A Particle Method for Fluid-Structure Interaction Simulations in Multiple GPUs

Julián Becerra-Sagredo[1(✉)], Leonardo Sigalotti[2], and Jaime Klapp[1,3]

[1] ABACUS-Laboratorio de Matemática Aplicada y Cómputo de Alto Rendimiento, Departamento de Matemáticas, Centro de Investigación y de Estudios Avanzados CINVESTAV-IPN, Carretera México-Toluca Km 38.5, La Marquesa, 52740 Ocoyoacac, Estado de México, Mexico
juliansagredo@gmail.com

[2] Departamento de Ciencias Básicas, Universidad Autónoma Metropolitana Campus Azcapotzalco, Avenida San Pablo Xalpa 180, Azcapotzalco, Reynosa Tamaulipas, 02200 D.F., Mexico, Mexico

[3] Departamento de Física, Instituto Nacional de Investigaciones Nucleares, La Marquesa Ocoyoacac s/n, 52740 Ocoyoacac, Estado de México, Mexico

Abstract. This chapter is a presentation of the programming philosophy behind a novel numerical particle method for the simulation of the interaction of compressible fluids and elastic structures, specifically designed to run in multiple Graphics Processing Units (GPUs). The code has been developed using the CUDA C Application Programming Interface (API) for fine-grain parallelism in the GPUs and the Message Passing Interface library (MPI) for the distribution of threads in the Central Processing Units (CPUs) and the communication of shared data between GPUs. The numerical algorithm does not use smoothing kernels nor weighting functions for the computation of differential operators. A novel approach is used to compute gradients using averages of radial finite differences and divergences using Gauss' theorem by approximations based on area integrals around local spheres around each particle. The interactions of the particles inside the fluid are modelled using the isothermal, compressible Navier-Stokes equations and a simple equation of state. The elastic material is modelled using inter-particle springs with damping. Results show the potential of the method for the simulation of flows in complex geometries.

1 Introduction

The simulation of fluids interacting with elastic structures has a broad number of applications in engineering, medicine and architecture. Aerodynamic design, thermodynamic cycles in motors, containment of fluids and blood flow, to name a few, can be described using the compressible Navier-Stokes equations for the fluid dynamics, combined with linear elasticity theory for the mechanics of solid boundaries. The computational tools designed for the numerical integration of the governing equations have therefore been the focus of a large research effort.

C.J. Barrios Hernández et al. (Eds.): CARLA 2016, CCIS 697, pp. 346–358, 2017.
DOI: 10.1007/978-3-319-57972-6_26

The most general problem requires the handling of complex dynamic geometries, the two way coupling of the forces in the fluid and the solid, and often, the resolution or approximation of turbulent flows.

In aerodynamic design, early works were focused on the use of finite differences or finite volume formulations using curvilinear grids [1–4]. This description can be useful for static boundaries but unpractical for moving or elastic ones. Mathematically, the grid orientation is an important factor to reduce numerical errors due to the lack of multidimensionality of the dimensional splitting technique, producing mesh-dependent solutions. Another partially successful technique is the use of vortex methods [5–7]. These were successful in describing the flow around boundaries for simple grids in Cartesian, cylindrical and spherical coordinates, for incompressible flows reaching Reynolds numbers of several thousands, and providing insight into turbulent flow control with actuators [8]. Nevertheless, the description of more general boundaries was challenged by the need of accurate interpolations near the complex interphase.

A promising technique to deal with complex geometries is the finite element method [9–12]. This technique has been the focus of many works, specially in blood flow simulations [13]. A clear advantage of this technique is the use of triangular or tetrahedral elements able to handle any boundary's geometric complexity. The mesh can be adaptive and able to follow moving boundaries. The elasticity of solid boundaries can be coupled to the fluid pressure for compressible or incompressible flows. Many libraries are available for the simulation of fluid-structure interactions using finite elements [16,17]. But one must keep in mind that the generation of the grids is not trivial and can be quite time consuming. Also, the equations are expressed in weak form and solved implicitly in time, what generates a large non-symmetrical system of linear equations to be solved using GMRES [14], a solver that is difficult to implement with fine grain parallelism [15].

Another technique is the use of Smoothed Particle Hydrodynamics (SPH) [18–22]. In this technique, the equations are discretized following the fluid element's trajectories and derivatives are approximated using the superposition of interpolation kernels for every particle. Some of the advantages of this technique are that there is no need to generate grids and that it naturally follows moving boundaries. Given the purely Lagrangian formulation, the fluid moves with the boundary velocity right at the interphase. If the domains are compressed or even the volume vanishes in some regions, like in the case of flow in pistons, the particles are able to leave the domain entirely and fill newly open spaces. The streamlines can be sketched using the particles' trajectories and simple equations of state can be used to simulate quasi-incompressible flows.

Hybrid techniques have been developed, combining both finite elements and particle trajectories, using a finite element grid that is advected at the boundary [23]. In fact, there is no alternative to this description because necessarily the mesh must follow the boundary. In this case, the generation of the mesh is a problem to be considered. A full Lagrangian description is out of the question for the most general case given that in the presence of vorticity, the grids get highly deformed producing a badly conditioned mass matrix.

During the last decade, the graphics processing unit (GPU) has surpassed the performance of the central processing unit (CPU) in floating point operations per second and local memory transfer velocity [24,25], and the trend is going to continue. The GPUs double their performance approximately every year while the CPUs do it every two years [26]. Therefore, the hardware has become an important factor in the design of fluid flow solvers. They must be designed or adapted to run entirely inside the GPU, with minimum communications with the CPU. This is because the velocity of memory transfers from CPU to GPU is several orders of magnitude slower than the internal memory transfers of the GPU, which can work entirely with L2-cache shared memory.

Considering all the factors, we have collected new ideas for the simulation of fluid-structure interactions using several GPUs. We are motivated to do it because our group has access to Abacus I, a supercomputer with one hundred Tesla K40 GPUs, providing 1200 GB of RAM and 288 000 cores of 745 MHz. In this chapter we describe the programming philosophy and the algorithms necessary for the implementation of a novel particle method running entirely inside a set of GPUs, associated to one another through a corresponding set of CPU threads using the Message Passing Interface library (MPI), and communicating by data transfers between the corresponding CPUs using MPI non-blocking send and receive functions. The particle method is entirely Lagrangian and different from an SPH because it does not use a smoothing particle kernel nor radial weighting functions. It has some similarities to the Moving Particle Semi-implicit method (MPS) [27], but in general, the approximations and search of neighbors are completely new. Its main characteristic is that it computes derivatives using averaged radial finite differences using approximations of area integrals on adaptive spheres around each particle, complemented with the divergence theorem when necessary. It is explicit in time, not requiring the inversion of a linear system of equations. It considers compressible fluids, closing the system using an equation of state for the pressure. The elastic boundary is simulated using another set of particles kept together with forces approximated by springs and damps. The two sets of particles are coupled using the forces of the fluid pressure and the velocity of the elastic surface. The results show the potential of the method to handle complex geometries.

2 Fluid Particles

Consider a Newtonian, compressible fluid with density ρ, pressure P and viscosity μ, kept at a constant temperature T, for simplicity but without loss of generality, which motion is described by the velocity vector $\boldsymbol{v}$. The material derivative $D/Dt = \partial/\partial t + \boldsymbol{v} \cdot \nabla$ is the chain rule for the total derivative in time, describing the variations along the trajectories of the fluid elements given by $d\boldsymbol{x}/dt = \boldsymbol{v}$.

The system of equations describing its dynamics are given by the conservation of mass

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot \boldsymbol{v}, \tag{1}$$

and Newton's second law

$$\rho \frac{D\boldsymbol{v}}{Dt} = -\nabla P + \mu \nabla^2 \boldsymbol{v}, \tag{2}$$

closed by an equation of state relating the density and the pressure $P = P(\rho, T)$. We consider moving solid boundaries where the fluid velocity is set to the solid velocity $\boldsymbol{v} = \boldsymbol{v}_s$. The equation of state

$$P = \frac{\rho}{3} \tag{3}$$

is used for simplicity during this work.

3 Solid Particles

The solid is represented by particles which conserve mass, subject to linearly elastic, spring like forces and damping. The conservation of mass in the solid is given by

$$\frac{D\rho_s}{Dt} = -\rho_s \nabla \cdot \boldsymbol{v}_s. \tag{4}$$

Newton's second law for the forces in the solid is given by

$$\rho_s \frac{D\boldsymbol{v}_s}{Dt} = \boldsymbol{f}_s, \tag{5}$$

where $\boldsymbol{f}_s$ is the sum of the elastic, friction and pressure forces, such that $\boldsymbol{f}_s = \boldsymbol{f}_e + \boldsymbol{f}_f + \boldsymbol{f}_p$.

4 Discretization

The discretization of the system could be done by seeding particles inside the fluid domain as desired, keeping them at a minimum distance between each other, denoted by h_{min}. We have chosen to seed the boundary with particles, either by a given boundary mesh, necessary for complex geometries, or using a level set function. If the boundary is provided by a given mesh, the location of the particles must be complemented with the surface's normal vector pointing towards the fluid domain for every boundary particle. These normal vectors are going to be used to determine the interior and exterior of the computational domain by simple weighted dot products with the relative position vector from the surface. If the boundary is given by a level set function, the boundary is seeded with particles and the level set is used to know if any position in space is part of the computational domain. We choose to seed the particles inside the fluid domain using a virtual Cartesian mesh covering the desired volume to mesh, eliminating the points that lie outside. Each location of the virtual Cartesian mesh is tested to be inside of the computational domain, and if so, stored. Figure 1 shows a cylinder seeded with particles using a Cartesian array

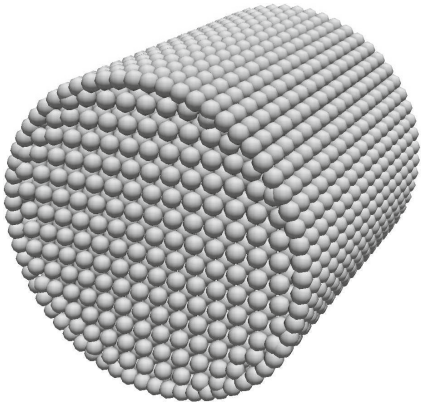

Fig. 1. Initial particle grid for a cylindrical domain. The particles are seeded using a Cartesian array and adjusted near the boundary to keep a minimum distance between particles.

with spacing h and keeping particles at a minimum distance $0.5\,h$ next to the boundary particles.

Once the initial particle positions are given, the fields are initialized for every particle, giving the densities ρ and ρ_s, and the velocities $\boldsymbol{v}$ and $\boldsymbol{v}_s$, at time $t = 0$.

The particle trajectories are solved with second order accuracy in time by setting

$$\boldsymbol{x}_i^{n+1} = \boldsymbol{x}_i^n + \frac{1}{2}(\boldsymbol{v}_i^n + \boldsymbol{v}_i^{n+1})\Delta t, \tag{6}$$

where the time is discretized, such that the super-index n denotes the time step $t^n = n\Delta t$ for $n = 0, 1, ..., N$. This formula provides a second order, explicit integration of the trajectories and is sketched in Fig. 2.

We choose to use a purely Lagrangian formulation where particles are never re-meshed and interpolations are not necessary for advection. This warranties that the transport is exact along the trajectories of the particles with small errors due to the trajectory integration. Additionally, we leave the mollifier kernel concept used in SPH and focus only in computing the derivatives on the right hand side of the equations for the conservation of mass and Newton's second law. All the derivatives are computed using averages of radial finite differences, considering a regular distribution of particles. The approximations will reduce accuracy when the particle field deforms and the corrections to this approximations are the focus of future research. Only first line of sight nearest neighbors are considered in every 26 directions in three dimensions and 8 in two dimensions. The gradient is computed using a vector average of radial finite differences

$$\nabla P_i \approx \frac{D}{N} \sum_{j \neq i} \frac{P_j - P_i}{||\boldsymbol{x}_j - \boldsymbol{x}_i||} \widehat{(\boldsymbol{x}_j - \boldsymbol{x}_i)}, \tag{7}$$

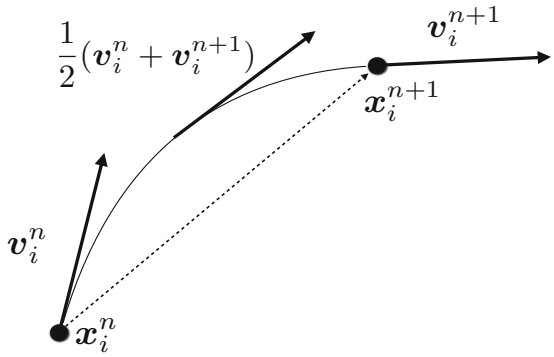

Fig. 2. Second order scheme for the integration of trajectories. The velocity is advanced using the momentum equation and the time-averaged velocity is used to advance the position of the particles.

where D is the number of dimensions and N is the number of nearest neighbor particles. The divergence is computed using Gauss' theorem by

$$\nabla \cdot \boldsymbol{v}_i \approx \frac{A}{V} \sum_{j \neq i} \frac{(\boldsymbol{v}_j + \boldsymbol{v}_i)}{2} \cdot \frac{(\boldsymbol{x}_j - \boldsymbol{x}_i)}{||\boldsymbol{x}_j - \boldsymbol{x}_i||}, \tag{8}$$

where $V = (4/3)\pi R^3$ and $A = 4\pi R^2/N$, are the volume and area of a reference sphere for the calculation of the divergence. The radius R is considered as the averaged half distance to the nearest neighbors. Finally, the Laplacian is the combination of both concepts,

$$\nabla^2 v_i \approx \frac{A}{V} \sum_{j \neq i} \frac{(v_j - v_i)}{||\boldsymbol{x}_j - \boldsymbol{x}_i||}. \tag{9}$$

The model for the friction force consists of a damping factor $\mathcal{D}$ times the square of the velocity magnitude,

$$\boldsymbol{f}_f = -\mathcal{D} v_s^2 \hat{\boldsymbol{v}}. \tag{10}$$

The elastic forces are modelled using springs to the nearest neighbors

$$\boldsymbol{f}_{e,i} = -\sum_{j \neq i} k(||\boldsymbol{x}_j - \boldsymbol{x}_i|| - L_{i,j})\widehat{(\boldsymbol{x}_j - \boldsymbol{x}_i)}, \tag{11}$$

where $L_{i,j}$ is the equilibrium length for the spring bonding particles i and j.

The conservation of mass and Newton's second law are advanced as ordinary differential equations over the trajectory of each fluid particle, explicitly in time with a forward Euler method, after evaluating the spatial operators on the right hand side.

Particle methods do not have conservation issues due to advection. Only the inaccurate computation of the sources of compressibility and force can partially affect conservation of the advected quantities. The accurate calculation of

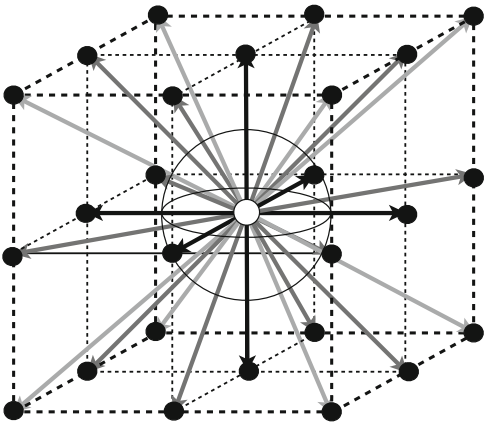

Fig. 3. Scheme for the radial finite differences approximation of the gradient (7), the divergence (8) and the Laplacian (9). The radial finite difference approximations are based on sums of individual difference vectors to the nearest neighbor particles combined with the divergence theorem using a sphere with radius of the average half distance to the neighbors.

gradients and divergences can be affected by a highly deformed particle field. Viscous flows have fluid elements that deform smoothly. Anyway, the computation of individual area weights for each neighbor particle is necessary for accurate approximations of the differential operators. Its detailed analysis and implementation are subjects of future research.

5 Data Structures

The data structure is a list of particles for each CPU process, where the solid particles, which describe also the fluid particles at the boundary, are given at the beginning of the list, followed by the particles in the bulk of the fluid.

A list of nearest neighbors is constructed in order to compute the differential operators in the right hand side of the transport equations for the solid and the fluid. Analogously to Lattice-Boltzmann algorithms, we consider 8 neighboring particles in two dimensions and 26 in three. An initial list is constructed or given. The list is updated after a fixed number of time steps during the numerical integration. The new list is produced following the hypothesis that for every particle, every new neighbor was in the neighbor list of its former neighbors. Algorithm 1 is the pseudo-code for the updating of the list of nearest neighbors $neighborlist(1:n, 1:26)$, where n is the total number of particles in the computational domain. The distance between particles is given by the function $distance(i, j)$ (line 3 and 11). If any particle lies outside a neighboring radius h, its index is tagged to be replaced. The list is double-checked to avoid the repetition of particles (line 14).

```
1: oldneighborlist(i,1:26) = neighborlist(i,1:26)
2: do j from 1 to 26
3: k = oldneighborlist(i,j)
4:    dr = distance(i,k)
5:    if (dr > h) neighborlist(i,k) = -1
6: do j from 1 to 26
7: k = oldneighborlist(i,j)
7:    if (neighborlist(i,k) = -1)
8:       mindr = 10 h
9:       do l from 1 to 26
10:         m = oldneighborlist(i,l)
11:         do o from 1 to 26
12:           p = oldneighborlist(l,o)
13:           dr = distance(i,p)
14:           if (dr < mindr)
15:             inthelist = false
16:             do q from 1 to 26
17:               if (p = neighborlist(i,q)) inthelist = true
18:             if (inthelist = false)
19:               neighborlist(i,j) = p
20:               mindr = dr
```

Algorithm 1. Update list of nearest neighbors.

6 The Programming Model

We use the template code presented in [28] but adapted for a list of particles. The computational domain is geometrically decomposed in a one-dimensional array of M sub-domains. The Message Passing Interface (MPI) library is used to start M threads for the same number of CPU cores. Every CPU thread corresponds to a process to be run in a different GPU. Frequently, each node of the cluster will have one or two GPUs, therefore it is necessary to distribute the M threads in different nodes, such that every process is able to pick at least one exclusive GPU.

Communications between GPUs is achieved loading the necessary GPU data to the local CPU memory, communicating the CPU threads using MPI unblocked but synchronized sends and receives, and loading it back to the GPU. In this implementation, only boundary data is communicated and particles are not transferred between processes. Future versions may contain the transfer of particles between GPUs.

Inside the GPU, operations are threaded over the list of particles, as described in Fig. 4. The list is distributed in a three-dimensional array with power of two dimensions, further subdivided in blocks to be given to the GPU cores. The list of threads will be in general larger than the list of particles. Those threads that do not correspond to a particle perform no work.

The programming model for a single GPU is focused in performing parallel L2 memory reads and a few global memory writes. The *nvcc* compiler is capable

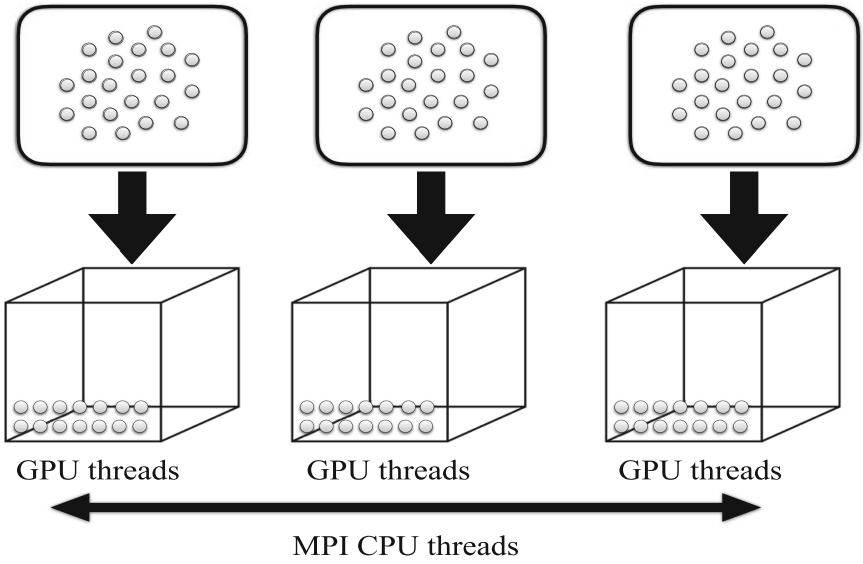

Fig. 4. Scheme of the computational decomposition of the list of particles and its arrangement as GPU threads in a cube. The GPUs are mastered by CPU threads using MPI.

of automatically allocating L2 memory reads if we provide read-only arrays. All numerical operations are done using the registers memory and the results are written back to write-only arrays in global memory.

7 Results

The algorithm is a newly proposed numerical method and the tests are focused in proving its correctness without going into deep analysis of order of convergence. The algorithm has been theoretically designed to be second order for a regular distribution of neighboring particles.

We use an initial Gaussian perturbation in the density

$$\rho(r) = 1.0 + 0.01 \exp^{-r^2/5}, \tag{12}$$

where r is the distance from the center of the domain with dimensions $[15, 15, 15]$, in a quiet Newtonian fluid with viscosity $\mu = 1$.

First we prove that our new radial difference formulas are correct by computing the norm of the pressure gradient and the divergence of the velocity field after a single time step $\Delta t = 0.02$. Figure 5 shows the comparison of the differential operators for a cylindrical domain filled with 250000 particles and a Cartesian 64^3 mesh with finite differences. It shows agreement and even a slight improvement in the case of the divergence.

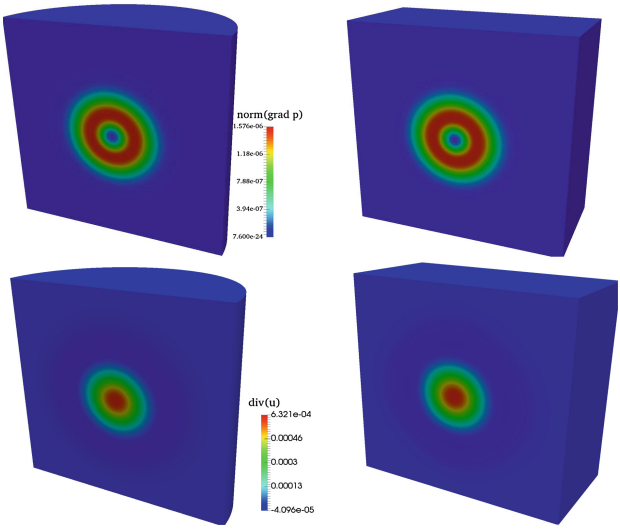

Fig. 5. Comparison of the square of the norm of the pressure gradient (top) and the divergence of the velocity (bottom), for the radial differences scheme using particles in a cylindrical domain (left) and finite differences in a cube (right). Both domains are shown cut in half by a plane normal to the x-axis.

We simulate the acoustic wave resulting from the Gaussian initial condition in a cylinder filled with 250000 particles. We compare it with a second order semi-Lagrangian scheme [29] in a 64^3 cube. Figure 6 shows agreement between the schemes even though the reflection of the waves is different for the square domain and the cylinder. Therefore, only early stages of the wave are compared.

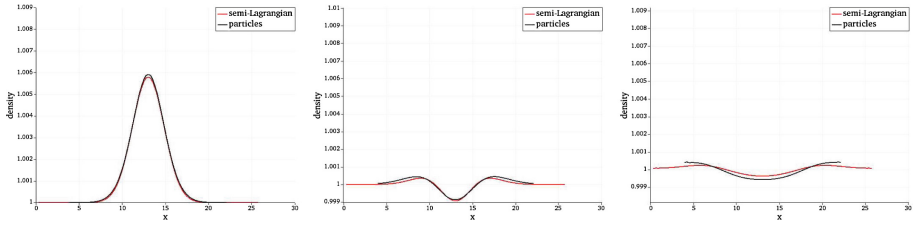

Fig. 6. Comparison of the acoustic wave for the newly proposed particle method with 250000 particles and a semi-Lagrangian scheme [29] for a 64^3 domain. From left to right we can see the density along the y-axis for times t = 0.2, t = 0.6 and t = 1.2.

We have run the code for one, two and four GPUs Tesla C2070. The results for a small run consisting of a thousand time steps are presented in Fig. 7. The runs show strong scalability in the vertical direction and weak scalability in the

horizontal direction. The weak scalability shows a small penalty due to the communication between GPUs. The objective of running in many GPUs is not the acceleration of the code, although it is possible to observe significant acceleration in the case of two GPUs compared to one. Nevertheless, we note that for the case of four GPUs, the acceleration is much less and extrapolating we can see that many more GPUs would not achieve significant acceleration. The point is that runs in several GPUs must be focused in the simulation of very large problems or with high resolutions. The GPUs are very fast processing units that should be exploited at maximum with the lowest number of communications possible. The use of several GPUs must be evaluated using the weak scalability concept where larger problems are run in the approximately same computational time, shown in Fig. 7 in the horizontal direction.

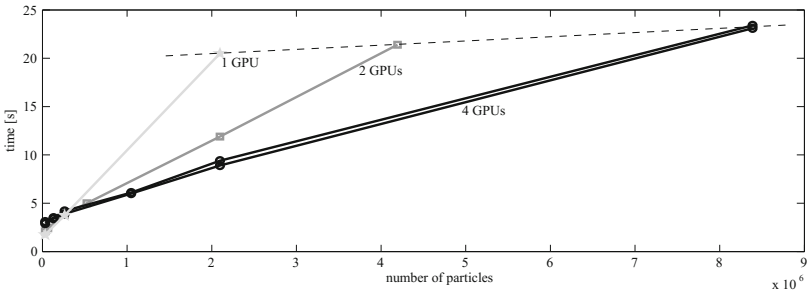

Fig. 7. Weak (horizontal) and strong (vertical) scalability of the domain decomposition scheme for the list of particles using MPI communication between GPUs. The two black curves correspond to one- and two-dimensional domain decomposition.

8 Conclusion

We presented a programming model using domain decomposition with message passing for computations using multiple GPUs adapted to a list of particles. This particle template code has been filled with a novel numerical method. The numerical method consists of a particle method for the integration of transport equations along material trajectories of fluid elements. The fluid elements are defined by the location of the particles. The equations are solved in time using a second order mid-point integration scheme for the positions of the particles and a first order explicit Euler integration for the velocity and density. The pressure is obtained explicitly using an equation of state. Averages of radial derivatives combined with the divergence theorem are used for the approximation of the spatial differential operators at the right hand side of the transport equations. These differences have been used in different forms in other works but never in the form presented here. We take the nearest neighbors in the 26 principal Cartesian directions like in a Lattice-Boltzmann scheme. The approximations have been

tested to be second order accurate for a regular, Cartesian distribution of particles, and are expected to loose accuracy as the particle field distorts. Moment conservation for the area covered by the neighbors around each particle will be explored to keep the accuracy regular for any configuration of the neighbors. The list of particles is complemented with a list of nearest neighbors, updated after a fixed number of time steps with a local search scheme. The results of the acoustic waves inside a cylinder meshed with a Cartesian array of particles show the potential of the code to solve problems in complex geometries without the need of complex mesh generators. For complex geometries, the boundary node positions and normal vectors pointing to the fluid must be provided.

Acknowledgements. This work was partially supported by ABACUS, CONACyT grant EDOMEX-2011-C01-165873. The calculations for this work have been performed in the Abacus I supercomputer.

References

1. Steger, J.L.: On application of body conforming curvilinear grids for finite difference solution of external flow. Appl. Math. Comput. **10–11**, 295–316 (1982)
2. Karki, K.C., Patankar, S.V.: Calculation procedure for viscous incompressible flows in complex geometries. Numer. Heat Transfer **14**(3), 295–307 (1988)
3. LeVeque, R.J.: Finite Volume Methods for Hyperbolic Problems. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge (2002)
4. Bijl, H., Wesseling, P.: A unified method for computing incompressible and compressible flows in boundary-fitted coordinates. J. Comput. Phys. **141**(2), 153–173 (1998)
5. Leonard, A.: Vortex methods for flow simulation. J. Comput. Phys. **37**(3), 289–335 (1980)
6. Cottet, G.H., Koumoutsakos, P.D.: Vortex Methods: Theory and Practice. Cambridge University Press, Cambridge (2000)
7. Ploumhans, P., Winckelmans, G.S., Salmon, J.K., Leonard, A., Warren, M.S.: Vortex methods for direct numerical simulation of three-dimensional Bluff body flows: application to the sphere at Re = 300, 500 and 1000. J. Comput. Phys. **178**(2), 427–463 (2002)
8. Koumoutsakos, P.: Active control of vortex-wall interactions. Phys. Fluids **9**(12), 3808–3816 (1997)
9. Strang, G., Fix, G.: An Analysis of the Finite Element Method. SIAM, Wesley-Cambridge Press, Philadelphia (1973)
10. Kuzmin, D., Hämäläinen, J.: Finite Element Methods for Computational Fluid Dynamics: A Practical Guide. Computational Science & Engineering. SIAM, Philadelphia (2014)
11. Löner, R., Morgan, K., Peraire, J., Zienkiewicz, O.C.: The free-lagrange method. In: Fritts, M.J., Crowley, W.P., Trease, H. (eds.) Recent developments in FEM-CFD. Lecture Notes in Physics, pp. 236–254. Springer, Heidelberg (2005)
12. Schweitzer, M.A.: Generalizations of the finite element method. Cent. Eur. J. Math. **10**(1), 3–24 (2012)
13. Taylor, C.A., Hughes, T.J.R., Zarins, C.K.: Finite element modeling of blood flow in arteries. Comput. Methods Appl. Mech. Eng. **158**(1), 155–196 (1998)

14. Saad, Y., Schultz, M.H.: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **7**, 856–869 (1986)
15. Ziane Khodja, L., Couturier, R., Glersch, A., Bahi, J.M.: Parallel sparse linear solver with GMRES method using minimization techniques of communications for GPU clusters. J. Supercomput. **69**(1), 200–224 (2014)
16. Whiting, C.H., Jansen, K.E.: A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis. Int. J. Numer. Methods Fluids **35**(1), 93–116 (2001)
17. Quarteroni, A.: Numerical Models for Differential Problems. Springer, Heidelberg (2009)
18. Monaghan, J.J.: Smoothed particle hydrodynamics. Annu. Rev. Astrophys. **30**, 543–574 (1992)
19. Monaghan, J.J.: Smoothed particle hydrodynamics. Rep. Prog. Phys. **68**(8), 1703–1760 (2005)
20. Antoci, C., Gallati, M., Sibilla, S.: Numerical simulation of fluid-structure interaction by SPH. Comput. Struct. **85**(11), 879–890 (2007)
21. Sigalotti, L.D.G., Klapp, J., Rendon, O., Vargas, C.A., Peña-Polo, F.: On the kernel and particle consistency in smoothed particle hydrodynamics. J. Appl. Numer. Math. **108**, 242–255 (2016)
22. Sigalotti, L.D.G., Rendon, O., Klapp, J., Vargas, C.A., y Campos, K.: A new insight into the consistency of Smoothed Particle Hydrodynamics. arXiv:1644200 [physics.com-ph] 21 August 2016
23. Donea, J., Huerta, A.: Finite Element Flow Problems. Wiley, Hoboken (2003)
24. Nickolls, J., Dally, W.J.: The GPU computing era. IEEE Micro **30**(2), 56–69 (2010)
25. Keckler, S.W., Dally, W.J., Khailany, B., Garland, M., Glasco, D.: GPUs and the future of parallel computing. IEEE Micro **31**(5), 7–17 (2011)
26. NVIDIA CUDA C Programming Guide, version 7.5, Nvidia (2015)
27. Koshizuka, S., Oka, Y.: Moving particle semi-implicit method for fragmentation of incompressible fluid. Nucl. Sci. Eng. **123**, 421–434 (1996)
28. Becerra-Sagredo, J., Mandujano, F., Málaga, C., Klapp, J., Teresa, I.: A template for scalable continuum dynamic simulations in multiple GPUs. In: Gitler, I., Klapp, J. (eds.) ISUM 2015. CCIS, vol. 595, pp. 473–484. Springer, Cham (2016). doi:10.1007/978-3-319-32243-8_33
29. Becerra-Sagredo, J.T., Málaga, C., Mandujano, F.: Moments preserving and high-resolution semi-Lagrangian advection scheme. SIAM J. Sci. Comput. **38**(4), A2141–A2161 (2016)