
High-Performance Computing and Smoothed Particle Hydrodynamics

C. Moulinec¹, R. Issa², D. Latino³, P. Vezolle⁴, D.R. Emerson¹ and X.J. Gu¹

¹ STFC Daresbury Laboratory, WA4 4AD, UK charles.moulinec@stfc.ac.uk

² EDF R&D, National Hydraulics and Environment Laboratory, and Saint-Venant Laboratory for Hydraulics, 6, quai Watier, 78400 Chatou, France reza.issa@edf.fr

³ IBM Middle East, Dubai Internet City, P.O. Box 27242, Dubai, UAE
dlatino@ae.ibm.com

⁴ IBM France, Rue de la Vieille Poste, BP 1021, F-34006, Montpellier, France
vezolle@fr.ibm.com

Abstract

Smoothed Particle Hydrodynamics is a gridless numerical method that can be used to simulate highly complex flows with one or more free surfaces. In the context of engineering applications, very few 3-D simulations have been carried out due to prohibitive computational cost since the number of particles required in 3-D is usually too large to be handled by a single processor. In this work, an improved version of a parallel 3-D SPH code, Spartacus-3D, is presented. Modifications to the code, which include a localisation of all the previously global arrays combined with a switch from global communications to local ones where possible, lead to a more efficient parallel code and allow for a substantial increase in the number of particles in a simulation.

1 Introduction

The Smoothed Particle Hydrodynamics (SPH) method has been successfully applied in fluid mechanics to simulate strongly distorted free surface flows in situations where classical Eulerian Computational Fluid Dynamics (CFD) approaches would fail because of grid steepness [2] [3] [4]. However, 3-D SPH simulations remain costly as a large number of particles are necessary in order to accurately simulate such flows and because of the nature of classical SPH, which requires a homogeneous initial particle distribution for incompressible flows. A parallel 3-D SPH code, namely Spartacus-3D, was developed in 2006 based on the MPI library [5], but most of the arrays were global. This approach limited the total number of particles that could be handled and the memory required by processors was largely over-estimated. Moreover, global communications were compulsory.

The Spartacus-2D code was developed in 1999 at EDF R&D, mainly for modelling coastal and environmental applications, such as spillways, dam breaking, and breaking waves. Based on this 2-D version, a 3-D version was first developed [1] in 2004 and initially parallelised in 2006 [5]. In this paper, details concerning the latest version of the code, where the focus has been on array localisation, will be presented. In Section 2, the methodology is explained and Section 3 introduces the equations in SPH framework. Section 4 discusses parallelisation and Section 5 highlights the new results and we conclude with some final remarks Section 6.

2 Methodology

The pseudo-compressible Navier-Stokes equations written in Lagrangian form for Newtonian fluids and incompressible, laminar flows augmented by the position equation read:

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla p + \nabla \cdot (\nu \nabla \mathbf{u}) + \mathbf{F}^e, \quad \frac{D\rho}{Dt} = \rho \nabla \cdot \mathbf{u}, \quad \frac{D\mathbf{r}}{Dt} = \mathbf{u}. \quad (1)$$

where D/Dt is a Lagrangian derivative, $\mathbf{u}$ the velocity vector, t the time, ρ the density, p the pressure, ν the kinematic viscosity, and $\mathbf{F}^e$ an external force. In addition, ' ∇ ' and ' $\nabla \cdot$ ' are the gradient and divergence operators, respectively. This system is closed by the following equation of state:

$$p = \frac{\rho_0 c_0^2}{\gamma} \left[\left(\frac{\rho}{\rho_0} \right)^\gamma - 1 \right] \quad (2)$$

where ρ_0 is the reference density, c_0 a numerical speed of sound, and γ a constant coefficient. Equations 1 are discretised explicitly in time (first order) and the SPH approach is used to perform the spatial discretisation.

3 Equations in SPH framework

The SPH continuity equation and momentum equation are given by:

$$\frac{D\rho_a}{Dt} = \rho_a \sum_b \frac{m_b}{\rho_b} \mathbf{u}_{ab} \cdot \nabla_a w_{ab}, \quad (3)$$

$$\frac{D\mathbf{u}_a}{Dt} = - \sum_b m_b \frac{p_a + p_b}{\rho_a \rho_b} \nabla_a w_{ab} + 16 \sum_b \frac{\nu}{\rho_a + \rho_b} \frac{\mathbf{u}_{ab} \cdot \mathbf{r}_{ab}}{r_{ab}^2} \nabla_a w_{ab} + \mathbf{F}_a^e, \quad (4)$$

where the subscripts a and b , respectively, represent the particle which the operators are calculated for, and its neighbours contained in the kernel compact support, $\mathbf{u}_{ab} = \mathbf{u}_a - \mathbf{u}_b$, $w_{ab} = w_h(r_{ab})$, $r_{ab} = |\mathbf{r}_{ab}| = |\mathbf{r}_a - \mathbf{r}_b|$. w_h is the kernel as a function of the smoothing length, $\nabla_a w_{ab}$ is the gradient of the kernel with respect to a , m is the

particle mass, and $\mathbf{F}_a^e$ is an external force applied to particle a . Note that the pressure gradient expression selected here is anti-symmetric with respect to the subscripts a and b and the viscous term is symmetric. This allows a reduction in computational cost because, when calculating the SPH operators, b 's contribution can always be deduced from a 's.

4 Parallelisation

Equations 3 and 4 show that SPH operators are expressed as differences or sums of contribution from particles a and b , with a sum on b . Due to the particle motion and the fact that neighbouring particles (neighbouring particles of particle a are particles which belong to the compact support, whose centre is the position of a) at a given time step might not be neighbouring particles at the next time step, the search for particles b has to be performed at each temporal iteration. The CPU time would normally scale as N^3 (in 3-D) where N denotes the total number of particles, if the search for neighbours would be carried out over the whole set of particles. Since spline kernels have a compact support, each particle a is only linked to its closest neighbours b for which $r_{ab} < h_t$, where h_t is the kernel compact support size. A coarse Cartesian grid made of homogeneous cubic cells embeds the physical domain to speed-up the search for the links.

Profiling the serial version of Spartacus-3D shows that the search for particle a 's links is the most expensive part of a temporal iteration, consuming up to 60% of the CPU time. When running in parallel, another section that takes a lot of time is the rebuilding of the list of particles by re-indexing them at each temporal iteration, which is done in order to minimise the communications between processors in the equation resolution stage. Some problems identified in the previous parallel version of the code came from the declaration of global arrays for the main variables (velocity, density, pressure, viscosity,...) and from the global communications required to build SPH operators. The way to circumvent most of these problems is described in the following, with a special focus on the attempt to use arrays whose size varies per processor, depending on the number of particles of a given processor augmented by the number of particles located on other processors, which play a role in building the SPH operators.

In the new parallel version of Spartacus-3D, four main steps are carried out per temporal iteration. At the $(n+1)^{th}$ iteration, for instance it yields:

- Step 1: Generation of the new particle list,
- Step 2: Search for particle links,
- Step 3: Link particle re-indexing,
- Step 4: Resolution of the equations.

Steps 1 to 4 are detailed in the following sections.

4.1 Step 1. Generation of the new particle list

If the total number of processors is NP the general idea is to approximately assign N/NP particles a per processor to ensure load-balancing, while reducing the number of processors containing the neighbours b of the N/NP particles a . In the present version of the code, there is no weight depending on the computing effort required per particle, even though different operations are carried out for wall (respectively dummy) particles and fluid ones.

- The Cartesian grid is generated and the cubes denoted by C_i ($i = 1, NC$) are linearly ordered, where NC is the whole number of cubes. The loop is carried out over NC . This is parallelised.
- Denoting the number of particles per cube C_i as NPC_i , a temporary array $DISPTB_i$ is built from the NPC_i s as the accumulated number of particles contained in the cubes already listed until cube of index ' i ', i.e. $DISPTB_i = \sum_{j=1}^i NPC_j$. The loop is carried out over NC .
- NP blocks BLO_l are built from $DISPTB_i$, with approximately N/NP particles per block. The loop is over NC .
- The next step consists of detecting which particles (whose index is defined in the list corresponding to iteration n) located on a given processor $PROC_k$ belong to a given block BLO_l as defined in the previous item. At the end of this operation, each processor has access to the number of its particles located on block BLO_l , and to the local list of its own particles per block. Both arrays are not complete as the loop is performed over the particles present on each processor $PROC_k$ at iteration n , and not over the total number of particles N .
- The full local list assembled on the master node is then broadcast to the other processors. Note that this array is still global in this version of the code.
- The global list is built up from the local list.

4.2 Step 2. Search for particle links

Links between particles a and their neighbouring particles b used in the calculation of the operators are locally constructed by processors, with the help of the cube structure. The next step differs from the previous version of the code, in the sense that arrays containing the main variables are now locally defined rather than globally. For each particle located on a given processor, the neighbouring particles (which may not be on the same processor) are gathered in extra arrays, and the rank of those processors stored. If the first loop of Step 2 goes over particle a located on processor $PROC_k$ as before, the second one goes on the neighbouring cubes of the cube containing particles a , taking into account the particles not located on the processor where particle a is.

4.3 Step 3. Link particle re-indexing

After the search for the links, the particle links are re-indexed in order to limit the communications between processors in the calculation phase. This means that the array containing the list of neighbouring particles has to be updated.

4.4 Step 4. Resolution of the equations

The equations are solved explicitly. The operators are built per processor, with loops going over the list of particles on $PROC_k$ at iteration $n + 1$. The neighbouring particles, which may not be located on the processor of concern, are collected in an external array and used to calculate the various operators. As a result, only local communication is necessary to build these operators.

5 Results

5.1 Description of the Machines

Simulations have been performed on Blue Gene/L, BlueGene/P and IBM AIX systems. Both single rack Blue Gene systems contain 1,024 chips, with 2 processor cores per chip in the L system and 4 processor cores per chip in the P system, giving a total of 2,048 cores and 4,096 cores, respectively. Memory is provided at 512 Mbytes per core in both systems. The basic processor in the L system is the Power 440 running at 700 MHz, whilst the P system uses a processor from Power 450 family running at 850 MHz. The HPCx system uses an IBM eServer 575 nodes for the compute and IBM eServer 575 nodes for login and disk I/O. Each eServer node contains 16 1.5 GHz 2Gb memory POWER5 processors. The main HPCx service provides 160 nodes for compute jobs for users, giving a total of 2,560 processors [6].

5.2 Accuracy

The first test is carried out to validate this laminar parallel version against the analytical Poiseuille solution. The Reynolds number based on the bulk velocity is equal to 100. The run is performed on 128 cores of BG/P. Figure 1 (Left) shows that Spartacus-3D's output compares well to the analytical solution.

5.3 Performance of the Code

Due to the update of the particle positions at each time step, each iteration consists of both a new partitioning to ensure load-balancing and the fluid dynamics calculation itself. For this approach, two types of performance are analyzed showing the total CPU time per iteration per processor, and the CPU time for the dynamics per iteration per processor, to be able to compare with any other CFD Eulerian software dealing with non-moving grids.

The results of the previous parallel version of the code are shown in Fig. 1 (Right) for 355,000 particles. This was carried out on a cluster of thin nodes, each of them having 4Gb memory per processor. The code scales well up to 8 processors, but from 16 processors it can be seen that the generalized use of global communications has an important impact on the performance.

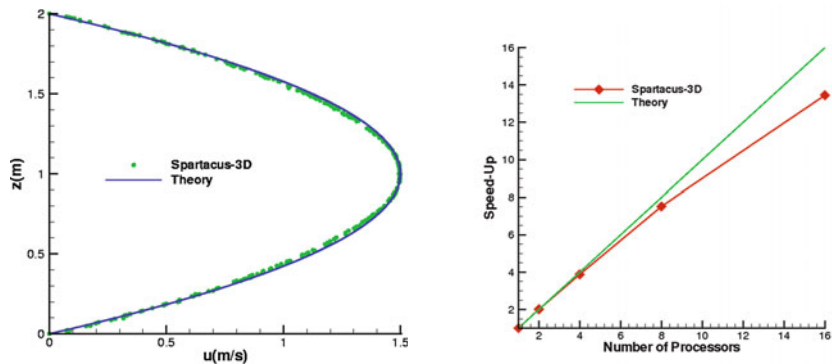


Fig. 1. Left: Poiseuille flow at $Re = 100$. Comparison between SPH and the analytical solution. Right: Speed-up of the previous parallel version of the code.

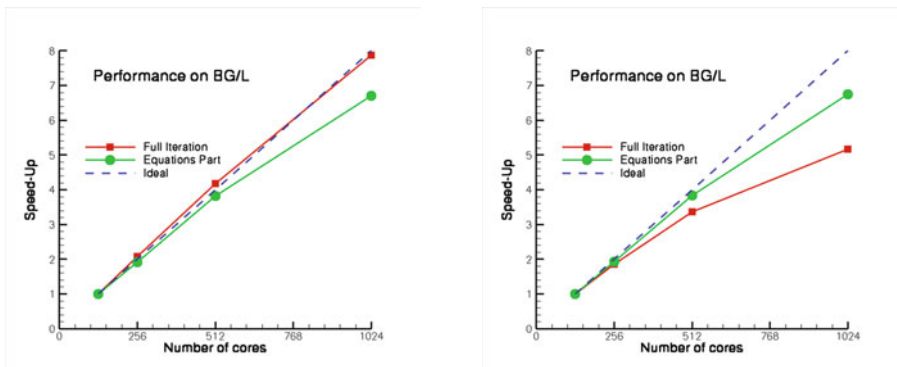


Fig. 2. Speed-up of the code on BGL without optimisation in the sorting of the links (Left) and with optimisation there (Right).

Another important factor of the previous test case of 335,000 particles is that it cannot be run with the former version of the code on the Blue Gene's (BG's) because of insufficient memory per processor. However, the new version, based on using local communications and a smaller amount of memory per processor, can be ran on the BG's and even more particles can be considered. In this work, the case of a 2M particle channel is simulated.

Table 1 shows the CPU time per iteration in 4 cases. Two versions of the codes are considered, the former without optimisation of the link re-indexing (denoted as P for portable) and the latter with this optimisation (denoted as NPO for non-portable, as the IBM ESSL library is used for that purpose). Figures 3 and 4 show the speed-up of the code from 64 to 1,024 processors (CO mode for

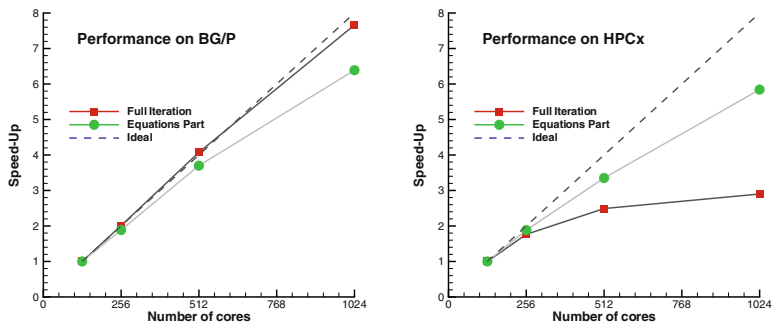


Fig. 3. Speed-up of the code on BGP without optimisation in the sorting of the links (Left) and on HPCx with optimisation there (Right).

	BGP (P)	BGL (P)	BGL (NPO)	HPCx (NPO)
CPU Time/Iteration	1.2894s	1.6919s	0.9689s	0.5732s

Table 1. CPU Time/Iteration on 1,024 processors on various machine, with a portable (P) version and a non-portable (NPO) one.

BG/L and SMP mode for BG/P). The scaling is generally good on both BG’s, with or without optimisation of the link sorting. On the other hand, if the calculation part scales well on HPCx, almost no gain is observed from 512 to 1,204 processors. It might be necessary to increase the number of particles, and also to localise the last global array to increase performance.

Note that the code reaches 6.3% of peak of HPCx system (IBM AIX), which is in the range for CFD codes (the peak computational speed of the HPCx system is 15.3 Tflops and the test has been carried out on 96 processors).

6 Concluding remarks

This paper showed that the new version of Spartacus-3D for HPC can efficiently handle very large simulations. This is due to the localisation of most of the arrays, and the use of local communications instead of global ones, wherever possible. The performance for such an amount of particles is good in terms of calculation time per iteration, but relatively poor for the total calculation time per iteration. The reason is that the particle search is managed through the construction of a global array for the particle list. As a result, some global communications are still required, which impacts on the performance. This is an area we aim to look at in the future.

- [1] R. Issa, Numerical assessment of the Smoothed Particle Hydrodynamics gridless method for incompressible flows and its extension to turbulent flows, PhD thesis, UMIST, (2004).
- [2] J. J. Monaghan, Particle Methods for Hydrodynamics. *Comput. Phys. Rep.*, 3 (1985) 71.
- [3] J.J. Monaghan, Smoothed Particle Hydrodynamics, *Ann. Rev. of Astronomy and Astrophysics*. 30 (1992) 543.
- [4] J. P. Morris and P. J. Fox and Y. Zhu, Modelling low Reynolds Number Incompressible Flows Using SPH, *J. of Comp. Phys.*, 136 (1997) 214.
- [5] C. Moulinec, R. Issa, J.C. Marongiu and D. Violeau, Parallel 3-D SPH Simulations. *Comp. Mod. in Engineering and Science* 25 (2008), 133.
- [6] <http://www.hpcx.ac.uk>.

Parallel Computational Fluid Dynamics 2008

Parallel Numerical Methods, Software Development and
Applications

Tromeur-Dervout, D.; Brenner, G.; Emerson, D.R.; Erhel,
J. (Eds.)

2010, XIV, 438 p. 233 illus., Hardcover

ISBN: 978-3-642-14437-0