij} + \frac{\alpha v_{\text{sig}}}{2} (u_i - u_j) \bar{e}_{ij} \cdot \bar{W}_{ij}. \tag{32}
\]
The first term in Eq. (32) relates jumps in kinetic energy and corresponds to the thermal energy contribution of the viscosity term appearing in Eq. (28). The resulting artificial viscosity terms resemble the standard artificial viscosity prescription used in previous SPH implementations (for example, Monaghan [23] and Balsara [26]), except for a factor \( h_i/|r_{ij}| \). Note that these terms are only active when particles approach each other, i.e. \( \dot{v}_{ij} \cdot \mathbf{e}_{ij} < 0 \) and are otherwise set to zero. On the other hand, the second term provides an artificial thermal conductivity, which acts to smooth discontinuities in the thermal energy. The effect of this term becomes significant at contact discontinuities. A similar artificial conductivity term has been used before by Price and Monaghan [27]. Note that a separate dissipation parameter \( \alpha \) and a separate signal velocity \( v_{sig}^\alpha \) is assigned to this term, with \( v_{sig}^\alpha \) given by (D. Price, private communication and [21]):

\[
v_{sig}^\alpha = \sqrt{|P_i - P_j| \over \rho_{ij}}.
\]  

(33)

The effect of the artificial conductivity term will be discussed in more detail when dealing with the one-dimensional shock tube tests described below. We generally only include an artificial thermal conductivity term when dealing with problems involving strong contact discontinuities. Following Morris and Monaghan [24], the parameter \( \alpha \) which determines the strength of the artificial viscosity terms is made time-dependent and given a separate value for each particle, so that the effect of artificial viscosity disappears in regions of smooth flow. This is put into effect by solving an additional differential equation for each particle of the form

\[
\frac{d\alpha_i}{dt} = \frac{\alpha_i - \alpha_{\text{min}}}{\tau_i} + S_i,
\]

(34)

which contains a source term \( S_i \), which determines the growth of the dissipation parameter when approaching a shock wave, and a decay timescale \( \tau_i \), which depends on the speed by which the particle crosses the discontinuity. \( \alpha_{\text{min}} \) is the minimum timescale to which we allow \( \alpha_i \) to decay. The timescale \( \tau_i \) is computed according to

\[
\tau_i = \frac{h_i}{C_{\alpha} v_{\text{sig}}^{\alpha_{\text{max}}}},
\]

(35)

in which \( v_{\text{sig}}^{\alpha_{\text{max}}} \) is the maximum signal velocity for all particle pairs in the neighbour list of particle \( i \), and \( C_{\alpha} \) is a constant which we typically set to \( C_{\alpha} = 0.2 \), so that \( \alpha_i \) decays over 5 smoothing lengths. We use the same source term as in Morris and Monaghan [24],

\[
S_i = \max\{-\nabla \cdot \mathbf{v}_{i}(\alpha_{\text{max}} - \alpha_i), 0\},
\]

(36)

in which \( \alpha_{\text{max}} \) is a maximum value for \( \alpha_i \), and the divergence of the velocity of a particle is evaluated according to

\[
\nabla \cdot \mathbf{v}_i = -\frac{1}{\rho_i} \sum_{j=1}^{N} m_j \dot{v}_{ij} \cdot \nabla W(r_{ij}, h_i).
\]

In the dissipation terms in Eqs. (28) and (32), we also preserve symmetry by introducing the average \( \alpha \) over particle pairs, \( \alpha = (\alpha_i + \alpha_j)/2 \).

5. Implementation of the TCG method and neighbour search

We now turn our attention to the implementation of the tree code. For a collection of \( N \) particles, the gravitational acceleration \( \ddot{\mathbf{g}}_i \) experienced by particle \( i \) can be written as a direct summation of two-body interactions:

\[
\ddot{\mathbf{g}}_i = -G \sum_{j=1, j \neq i}^{N} \frac{m_j (\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}^3}.
\]

(38)

With SPH, we try to describe all quantities as continuous functions in space and time, and we have to avoid very large forces \( \sim 1/r^2 \) when particles approach each other closely. The gravitational forces exerted by the particles are smoothed in the same way as we did for the treatment of hydrodynamical forces, using the spherically symmetric kernel defined in Eq. (5), with a gravitational smoothing length \( e = h \) so that we obtain the same spatial resolution for both the hydrodynamic and the gravitational forces. If two particles overlap (i.e. \( r_{ij} < 2(h_i + h_j) \)), the gravitational force between them is computed using Gauss’ theorem. Otherwise they are treated as point masses. The mass distribution of a single particle can be written as

\[
\rho(u, h_i) = m_i W(u, h_i).
\]

(39)

Applying Gauss’ theorem, the gravitational acceleration exerted by particle \( j \) on a neighbouring particle \( i \) can be written as

\[
\ddot{\mathbf{g}}_i = -\frac{G m_j}{h_i} \left[ \Phi'(r_{ij}, h_i) \mathbf{e}_{ij} \right].
\]

(40)

The functional form of the core force \( \Phi'(r, h) \) can be found in Ref. [3].

The use of variable smoothing lengths implies that the force acting between two particles needs to be symmetrised in order to conserve angular and linear momentum, so that the force on particle \( i \) becomes

\[
\ddot{\mathbf{g}}_i = -\frac{G m_j}{h_i} \left[ \Phi'(r_{ij}, h_i) + \Phi'(r_{ji}, h_j) \right] \mathbf{e}_{ij}.
\]

(41)

When adopting an adaptive smoothing length formalism as used in this paper, care must be taken to ensure the conservation of energy, since the gravitational potential energy of the particles is constantly changing in both space and time. The derivation of the correction terms for the particle acceleration when using adaptive smoothing lengths is discussed in detail in Price and Monaghan [33]. Applying the Euler–Lagrange equations for a self-gravitating fluid yields the following expression for the gravitational acceleration of particle \( i \):

\[
\ddot{\mathbf{g}}_i = -G \sum_{j=1}^{N} \frac{m_j}{h_i} \left[ \Phi'(r_{ij}, h_i) + \Phi'(r_{ji}, h_j) \right] \mathbf{e}_{ij} - \frac{G}{2} \sum_{j=1}^{N} \left[ \frac{\partial_j \Omega_i}{\Omega_i} \nabla W(r_{ij}, h_i) + \frac{\partial_j \Omega_i}{\Omega_j} \nabla W(r_{ij}, h_j) \right].
\]

(42)

in which the quantities \( \Omega_i \) have already been defined in Eq. (25). In addition, the quantities \( \zeta_i \) are given as follows:

\[
\zeta_i = \frac{\partial h_i}{\partial \rho_i} \sum_{j=1}^{N} m_j \frac{\partial \Phi(r_{ij}, h_i)}{\partial h_i}.
\]

(43)

in which \( \Phi(r, h) \) denotes the softened gravitational potential of a particle. The derivatives of \( \Phi \) with respect to the smoothing length can be tabulated directly for use in the program. Expressions for \( \Phi \) and \( \partial \Phi/\partial h \) for the cubic spline kernel can be found in Appendix A of Price and Monaghan (2007) [33].

When dealing with a large number of particles, the computational time of the direct summation implied by Eq. (38) becomes prohibitive, as it scales with particle number as \( O(N^2) \). Efficient strategies now exist for reducing the dependence of the computational time on the number of particles. We implemented Tree Code Gravity (TCG) [28,30], which scales as \( O(N \log(N)) \). This method uses an octal tree containing spatial information on the distribution of the particles and the position of the center of mass of groups of particles. The combined gravitational force of distant groups of particles is approximated by means of a multipole expansion. In this way, the gravitational force arising from single
particles can be substituted with the gravitational force of groups of particles, making this algorithm much more efficient than direct summation.

The tree is constructed from the top down. The root cell, which is the lowest node of the tree, is a cube which contains all particles in the system. The root cell is then recursively subdivided into eight equal subvolumes, until a subvolume at a given level in the tree either contains only one particle or no particles at all. Cells which contain no particles are discarded. The center of mass of each cell is computed from the center of mass of its subcells. The construction of the tree is repeated at each integration step, and for each cell with more than one particle we store the position of the center of mass of the cell. In addition, we also compute the total mass and the quadrupole moments of each cell.

The gravitational acceleration on a particle with position vector \( \vec{r}_j \) with respect to the center of mass of cell \( j \) with total mass \( m_j \) is given by \( g_j = -\nabla \Phi_j \), where the potential \( \Phi \) can be written as a multipole expansion up to quadrupole order [32]:

\[
\Phi = -\frac{G m_j}{|\vec{r}_j|} - \sum_{l=2}^{2} a_l \frac{\|\vec{r}_j\|^2}{2|\vec{r}_j|^2} m_j \tag{44}
\]

and summation over repeated indices is assumed. Note that there is no dipole term because the origin of the coordinate system is located at the center of mass of the cell. If the cell contains \( N \) particles, explicit expressions for the components of the quadrupole tensor are given by [32]

\[
Q_{kl} = \sum_{p=1}^{N} m_p \left( 3r_{p,k}r_{p,l} - |\vec{r}_p|^2 \delta_{kl} \right) \tag{45}
\]

where \( r_{p,k} \) denotes the position vector of particle \( p \) with respect to the center of mass of the cell. In addition, if a cell in the tree contains subcells, the quadrupole moment of the cell is obtained from the quadrupole moment of the subcells:

\[
Q_{kl} = \sum_{p=1}^{N_{subcells}} Q_{kl}^p + \sum_{p=1}^{N_{subcells}} m_p \left( 3r_{p,k}r_{p,l} - |\vec{r}_p|^2 \delta_{kl} \right) \tag{46}
\]

where \( r_{p,k} \) now denotes the coordinates of subcell \( p \) with respect to the center of mass of the parent cell. This equation is applied recursively during the construction of the tree. The gravitational force on a particle \( i \) is evaluated by walking down the tree, starting at the root node. If the cell \( j \) contains only one particle, we compute its gravitational force on \( i \) using the direct summation formula (Eq. (41)). If \( j \) contains more than one particle, we calculate the gravitational acceleration at particle \( i \) using the monopole–quadrupole approximation of the cell unless the cell is so close that we must investigate the gravitational forces exerted by its subcells instead. We use a geometrical criterion to decide whether a cell should be subdivided. In the original version of the Barnes–Hut algorithm, the criterion used is given by

\[
\frac{l_j}{\theta} \leq |\vec{r}_i|, \tag{47}
\]

where \( l_j \) is the linear size of cell \( j \) and \( \theta \) is a user-defined parameter, which is used to control the accuracy of the gravitational force calculation. Since the above criterion can give rise to large errors when the center of mass of a cell is far removed from its geometric center, we use the modified version suggested by Salmon and Warren [31]

\[
\frac{l_j}{\theta} + \delta_j \leq |\vec{r}_i|, \tag{48}
\]

where \( \delta_j \) denotes the distance between the geometric center and the center of mass of a cell. The critical radius \( l_j/\theta + \delta_j \) only depends on the properties of node \( j \), and can be conveniently calculated at the time of tree construction and stored in the tree data structure for use during the force calculation. The process of subdividing cells is recursively repeated until all cells in the tree are either added as a monopole–quadrupole approximate contribution or found to contain a single particle. The optimal choice for the value of \( \theta \) is a trade-off between increasing computational speed for lower values of \( \theta \), while at the same time improving the accuracy of the force calculation. We use \( \theta = 0.5 \) as a good compromise between these two requirements [30].

Our implementation of TCG is based on FORTRAN routines made available by Prof. Joshua Barnes [29]. We wrote additional routines to use the tree representation of the particle distribution for the double duty of providing both the gravitational forces and the near-neighbour list of each particle.

6. Time integration

We implemented a second-order PEC (predict-evaluate-correct) scheme which allows for individual time steps for each particle following the method proposed by Serna et al. [34]. We slightly modified the original algorithm so that at each time step, the particles are advanced if their hydrodynamical and gravitational accelerations need to be updated, irrespective of their location in the computational domain. The algorithm enters time step \( n \) at the global time \( t^n \) with known positions \( \vec{r}_i^n \), velocities \( \vec{v}_i^n \) and accelerations \( \vec{a}_i^n \) of the particles, along with their smoothing length \( h_i^n \), viscosity parameter \( \alpha_i^n \), specific internal energy \( u_i^n \) and its time derivative \( \dot{u}_i^n \). In addition, the integration scheme uses two vectors \( \vec{e}_i^{last} \) and \( \vec{e}_i^{next} \), where \( \vec{e}_i^{next} \) is the time at which the last update of the acceleration of particle \( i \) was performed and \( \vec{e}_i^{last} = \vec{e}_i^{next} + \Delta t_i \) is the time at which a new update of the particle will be necessary in the future. The time steps \( \Delta t_i \) are limited by the requirement of numerical stability of the PEC scheme. We impose two restrictions on the magnitude of the time step of a particle. The first restriction limits the timescale for significant changes in velocity due to the accelerations:

\[
\Delta t_{i,1} = \left( \frac{h_i}{|\vec{a}_i|} \right)^{1/2}, \tag{49}
\]

while the signal-velocity approach leads to a Courant-like timestep condition:

\[
\Delta t_{i,2} = \max_i \left( \frac{c_s}{|\vec{v}_i|^2 + \vec{e}_i^2} \right), \tag{50}
\]

where the maximum is taken over the neighbours of each particle. The time step for particle \( i \) is then obtained as follows:

\[
\Delta t_i = \min \left( \Delta t_{i,1}, \Delta t_{i,2} \right), \tag{51}
\]

in which \( C_s \) denotes the Courant number. We conservatively take \( C_s = 0.3 \). The algorithm then goes through three steps. Firstly, for all particles we predict the values of \( \vec{r}_i^{n+1}, \vec{v}_i^{n+1}, u_i^{n+1} \) and \( \alpha_i^{n+1} \) at time step \( t^{n+1} = t^n + \Delta t_i \):

\[
\vec{r}_i^{n+1} = \vec{r}_i^n + \vec{v}_i^n \Delta t_i + \frac{\vec{a}_i^n \Delta t_i^2}{2} \tag{52a}
\]

\[
\vec{v}_i^{n+1} = \vec{v}_i^n + \vec{a}_i^n \Delta t_i \tag{52b}
\]

\[
\alpha_i^{n+1} = \alpha_i^n + \vec{a}_i^n \Delta t_i \tag{52c}
\]

\[
u_i^{n+1} = u_i^n + \dot{u}_i^n \Delta t_i \tag{52d}
\]

where \( \Delta t \) is the system time step defined as

\[
\Delta t = \min_i (t_i^{next} - t^n). \tag{53}
\]

Note that the individual time steps \( \Delta t_i \) are generally different from the system time step \( \Delta t \). In the second step, we construct the list
of "active" particles that will be updated at the current time step. This is done by looping through the list of particles and checking whether particle $i$ satisfies the inequality

$$t_i^{\text{last}} + 0.5 \Delta t_i - t_i^n - \Delta t < 0.$$  
(54)

For the particles which pass this test, we update their hydrodynamic variables ($\rho_i^{n+1}, p_i^{n+1}, h_i^{n+1}, \vec{u}_i^{n+1}, \alpha_i^{n+1}$) and their accelerations $\delta_i^{n+1}$. In the third step we correct $\vec{p}_i^{n+1}$, $\vec{v}_i^{n+1}$, the internal energy $u_i^{n+1}$ and the viscosity parameter $\alpha_i^{n+1}$ using the following expressions:

$$\vec{p}_i^{n+1} = \frac{\rho_i^{n+1} - \rho_i^{n}}{\rho_i^{n}} \delta_i, \quad \vec{v}_i^{n+1} = \vec{u}_i + \frac{\alpha_i^{n+1} - \alpha_i^{n}}{\alpha_i^{n+1}} \delta_i, \quad u_i^{n+1} = u_i^{n+1} + C (\alpha_i^{n+1} - \alpha_i^{n}) \delta_i, \quad \alpha_i^{n+1} = \alpha_i^{n+1} + C (\alpha_i^{n+1} - \alpha_i^{n}) \delta_i,$$
(55a) (55b) (55c) (55d)

where $\delta t_i = t_i + \Delta t - t_i^{\text{last}}$ is the time interval elapsed from the last evaluation of $\delta_i$ to that performed in the current timestep. The scheme described above maintains second order accuracy with the following choice for the constants $A$, $B$ and $C$: $A = 1/3$, $B = 1/2$ and $C = 1/2$. Finally, we update the global time $t_i^{n+1} = t_i^n + \Delta t_i$, as well as the values of $t_i^{\text{last}}$ and $t_i^{\text{next}}$ for each active particle.

7. Parallelisation strategy

For computational cost saving when large numbers of particles are used, the implementation of a three-dimensional SPH code generally requires parallelisation, in order to enable calculations to be performed on a supercomputer with distributed memory architecture. We parallelised our code using the standard MPI (Message Passing Interface) library for interprocessor communication, used in combination with FORTRAN 90. The current version of the program uses a simple parallelisation strategy which has been discussed recently in [35]. At the start of a run, all processors store a copy of the arrays containing the list of properties of all particles participating in the simulation:

$$(\rho_i, \vec{u}_i, h_i, \vec{v}_i, \alpha_i, t_i^{\text{next}}, t_i^{\text{last}}, \Delta t_i, \Omega_i, \zeta_i),$$
(56)

where the index $i$ runs from 1 to the total number of particles $N$, along with the data structure of the TCG part of the code.

We then divide up the task among NPROCS processors in a rather straightforward way. First, we break up the particles into NPROCS subsets, each with approximately the same number of particles. During a time step, each processor first predicts the positions, velocities and internal energies of its assigned particles according to Eqs. (52). As a result, each processor has updated properties for its section of the arrays and out of date information for the other sections. The processors then exchange their data and receive the full updated arrays by means of ALLGATHER commands. Next, the tree is reconstructed on each processor and each processor determines the list of active particles among its subset of the particles. Rebuilding the Barnes–Hut tree usually takes up only around 1% of CPU time. The processors then update the smoothing lengths, densities and neighbour lists for their active particles using the iterative procedure described in Section 3 and compute updated values of the quantities $\Omega_i$ and $\zeta_i$ for each particle as well. The updated arrays $(\rho_i, h_i, \Omega_i, \zeta_i)$ are then exchanged among the processors with ALLGATHER commands. In the next step, each processor computes the hydrodynamical and gravitational accelerations for the active particles within its section and corrects the positions, velocities and internal energies of the particles by means of Eqs. (55). The full corrected arrays are then exchanged among the processors by means of a third data transfer phase similar to the prediction step. Note that this procedure avoids the exchange of neighbour lists between the processors. Finally, the processors update the arrays $t_i^{\text{next}}, t_i^{\text{last}}$ and $\Delta t_i$ and compute the system time step by means of an ALLREDUCE command. The process then starts all over again in the next timestep. The performance of the code and its memory limitations using this approach are discussed further in Section 10.

8. Numerical test problems

8.1. 1D shock tube tests

We first demonstrate the performance of the adaptive smoothing length formalism by simulating three one-dimensional shock tube problems. They correspond to tests 1, 3 and 4 in Table 4.1, p. 129 of Toro (2001) [37].

8.1.1. Sod’s shock tube

The standard shock tube test for any compressible fluid dynamics code is that of Sod (1978) [36]. This problem consists of dividing the computational domain in two halves, one containing a high pressure, high density gas, while the other halve consists of a low pressure, low density gas. The two halves are allowed to interact at $t = 0$, and the interaction between the two gases results in both a shock wave and a rarefaction wave moving through the gas. We solve this problem in the domain $[-1, 1]$ with an ideal gas equation of state of the form

$$P = (\gamma - 1) \rho u,$$
(57)

with an adiabatic exponent $\gamma = 7/5$. To the left of the origin, the initial state is given by $[P, \rho, v_x] = [1.0, 1.0, 0.0]$, while to the right of the origin the initial state is $[P, \rho, v_x] = [0.1795, 0.25, 0.0]$. The initial conditions are constructed by placing 2400 particles to the left of the origin and 600 particles to the right of the origin with spacing given by $dx = 1/2400$ and $dx = 1/600$, respectively. All particles have equal mass $m = 1/2400$, while the velocities and thermal energies of the particles are initialised according to the left and right state given above. Following Price and Monaghan [19], we do not use any smoothing of the initial conditions near the discontinuity, but instead apply an artificial conductivity term as defined in Eq. (32), with a constant $\alpha^N = 1.0$ and the signal velocity given by (30). The time-dependent artificial viscosity terms are integrated with $\alpha_{\text{min}} = 0.1$, $\alpha_{\text{max}} = 2.0$ and $C = 0.2$. We use the variable smoothing length formalism with a consistent update of the smoothing length with the density and $N_{\text{opt}} = 0$. At both ends of the computational domain, we simulate reflecting boundary conditions by adding ghost particles, whose properties are identical to those of the particles closest to the boundary, except for a sign reversal of the velocities.

In Fig. 1, the dashed lines give the resulting profiles of the density, velocity, pressure and thermal energy at $t = 0.5$, along with the exact solution computed with the Riemann solver described in Toro [37]. The exact solution is indicated by the solid line in each plot. The initial condition described here has no initial flow, and the reflecting boundaries do not affect the results because the waves generated at the contact discontinuity have not yet reached the boundary at this time. The agreement between the exact solution and the SPH result is very good for all physical quantities, with excellent resolution of both the shock and the rarefaction waves. The introduction of the artificial thermal conductivity term also efficiently removes the effect of wall heating at the contact discontinuity, in agreement with Price [20,21].

8.1.2. Strong blast wave

We now consider a more extreme version of the shock tube considered previously. In this problem, the initial conditions to the
left and to the right of the origin in the domain \([-1, 1]\) are given by \( [P, \rho, v_x] = [1000, 0, 1.0, 0.0] \) and \( [P, \rho, v_x] = [0.01, 1.0, 0.0] \), respectively, with reflecting boundary conditions at both ends. The initial pressure ratio of 100000 between the left and the right states creates a very strong blast wave which propagates into the fluid to the right of the origin. The velocity of the contact discontinuity is very close to that of the shock, producing a strong density enhancement behind the shock front. This test is therefore challenging for all numerical hydro codes. We construct the initial condition using 3000 equal mass particles with mass \( m = \frac{1}{3000} \) and uniform spacing \( dx = \frac{1}{3000} \) with no smoothing of the initial conditions. The viscosity and thermal conductivity terms are treated in the same way as before. The results of this test are shown in Fig. 2, indicating an excellent agreement with the solution provided by the exact Riemann solver. Both the shock and rarefaction waves are clearly resolved, and the artificial thermal conductivity leaves only a small error in the pressure at the contact discontinuity.

8.1.3. Riemann problem resulting in two shock waves

We finally consider the Riemann problem with left and right states in the interval \([-1, 1]\) given by \( [P, \rho, v_x] = [460.894, 5.99924, 19.5975] \) and \( [P, \rho, v_x] = [46.0950, 5.99924, -6.19633] \), respectively. This initial condition leads to two shock waves propagating in the left and right fluid separated by a contact discontinuity. The initial conditions for this test are constructed with 3000 uniformly spaced equal mass particles with \( dx = 1/3000 \) and \( m = \rho \, dx \) and open boundary conditions at both ends of the computational domain, implemented by adding a sufficient number of ghost particles. The analytical and SPH solutions for this test are illustrated in Fig. 3, indicating a very good agreement between the two.

8.2. Three-dimensional tests

8.2.1. Gravitational collapse of an adiabatic gas sphere

A considerably more demanding problem is the so-called Evrard collapse test \([38]\) which serves as one of the standard tests for SPH codes including self-gravity. This test follows the adiabatic collapse of an initially cold and static gas sphere with mass \( M \) and radius \( R \). The equation of state is for an ideal gas with \( \gamma = 5/3 \). The initial density profile as a function of the radial distance \( r \) from the cloud center is given by:

\[
\rho = \frac{M}{2\pi R^2} \frac{1}{r^2}.
\]

The total gravitational potential energy of the sphere can be derived from the expression:

\[
E_P = -\frac{2}{3} \frac{GM^2}{R}.
\]
in which $G$ denotes the gravitational constant. In addition, all SPH particles initially have the same thermal energy per unit mass, which we put equal to

$$u = 0.05 \frac{GM}{R}.$$  \hfill (60)

The initial velocity of the particles inside the sphere was set to zero. In the code we use the natural system of units, in which we set:

$$G = M = R = 1.$$ \hfill (61)

The initial particle distribution is constructed by putting particles on a hexagonal cubic lattice within a sphere of radius 1 and stretching the particle distribution so that a $1/r$ density profile is obtained. The initial gravitational potential energy is $E_G = -2/3$.

Fig. 4 shows the time evolution of the energies during a run of the code with 9860 particles. The values for the viscosity parameters are set to $\alpha_{\text{min}} = 0.1$, $\alpha_{\text{max}} = 0.5$ and $C = 0.2$, respectively. Time is given in units of the initial freefall time $t_f$ of the cloud. Initially far from equilibrium, the cloud collapses under its own self-gravity and converts most of its kinetic energy into thermal energy during the time of maximum compression between $t \sim 0.8 \ t_f$ and $t \sim 1.2 \ t_f$. After some time a central bounce occurs and a shock wave is formed which propagates outwards. The cloud expands slowly and finally reaches a state close to virial equilibrium for $t > 2 \ t_f$ with total internal energy $E_U = -E_G/2$. It is interesting to compare the results of this collapse test including the “grad-h” terms with the results obtained when the effect of the variable smoothing lengths is ignored. In the latter case, we use the conventional set of equations defined by (11), (12) and (13) with the same viscosity terms. In addition, the simulations with and without the “grad-h” terms compute the smoothing lengths of the particles using Eq. (19) and $N_{\text{opt}} = 50$. The results of both simulations are indicated as the red and blue lines in Fig. 4. Obviously, there are only minor differences between the two formalisms with regard to the overall time evolution of the collapse. In Fig. 5, we compare the time evolution of the total energy of the system, normalised to its value at the beginning of the simulation. Apart from the corrections for the variable smoothing lengths of the particles, the dominant factors affecting the conservation of energy are the accuracy with which the tree code computes the gravitational forces and the accuracy of the variable time-stepping formalism. To reduce the effect of both sources of error, we reduced the opening angle of the tree code to $\theta = 0.1$ and the Courant number to $C_\alpha = 0.1$. Fig. 5 indicates that both versions of SPH conserve total energy to within a relative error below 2% for the entire duration of the simulation. The Lagrangian SPH formalism shows a slightly better performance on this test and conserves energy to within 1%. The total linear and angular momentum of the cloud, which are initially zero for the static initial conditions, are also very well conserved.
Fig. 3. Results of a Riemann problem leading to two shock waves separated by a contact discontinuity at $t = 0.04$. Again, the panels show density, pressure, velocity and thermal energy with the solid and dashed lines indicating the exact solution and the SPH results as before.

Fig. 4. Comparison of the time evolution of the different energies during the adiabatic collapse of an initially isothermal sphere. From top to bottom, the curves denote the thermal energy, kinetic energy, total energy and gravitational potential energy, respectively. The red and blue lines compare the results obtained using the Lagrangian formulation of SPH (red lines) and the conventional SPH formalism (blue lines) (see text for details). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 5. Comparison of the time evolution of the total energy of system, normalised to its initial value, for the Lagrangian SPH formalism (red line) and the conventional SPH formalism (blue line) (see text for details). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The structure of the collapsing gas sphere obtained with the SPH code is illustrated in Fig. 6. Here we show radial profiles of the density, pressure and radial velocity, azimuthally averaged in the midplane of the cloud at $t = 0.8$, just before the bounce occurs. In order to resolve the structure of the shock wave after the bounce, a substantial number of particles is necessary. We therefore show the results for simulations using 75,000 particles (red dashed lines) and 150,000 particles (blue dotted lines). An analytic solution is
not available for this problem. In addition, the solid lines show the results of a high resolution one-dimensional simulation of the cloud collapse obtained by Steinmetz and Müller [39] with a PPM scheme using 350 zones in the radial direction.

We see that with the number of particles used in this test, the 3D SPH simulations reproduce the overall features of the 1D PPM solution reasonably well, although the artificial viscosity terms induce some broadening of the shock front, especially when considering the radial velocity distribution. The small departure from the PPM solution which is visible in the density and pressure profiles is the a relaxation effect of the initial particle distribution. Apart from these deficiencies, the radial location of the outward moving shock is reproduced correctly. For both particle numbers used, the radial profiles of density, pressure and velocity outside of the shock region have nearly converged to the PPM solution.

### 8.2.2. Static structure of a polytrope

Another test of self-gravitating hydrodynamics is to verify the static structure of a polytrope by allowing an initial arrangement of gas to settle towards hydrostatic equilibrium. We perform the test here by putting 5000 equal mass particles onto a hexagonal cubic lattice within a uniform density sphere of radius \( r = 1 \) and mass \( M = 1 \). As before for the Evrard collapse test, we use a natural system of units. This particle distribution is then allowed to collapse under its own self-gravity with a polytropic equation of state \( P = K \rho^n \) with \( n = 5/3 \). In order to damp down the oscillations of the system around its equilibrium state, we introduce an additional dissipative term in the equation of motion:

\[
\frac{d\vec{v}}{dt} = -\kappa \vec{v} + \vec{f},
\]

in which we put \( \kappa = 0.05 \) and \( \vec{f} \) denotes the total force on a particle, including gravity and hydrodynamics. The artificial viscosity parameters are set to \( \alpha_{\min} = 0.1 \), \( \alpha_{\max} = 2.0 \) and \( \zeta = 0.2 \), respectively. The opening angle used by the tree code is set to \( \theta = 0.1 \) for this problem in order to minimise small residual deviations of the equilibrium structure resulting from errors in the calculation of the gravitational forces on the particles. Note that the polytropic equation of state implies that any kinetic energy that is removed by the artificial viscosity and damping terms is assumed to be radiated away immediately from the system, rather than being deposited as thermal energy. The exact static structure of the polytrope is determined by solving the Lane–Emden equation numerically using a finite difference scheme. The solution is then scaled to give a polytrope of radius and mass unity by choosing \( K = 0.4242 \) in our system of units. The resulting equilibrium configurations for the radial pressure and density distribution, azimuthally averaged in the midplane of the cloud, are shown as the dashed lines in Fig. 7 and can be compared with the exact solution indicated by the solid lines. The density and pressure profiles demonstrate a good correspondence to the expected exact solution in each case.

### 8.2.3. Boss and Bodenheimer collapse test

A third demanding problem that couples hydrodynamics and self-gravity is the collapse test described by Boss and Bodenheimer in a seminal paper in 1979 [41]. This test problem has acquired the status of a common reference for convergence testing and comparison of computer codes in the field of numerical simulations of star formation [40,42]. Here we present the results of simulations performed on a slightly modified version of the original Boss and Bodenheimer collapse test with initial conditions described in Burkert and Bodenheimer [43]. The Boss and Bodenheimer collapse test has been calculated by several authors using both AMR and SPH codes [44–48,50].

The initial condition is a uniform spherical cloud of mass \( 1 M_\odot \) with radius \( 5 \times 10^{16} \) cm and uniform temperature \( 10 \) K. The mean density of the cloud is \( \rho_0 = 3.8 \times 10^{-18} \) g cm\(^{-3}\). The cloud rotates as a solid body with angular velocity \( \omega = 7.2 \times 10^{-13} \) s\(^{-1}\). The mean molecular weight of the gas is \( \mu = 3 \). The initial ratios of the thermal and kinetic energies to the absolute value of the gravitational potential energy are \( \alpha_T = 0.26 \) and \( \alpha_k = 0.16 \), respectively. The isothermal sound speed is \( C_{\text{th}} = 1.66 \times 10^3 \) cm s\(^{-1}\) and the initial mean freefall time is \( t_f = 1.07 \times 10^{12} \) s. In addition, a small \( m = 2 \) density perturbation of the form

\[
\rho_i = \rho_0 (1 + A \sin(m\phi)),
\]

where \( A \) is the amplitude of the perturbation, is introduced to excite the disc instability. The artificial viscosity and damping terms are included to damp down the remaining oscillations.

\[
\frac{d}{dt} \left( \frac{\rho_i}{\rho_0} \right) = -\kappa \left( \frac{\rho_i}{\rho_0} \right) + \vec{f},
\]

in which we put \( \kappa = 0.05 \) and \( \vec{f} \) denotes the total force on a particle, including gravity and hydrodynamics. The artificial viscosity parameters are set to \( \alpha_{\min} = 0.1 \), \( \alpha_{\max} = 2.0 \) and \( \zeta = 0.2 \), respectively. The opening angle used by the tree code is set to \( \theta = 0.1 \) for this problem in order to minimise small residual deviations of the equilibrium structure resulting from errors in the calculation of the gravitational forces on the particles. Note that the polytropic equation of state implies that any kinetic energy that is removed by the artificial viscosity and damping terms is assumed to be radiated away immediately from the system, rather than being deposited as thermal energy. The exact static structure of the polytrope is determined by solving the Lane–Emden equation numerically using a finite difference scheme. The solution is then scaled to give a polytrope of radius and mass unity by choosing \( K = 0.4242 \) in our system of units. The resulting equilibrium configurations for the radial pressure and density distribution, azimuthally averaged in the midplane of the cloud, are shown as the dashed lines in Fig. 7 and can be compared with the exact solution indicated by the solid lines. The density and pressure profiles demonstrate a good correspondence to the expected exact solution in each case.
at a critical density two terms on the right-hand side of Eq. (64) must become equal. The gas becomes optically thick to its own radiation. In this approximation the hydrostatic behaviour of the gas during cloud collapse can, in principle, only be described consistently with a full treatment of radiative transfer at high spatial resolution. The main drawback of this approach, however, is the huge computational effort involved. We, therefore, adopt an approximate barotropic equation of state, as in Boss et al. [45] and Arreaga-García et al. [47]:

\[ P = c_{\text{iso}}^2 \rho + K \rho^\gamma, \]  

where \( \gamma = 5/3 \) denotes the adiabatic exponent in the optically thick regime and \( K \) is a constant. Eq. (57) describes the gradual transition of an initially isothermal equation of state at low densities to an adiabatic equation of state at high densities when the gas becomes optically thick to its own radiation. In this approximation, the energy equation is not explicitly solved and the pressure depends only on the density. Eq. (57) is valid for temperatures well below 100 K. In this temperature range, the rotational and vibrational states of the hydrogen molecule have not yet been excited and only translational degrees of freedom need to be taken into account [45,49]. The constant \( K \) is determined by requiring that the two terms on the right-hand side of Eq. (64) must become equal at a critical density \( \rho_{\text{crit}} \), which separates the isothermal from the adiabatic regime:

\[ K = c_{\text{iso}}^2 \rho_{\text{crit}}^{-\gamma}. \]

which in which we adopt a value for \( \rho_{\text{crit}} = 5 \times 10^{-14} \text{ g cm}^{-3} \) according to Boss et al. [45]. The initial condition described above is implemented in our SPH code by placing equal-mass particles on a hexagonal closely packed lattice and retaining all particles within a sphere of radius 1. We then move each particle from its unperturbed position in spherical polar coordinates \((r, \phi, z)\) to its perturbed position \( \phi^* \) in the azimuthal direction according to the solution of the equation

\[ \phi^* + \frac{A \sin(m\phi^*)}{m} = \phi. \]  

The collapse/fragmentation problem described above is demanding because the fragmentation only happens in the very center of the cloud. Thus, for grid codes one has to make use of adaptive mesh refinement (AMR) techniques to achieve the required spatial resolution. SPH, on the other hand, provides inherently adaptive spatial resolution because the smoothing length of the particles decreases in the regions of high density. A second problem arises because the mass fraction contained in the region where the fragments form is small compared to the total mass of the cloud. We therefore need a considerable number of SPH particles in the initial condition because only a small fraction of them will end up in the fragments near the center of the cloud. Bate and Burkert [50] showed that SPH simulations have to satisfy a resolution criterion to avoid artificial fragmentation when applied to problems of gravitational collapse. The minimum resolvable mass \( M_{\text{min}} = 2N_{\text{neigh}} \rho_{\text{opt}} \), where \( m \) denotes the mass of a particle and \( N_{\text{neigh}} \) is the number of neighbours, must be smaller than the local Jeans mass:

\[ M_J = \frac{\pi^{3/2} c_{\text{iso}}^3}{G^{1/2} \rho_{\text{opt}}^{3/2}}. \]  

Following Arreaga et al. [47], the Jeans condition can be expressed as a condition on the maximum mass of a particle, and becomes for \( \rho = \rho_{\text{crit}} \):

\[ m < \frac{\pi^{3/2} c_{\text{iso}}^3}{2N_{\text{neigh}} \rho_{\text{crit}}^{1/2} G^{3/2}}. \]  

This equation can be used to estimate the required number of particles to follow the collapse in the isothermal regime, where the Jeans mass decreases with rising densities, up to the point where the collapse ceases to be isothermal and the Jeans mass increases again due to the adiabatic rise in temperature. We performed a simulation of the Boss and Bodenheimer collapse test with 195,525 particles. For this choice of the number of particles, the Jeans condition is satisfied at the critical density since the mass of a particle is \( 5.1 \times 10^{-6} \text{ M}_\odot \) and the right-hand side of Eq. (68) gives an upper limit of \( 3.31 \times 10^{-5} \text{ M}_\odot \). We also used the time-dependent artificial viscosity terms with \( \alpha_{\text{min}} = 1.0 \) and \( \alpha_{\text{max}} = 2.0 \) and \( \eta_{\text{opt}} \) was set to 50. Fig. 8 shows successive snapshots of the density distribution in the midplane of the cloud for 6 different times during the evolution of the simulation. The panels show the density scaled to the initial value of the density on a logarithmic scale. The unit of length in the plots equals the initial radius of the cloud. The series of snapshots begins at 1.3 \( t_G \) after the start of the simulation. At this stage of the collapse, the maximum density has exceeded the critical density \( \rho_{\text{crit}} \) and the isothermality of the equation of state starts to break down. Two protostellar fragments have been formed in the center of the cloud. As shown in Fig. 8a, these two objects are connected though a thin bar which starts to become optically thick and ceases to collapse further upon itself due to the increasing pressure gradients. The bar and the two protostellar fragments are embedded in a two-armed spiral structure, which reflects the symmetry of the original density perturbation (Fig. 8b). The subsequent evolution of the binary system can be followed in Figs. 8c–8f. First, the protostars start to accrete the low angular momentum gas which makes up the bar in between them. This accretion of low angular momentum gas causes the two protostars to spiral in toward each other. Simultaneously, the spiral arms are wrapping up and becoming more elongated (Figs. 8c and 8d). The separation between the protostars reaches a minimum of \( \sim 58 \text{ AU} \).
at $t = 1.355 \, t_{ff}$, after which the protostars begin to accrete matter from the spiral arms. The higher angular momentum of this gas causes the fragments to move away from each other again. At the end of the simulation at $t = 1.4 \, t_{ff}$, the spiral arms are starting to evolve into a kind of circumbinary disk structure surrounding the two protostars. The separation between the protostars at the time of the final snapshot shown in Fig. 8 is $\sim 274 \, \text{AU}$.

The overall evolution illustrated by the snapshots in Fig. 8 is morphologically very similar to that reported by Arreaga-Garcia et al. [47] and the AMR simulations described in Klein et al. [48]. We emphasise, however, that some quantitative differences still remain between the simulations. For their model U2B, Arreaga-Garcia et al. report a minimum separation of 89 AU between the fragments at $t = 1.3549 \, t_{ff}$. On the other hand, Klein et al. report a minimum separation of 44 AU in their Fig. 3. The minimum separation found with GRADSPH lies in between the values found by Klein et al. and Arreaga-Garcia et al. The last authors quote a final separation of 274 AU, which is close to the result obtained with GRADSPH. We also calculated the mass of the fragments at the end of the simulation by searching for the particle with the maximum density in each of the fragments and adding the mass of all particles in the volume surrounding this particle whose density is larger than a fraction $f$ of the densest particle. Choosing $f = 0.001$ yields final masses of $m_1 = 0.0894$ and $m_2 = 0.0864$, respectively, in units of the initial mass of the cloud, indicating that about $\sim 18\%$ of the mass of the cloud ends up in the fragments, in reasonable agreement with Arreaga-Garcia et al. [47] ($m_1 = 0.0929$ and $m_2 = 0.0903$) and Klein et al. [48], who report fragment masses of 0.08 $M_\odot$ for their Fig. 6. In Fig. 9, we compare the evolution of the maximum density of the cloud during the course of the simulation.
with the result obtained by Arreaga-García et al. using Gadget-2. Both curves report the maximum density divided by the initial average density of the cloud on a logarithmic scale. Gadget-2 overestimates the maximum density of the initial condition (Jaime Klapp, private communication), and yields a systematically higher density up to $t \sim 1.3 \, \text{t}_{\text{ff}}$, after which the situation is reversed at the end of the simulation. The final maximum density obtained with GRADSPH is $2.6 \times 10^{-11} \, \text{g cm}^{-3}$ compared to $1.43 \times 10^{-11} \, \text{g cm}^{-3}$ using Gadget-2. Klein et al. report final densities which are still a factor 4–5 higher. Although the number of particles in our simulation is smaller than that used in simulation U2B of Arreaga-García et al., which used 1.2 million particles, tests with increasing numbers of particles indicate that our model is also close to convergence. We conclude that the overall features of the collapse are reasonably well reproduced by the various codes used to model this problem, although the quantitative predictions are still subject to a considerable margin of uncertainty.

9. Program structure and execution

This section describes the overall structure of GRADSPH, lists the various subroutines contained in the distribution package and provides details on how to compile and use the program.

9.1. Main program and module

The main program file is called GRADSPH. The program makes use of a FORTRAN 90 module file called GRADSPH.h. This file includes the definition of global parameters and constants, as well as COMMON blocks and global arrays that are initialised dynamically during program execution. The public version of GRADSPH contains a three-dimensional implementation of the algorithms described above and uses an ideal gas equation of state to describe the thermodynamics of the gas. The equation of state can be freely modified by the user.

9.2. List of subroutines

This section lists all the main subroutines and functions used by the program with a brief description of each one.

Subroutine INTRUN reads the input parameter file run.pars and initialises the values of the global constants and parameters that are not changed during execution of the program.

Subroutine PARARANGE determines the workload for each processor.

Subroutine INITTABLES computes tabulated values of the smoothing kernel function as well as the gradient of the kernel function for use in the SPH summation formulas.

Subroutine FINDIRHO computes the densities and smoothing lengths of the particles, and computes the quantities $\Omega$ and $\zeta$ for each particle as well.

Subroutine RHOS computes the density of a particle.

Subroutine ADVANCE evolves the system during a single time step.

Subroutine TIMESTEP determines the individual time step of each particle and the system time step.

Subroutine CALCDEIVS calculates the total acceleration of a particle, the rate of change of its internal energy and viscosity parameter, and the time derivative of the smoothing length.

Subroutine CALCDIV calculates $\nabla \cdot \vec{v}$ for a particle.

Subroutine WRITESTAT calculates diagnostic quantities for each time step and writes them to an output file.

Subroutine PDUMP writes a particle dump to an output file. It is called by the main program.

Function EOS determines the pressure using the equation of state. This routine can be adapted by the user.

Function W calculates the kernel function.

Function DW calculates the gradient of the kernel function.

Functions FG and GG calculate smoothing functions used for softening the gravitational forces.

Function DFG calculates the derivative of the smoothing function defined in Function FG.

Function DWDH calculates the derivative of the kernel function with respect to the smoothing length.

Subroutine TRG calculates the gravitational acceleration of a particle using the Barnes–Hut tree.

Subroutine TRN determines the neighbour list of a particle using the Barnes–Hut tree.

Subroutine MICTREE constructs the Barnes–Hut tree and loads particles into the tree.

Subroutine HACKM determines the properties which characterise the cells of the tree, including the quadrupole moments if required.

Subroutine BFLIST orders the cells in the tree according to their size.

Function SBINDEX determines the subcell indices within the nodes of the Barnes–Hut tree.

Function MKCELL adds a cell to the tree structure.

Subroutine LDDBODY loads a particle into the tree.

Subroutine OUTERR outputs error messages related to the tree code.

9.3. Data input

GRADSPH does not provide the user with subroutines that can be used for constructing the initial conditions. The user can use any program to do this, provided that the input file has the structure described here. The input file simply contains a list of the position vector, the velocity vector, the mass, an initial value of the smoothing length and the internal energy for each particle in FORTRAN DOUBLE PRECISION format. The following piece of FORTRAN 90 code illustrates how a suitable input file may be created as follows:

```fortran
OPEN(1,FILE=INITFILE,FORM='UNFORMATTED',STATUS='NEW')
DO i=1,NSPH
WRITE(1) x(1,i),x(2,i),x(3,i),v(1,i),v(2,i),v(3,i),h(i),ui(i)
```
ENDDO
CLOSE(1)

In the above code excerpt, nsph denotes the total number of particles. Note that the arrays x and v should be defined as x(3,1:nsph) and v(3,1:nsph). The program allows the use of a different mass for each particle. The initial value of the smoothing length is used as a first guess for the iteration loop in subroutine FINDHHRO. GRADSPH accepts any system of units for the input data, but assumes a gravitational constant $G = 1$ in order to avoid unnecessary floating point operations. Note also that we assume that all SPH particles have the same mass in the current implementation of GRADSPH. We therefore include the particle mass as a parameter in the input file run_pars, the contents of which are described in the next section.

9.4. Parameter and output files

9.4.1. The run_pars file

This file contains some data which identify the simulation and provide the program with input parameters that will not be modified during the execution of the program. The file has the following structure:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILEHEADER</td>
<td>evrard</td>
</tr>
<tr>
<td>INITFILE</td>
<td>evrard.dat</td>
</tr>
<tr>
<td>NSPH</td>
<td>9860</td>
</tr>
<tr>
<td>MAXWALLTIME</td>
<td>100000.0</td>
</tr>
<tr>
<td>MAXSTEPS</td>
<td>6000</td>
</tr>
<tr>
<td>NSTEPS</td>
<td>300</td>
</tr>
<tr>
<td>CN</td>
<td>0.1</td>
</tr>
<tr>
<td>AMIN</td>
<td>0.1</td>
</tr>
<tr>
<td>AMAX</td>
<td>0.5</td>
</tr>
<tr>
<td>GAM</td>
<td>1.666667</td>
</tr>
<tr>
<td>NOPT</td>
<td>50</td>
</tr>
<tr>
<td>THETA</td>
<td>0.1</td>
</tr>
<tr>
<td>USQUAD</td>
<td>T</td>
</tr>
<tr>
<td>COND</td>
<td>F</td>
</tr>
</tbody>
</table>

Below we describe briefly the meaning of the above parameters:

- **FILEHEADER** is a string which identifies the simulation. Its length should not exceed 10 characters. Upon execution, the program sequentially creates output files of the form FILEHEADER001, FILEHEADER002, etc.
- **INITFILE** is a string which contains the name of the file providing the initial conditions. Its length should not exceed 30 characters.
- **NSPH** is the number of particles in the simulation.
- **MAXWALLTIME** is the maximum wallclock time allowed for program execution expressed in sec. Note that we are not referring here to the total CPU time of all processes combined. The program stops when this limit is exceeded.
- **MAXSTEPS** is the maximum number of integration steps allowed by the user.
- **NSTEPS** is the number of integration steps between particle dumps.
- **CN** is the Courant number.
- **MDT** is the minimum system time step allowed by the user. The program halts execution when the system time step falls below this limit.
- **MP** is the mass of a single SPH particle.
- **AMIN** is the minimum value of the time-dependent artificial viscosity parameter associated with each particle.
- **AMAX** is the maximum value of the time-dependent artificial viscosity parameter associated with each particle.
- **CA** is value of the constant defined in Eq. (35).
- **GAM** is the adiabatic exponent.
- **NOPT** is the optimal number of neighbours.
- **USQUAD** is a flag which indicates if quadrupole moments should be included in the evaluation of the gravitational forces.
- **COND** is a flag which indicates if artificial conductivity terms should be included in the energy equation.

The values of the parameters given above are for an Evrard collapse test with 9860 particles. This example serves as our typical testcase for the program. The associated initial conditions and output files are provided with the public release and described in Section 9.6.

9.4.2. The stat_pars file

The ASCII file stat_pars contains diagnostic output that is computed by the subroutine WRITESTAT for each time step. WRITESTAT first determines the maximum and minimum values of the particle coordinates associated with the 3 spatial directions. This determines the system box. It then computes the total linear and angular momentum of the system of particles, as well as the total kinetic, potential and internal energy and sums these values to obtain the total energy. The conserved quantities are computed as follows:

\[
J_T = \sum_{i=1}^{N} m_i \left| \mathbf{r}_i \times \mathbf{v}_i \right|, \\
P_T = \sum_{i=1}^{N} m_i \left| \mathbf{v}_i \right|, \\
E_K = 0.5 \sum_{i=1}^{N} m_i \left| \mathbf{v}_i \right|^2, \\
E_P = 0.5 \sum_{i=1}^{N} m_i \Phi_i, \\
E_T = \sum_{i=1}^{N} m_i u_i, \\
E_{\text{r}} = E_T + E_K + E_P. 
\]

In addition, the program determines the maximum and minimum densities, the maximum and minimum values of the smoothing length and the maximum and minimum values of the internal energy for the entire system of particles and determines the index and coordinates of the particles associated with these values. GRADSPH also outputs messages concerning the status of the program execution and provides information on the time when particle dump files are created.

9.4.3. Format of dump files

GRADSPH creates particle dump files in exactly the same format as the initial conditions described in Section 9.3, except for an additional header line which contains the time, in code units, when the program has written the output file. In FORTRAN 90, a dump file could be read easily as follows:

```
OPEN(1,FILE='evrard001',FORM='UNFORMATTED',STATUS='OLD')
READ(1) TDUMP
DO i=1,NSPH
  READ(1) x(1,i),x(2,i),x(3,i),v(1,i),v(2,i),v(3,i),h(i),ui(i)
ENDDO
```
For displaying and processing of SPH data, the current version of GRADSPH does not provide tools of its own. Instead we refer the reader to excellent visualisation tools like SPLASH [51] which can be easily combined with the output of GRADSPH (see also http://www.maths.monash.edu.au/dprice/splash/).

9.5. Installing GRADSPH and starting a simulation

The GRADSPH installation is straightforward and involves the following steps. Download the file GRADSPH.tar.gz and copy it somewhere in one of your local subdirectories on a UNIX system. Once you have downloaded the package, the following commands can be used to unpack and unfold the files:

- gunzip GRADSPH-1.0.tar.gz,
- tar -xvf GRADSPH-1.0.tar.

The above commands will generate the following subdirectories:

- /GRADSPH-1.0 (GRADSPH installation directory).
- /GRADSPH/SRC (Source code).
- /GRADSPH/TESTCASE (Input and output files for a test case).

We provide a makefile for compiling the program in subdirectory /GRADSPH/SRC. The name and location of the executable can be modified if desired. Before starting a run, the user needs to edit the header file GRADSPH_h.f90 and set the value of the constants nmax and nproc to the number of particles used in the initial conditions and the number of processors for parallel execution, respectively. The value of nmax must be the same as the variable nspf in the input file run_pars. In addition, we also provide a script file called GRADSPH_PBS_script which can be used for running GRADSPH on a parallel computing cluster if Portable Batch System (PBS) is available. Depending on the details of his/her system, the user can make the necessary changes to this file.

9.6. Testcase

In order to check that GRADSPH is working correctly, the user should check the testcase reported in the GRADSPH distribution (subdirectory /GRADSPH/TESTCASE). This directory contains the setup for an Evrard collapse test with 9860 particles in files evrard.dat and run_pars. Upon execution, the program runs this test with MAXSTEPS=6000 and NSTEPS=300 and should produce the output given in the parameter file stat_pars and the output files evrard001–evrard020. We use the same system of units as in Section 8.2.1. Note that because \( g = 1 \) in this system of units, the unit of time should be multiplied by a factor \( \pi / 2 \sqrt{2} \) to obtain consistency with Fig. 4.

10. Scalability and performance

The performance of a parallel simulation code is dependent on many factors, such as the number of processors used, the number of resolution elements, and basic numerical parameters of the algorithm. It is therefore difficult to compare different simulation codes objectively, and we therefore restrict ourselves in this section to a basic characterisation of the performance and scalability of GRADSPH. We present scalability tests of GRADSPH performed on VIC, the HPC cluster of the KU Leuven. This system has been used while testing and developing GRADSPH. In its current configuration VIC contains 305 nodes connected via an infiniband network. Most of the nodes contain 2 AMD Opteron CPUs, each containing 2 computing cores with 2 GB RAM memory. The code was compiled using the PathScale EKOPATH compiler suite version 2.9.99. We have used the following compilation option:

```
mpif90 -o3
```

where 03 indicates aggressive optimisation. Rather than measuring the performance of a simulation code in floating point operations per second, we prefer to use the science rate, which is defined as the number of resolution elements processed per second (i.e. SPH particles in our case) as a more practical way of estimating the speed of the code. For tightly coupled gravitational problems such as the Evrard test considered here, perfect linear scaling up to a large number of processors can generally not be achieved. This happens mainly because of two reasons: (1) The more processors are being used, the harder it becomes to balance the workload among them, and the more time is being lost because of processors waiting idle. This issue becomes aggravated when variable time stepping is used because load balancing is more difficult in this case. In the first version of GRADSPH, we did not attempt to implement a load balancing strategy. (2) When more processors are involved, the workload for each processor diminishes and the computing/communication ratio decreases. The update of arrays in the memory of the processors implies a considerable amount of communication in our parallelisation scheme.

Figs. 10 and 11 illustrate the scalability of GRADSPH on VIC. Fig. 10 shows a graphical representation of the code scalability for an Evrard collapse test with 146572 particles. In this plot, the average science rate of the first 10 integration steps is shown as a function of the number of processors while the number of particles is kept constant. The squares indicated in Fig. 10 indicate the science rates when all particles are simultaneously updated and the variable time stepping scheme is not active. The code scales reasonably well up to 32 processors, although the speedup is below linear because of the huge communication costs, after which the science rate levels off. GRADSPH is still making some gains when the number of processors is further increased up to 64. The science rates indicated by the stars demonstrate the benefit of adaptive time stepping. The increase in performance obtained with the variable time stepping scheme described in Section 6 is a factor 2 for this test, although the actual speedup is highly dependent on particle distribution and geometry. When increasing the number of processors up to 64, we see a slight decrease of the science rate in this case, because of the additional load balancing issues mentioned above. Fig. 11 shows the performance of GRADSPH when the total size of the simulations is increased for a fixed number of processors. GRADSPH performs reasonably well on this test, with science rates decreasing by a factor 2–3 when the number of particles increases eightfold. The total number of particles that can be included in a simulation is limited by the RAM memory of each processor for the current version of GRADSPH. The simulation using 1.2 million particles illustrated in Fig. 11 consumes ~ 225 MB RAM memory allocated on each processor. With 2 GB of available RAM memory on each processor, the size of the total workload can therefore be increased to 10 million particles, which is sufficient for most practical purposes, except for large scale simulations in cosmology.

Table 1 provides a detailed breakdown of the CPU time spent by the program in its different parts. We again consider an Evrard collapse test with NSPH=146572. The program is executed for 10 integration steps with 4 and 16 processors, respectively. The first line in the table gives the total wallclock times measured on VIC. The total wallclock time is then broken up in its relative contributions associated with each part of the program. The floating point operations associated with the predictor step and the execution of the output subroutine WRITESTAT consume only negligible amounts of CPU time and are therefore not listed separately in the
the communication between the processors with our simple partition increases considerably from 26.1% for NPROC CPU time. As expected, the relative time spent during communication for finding neighbours and for computing the gravitational accelerations and updating the active particles during the corrector step in subroutine FINDHRHO table. Table 1 shows that the program spends most of the time updating smoothing lengths and densities in subroutine FINDHRHO and updating the active particles during the corrector step in subroutine ADVANCE. Both operations involve repeated tree traversals for finding neighbours and for computing the gravitational accelerations of the particles. Together these operations take up a lot of CPU time. As expected, the relative time spent during communication increases considerably from 26.1% for NPROC=4 to 44.4% for NPROC=16.

Since we do not attempt to balance the Barnes–Hut tree and the communication between the processors with our simple parallelisation strategy, it is important to estimate the impact of load imbalance on the program’s performance. To this end, we introduce two additional metrics to define load imbalance for parallel programs [52]. The first one is the imbalance percentage, which is defined as

\[ \text{imbalance percentage} = \frac{\text{maximum time} - \text{average time}}{\text{maximum time}} \times 100 \]  

(70)

This metric can be viewed as a measure of the percentage of time the system of processors, excluding the slowest one, is not doing useful work on the particular task at hand while waiting for the slowest element to complete its contribution. If the other processes are idle, this time is effectively wasted for parallelism. When expressed as a number between 0 and 100, a perfectly balanced region of the program would have zero imbalance and a serial program segment executed on only one processor would have an imbalance percentage of 100. A highly imbalanced part of a program does not necessarily require much optimisation, because its contribution to the total could be negligible. We therefore define a second metric, which indicates how much time in the overall program could be saved if the considered program segment is completely balanced. We call this statistic the imbalance time and define it simply as:

\[ \text{imbalance time} = \text{maximum time} - \text{average time} \]  

(71)

The imbalance time for a code section is only an upper limit to the potential saving that could result from perfect load balancing, since it does assume that the other processors are simply waiting for the slowest one to finish. In Tables 2 and 3, we give the imbalance times and the imbalance percentages for the program sections mentioned in Table 1. The imbalance times are suitably expressed as fractions of the total wallclock time. As can be seen from Table 2, the imbalance times increase for a larger number of processors and can take more than half of the CPU time, indicating that a factor of about 2 increase in speed could be attained by improving load balancing of the tree and the communication calls. The imbalance percentages show a similar pattern, except for the communication part which improves its load balancing percentage for NPROC=16, while slightly increasing its imbalance time. We expect that two upcoming upgrades of GRADSPH will considerably improve load balancing and communication overload, as well as the overall performance of the program:

Table 2

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>NPROC=4</th>
<th>NPROC=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoothing length update</td>
<td>11.0%</td>
<td>15.0%</td>
</tr>
<tr>
<td>Corrector step</td>
<td>14.3%</td>
<td>16.0%</td>
</tr>
<tr>
<td>Communication</td>
<td>25.0%</td>
<td>32.0%</td>
</tr>
</tbody>
</table>

The program is executed on VIC for two processor numbers: NPROC=4 and NPROC=16. The imbalance times are expressed as relative contributions to the total elapsed walltime. See text for details.

Table 3

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>NPROC=4</th>
<th>NPROC=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoothing length update</td>
<td>35.3%</td>
<td>41.6%</td>
</tr>
<tr>
<td>Corrector step</td>
<td>33.3%</td>
<td>41.4%</td>
</tr>
<tr>
<td>Communication</td>
<td>65.2%</td>
<td>44.7%</td>
</tr>
</tbody>
</table>

The program is executed on VIC for two processor numbers: NPROC=4 and NPROC=16. See text for details.
Table 4
Science rates obtained when running GRADSPH on VIC for a single time step of the Boss and Bodenheimer collapse test with NSPH=195525.

<table>
<thead>
<tr>
<th>Science</th>
<th>Rate (1/time step)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gravity+SPH</td>
<td>10554</td>
</tr>
<tr>
<td>SPH only</td>
<td>11113</td>
</tr>
<tr>
<td>Gravity only</td>
<td>28633</td>
</tr>
</tbody>
</table>

The timings were computed for the clustered state at $t=1.4\,t_f$ using 32 processors. The table compares the total science rate to the science rate obtained when the particles are updated using only gravitational or hydrodynamical forces.

- Optimisation of neighbour search and gravitational force calculation by adopting a grouping strategy at a suitable level of the tree [53].
- Implementation of a more sophisticated parallelisation model in which the particles are distributed among the processors according to the first levels of the Barnes–Hut tree, together with a load balancing strategy based on buffering and one-sided MPI calls. This strategy has already been implemented in some existing N-body codes [54].

Because both hydrodynamic and gravitational forces are updated together during the correction step, it is interesting to look at their separate contributions when other parts of the code are not being executed. Table 4 shows the science rates for a single time step of a Boss and Bodenheimer collapse test described in Section 8.2.3 when the clustered state has been reached at $t=1.4\,t_f$. One can see that the total science rate is dominated by SPH, mainly because of the additional tree traversals which are necessary to iteratively update the densities and smoothing lengths.

11. Summary and outlook

We have described the first version of GRADSPH, a parallel TREESPH code which combines state of the art features, including spatially variable smoothing lengths and a variable time stepping procedure, with a tree-based approach for calculating self-gravitating flows. The code can be freely downloaded from the CPC program library. Future versions of GRADSPH will include an improved parallelisation model and associated load balancing strategy to further enhance its performance on massively parallel computing systems.

We intend to use GRADSPH in future parametric studies of self-gravitating protostellar systems. To better model the reality of astrophysical systems, the standard SPH method will have to be extended to incorporate additional physics. In particular, the dynamical influence of magnetic fields needs to be included properly in the gridless approach. Also, the shock-capturing capabilities of current SPH codes can possibly still be improved by replacing the artificial viscosity treatment with a more Riemann-solver based method. How to exploit these techniques in an extended SPH framework is the subject of ongoing work.

Acknowledgements

We gratefully acknowledge the help of Dr. Wim Obbels for his support in using the HPC cluster VIC operated by the KU Leuven, and Dr. A. Serna for discussions on the implementation of variable time stepping procedures. In addition, Prof. Dr. M. Bate and Prof. Dr. P. De Causmaecker are gratefully acknowledged for their support and stimulating discussions during this work. We also would especially like to thank Prof. Matthias Steinmetz for the PPM solution of the Evrard collapse test and Dr. Jaime Klapp for sharing their numerical data of the Boss and Bodenheimer collapse test with us.

References


