Contents

Part I

Application

Introduction

The first part of this thesis presented the tools needed to understand and target performance in HPC. The second part exposed our metric showing the benefit of accelerators, in this case GPUs, over classical GPU in two contexts: irregular computation and irregular communication/memory behaviors. We showed that the accelerators gave a real advantage on those two problems and allows us to push the limits.

In order to extend this study and our metric we decided to target another irregular behavior problem embedding both computation/communication/memory wall behavior. In order to show how accelerators handle real world problems, we searched for an application fulfilling our needs. Our choose fell on the Smoothed Particle Hydrodynamics problem applied to fluid and astrophysics. The first version of this application was developed during a 2 months internship at the Los Alamos National Laboratory from the FleCSI framework.

In this part we first present the Smoothed Particle Hydrodynamics method from a physical point of view and making a parallel with the computer science problems involved. Indeed, a huge amount of time have been spend on the understanding of the physics side to be able to do realistic simulation and thus realistic behavior. The second chapter present our application, FleCSPH. Starting from the current FleCSI version, we introduce the algorithm and methods to solve efficiently this problem on classical processor and the acceleration generated adding GPUs.

Chapter 1

Complex system modelization

1.1 Introduction

In this section we give details on our choices for the generic application confronted to both computation and communication walls in irregular context. The problem we choose, the Smoothed Particle Hydrodynamics method implementation, is described on a physical aspect. We point out the difficulties involved in the resolution on supercomputers and more especially accelerators. A lot of work have been done on the comprehension of the physical aspect to produce a code that meet the behavior of real physics problems.

The first section is a presentation of the SPH method itself and the overall limitation we have to face. Then we describe different kind of problems we use as a benchmark of the application itself.

1.2 Physical modelization and downsides

In order to target a complex problem targeting all the walls and confront classical architectures to hybrid ones the problem had to be very specific.

1.2.1 Smoothed Particles Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) is an explicit numerical mesh-free Lagrangian method. It is used to solve hydrodynamical partial differential equations (PDEs) by discretized them into a set of fluid elements called particles. This computational method was invented for the purpose of astrophysics simulations by Monaghan, Gingold and Lucy in 1977 [Luc77, GM77]. This first SPH work conserved mass and they later proposed a method which also conserves linear and angular moment [GM82]. The method was extended for general fluid simulation and many more fields from ballistics to oceanography. The development of new reliable, parallel and distributed tools for this method is a challenge for future HPC architectures with the upcoming Exascale systems.

The method, as illustrated in figure 1.1, computes the evolution of a group of particles representing physical quantities. Those physical quantities are either fixed or computed for every particle regarding its neighbors in the radius of its smoothing length h. The particles in this radius are then valued according to their distance using a smoothing function W, also called a kernel. The fundamental SPH formulation for any physical quantity A is then to compute with all the neighbors of b of a a particle by:

$$A(\vec{r})_a \simeq \sum_b \frac{m_b}{\rho_b} A(\vec{r_b}) W(|\vec{r} - \vec{r_b}|, h) \tag{1.1}$$

On a physics aspect, this method has several advantages: it can handle deformations, low densities, vacuum, and makes particle tracking easier. It also conserves mass, linear and angular
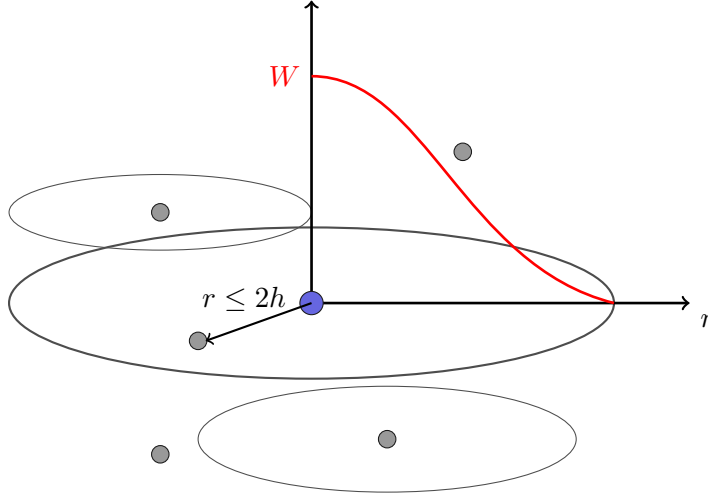
Figure 1.1: SPH kernel W and smoothing length h representation

momenta, and energy by its construction that implies independence of the numerical resolution. Another strong benefit of using SPH is its exact advection of fluid properties. Furthermore, the particle structure of SPH easily combines with tree methods for solving Newtonian gravity through N-body simulations. As a mesh-free method, it avoids the need of grid to calculate the spatial derivatives.

However, there are cons to consider using SPH: it cannot be extend to all PDE formulations; it requires careful setup of initial distribution of particles; further, it can be struggle to resolve turbulence-dominated flows and special care must be taken when handling high gradients such as shocks and surface structure of neutron stars. Many works are leading to handle more cases and to push the limitations of this method [DRZR17, LSR16, RDZR16].

In this work, we are solving Lagrangian conservation equations (Euler equations) for density, energy and momentum of an ideal fluid [LL59] such that:

$$\frac{d\rho}{dt} = -\rho(\nabla \cdot \vec{v}), \quad \frac{du}{dt} = -\frac{P}{\rho}(\nabla \cdot \vec{v}), \quad \frac{d\vec{v}}{dt} = -\frac{1}{\rho}(\nabla P) \tag{1.2}$$

with ρ the density, P the pressure, u the internal energy and v the velocity, where $d/dt = \partial/\partial_t + \vec{v} \cdot \nabla$ which is convective derivative.

By using the volume element $V_b = m_b/\rho_b$, we can formulate the Newtonian SPH scheme [Ros09] such that

$$\rho_a = \sum_b m_b W_{ab}(h_a) \tag{1.3}$$

$$\frac{du_a}{dt} = \frac{P_a}{\rho_a^2} \sum_b m_b \vec{v}_{ab} \cdot \nabla_a W_{ab} \tag{1.4}$$

$$\frac{d\vec{v}_a}{dt} = -\sum_b m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} \right) \nabla_a W_{ab} \tag{1.5}$$

where $W_{ab} = W(|\vec{r}_a - \vec{r}_b|, h)$ is the smoothing kernel. The equations we would like to solve allow for emergence of discontinuities from smooth initial data. At discontinuities, the entropy increases in shocks. That dissipation occurs inside the shock-front. The SPH formulation here is inviscid so we need to handle this dissipation near shocks. There are a number of way to handle this problem, but the most widespread approach is to add artificial viscosity (or artificial dissipation) terms in SPH formulation such that:

$$\left(\frac{du_a}{dt} \right)_{art} = \frac{1}{2} \sum_b m_b \Pi_{ab} \vec{v}_{ab} \cdot \nabla_a W_{ab} \tag{1.6}$$

$$\left(\frac{d\vec{v}_a}{dt}\right)_{art} = -\sum_b m_b \Pi_{ab} \nabla_a W_{ab} \tag{1.7}$$

In general, we can express the equations for internal energy and acceleration with artificial viscosity

$$\frac{du_a}{dt} = \sum_b m_b \left(\frac{P_a}{\rho_a^2} + \frac{\Pi_{ab}}{2}\right) \vec{v}_{ab} \cdot \nabla_a W_{ab} \tag{1.8}$$

$$\frac{d\vec{v}_a}{dt} = -\sum_b m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} + \Pi_{ab}\right) \nabla_a W_{ab} \tag{1.9}$$

Π_{ab} is the artificial viscosity tensor. As long as Π_{ab} is symmetric, the conservation of energy, linear and angular momentum is assured by the form of the equation and antisymmetry of the gradient of kernel with respect to the exchange of indices a and b. Π_{ab} may define in different ways and here we use [MG83] such as:

$$\Pi_{ab} = \begin{cases} \frac{-\alpha \bar{c}_{ab} \mu_{ab} + \beta \mu_{ab}^2}{\bar{\rho}_{ab}} & \text{for } \vec{r}_{ab} \cdot \vec{v}_{ab} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.10}$$

$$\mu_{ab} = \frac{\bar{h}_{ab} \vec{r}_{ab} \cdot \vec{v}_{ab}}{r_{ab}^2 + \epsilon \bar{h}_{ab}^2} \tag{1.11}$$

Using the usual form c_s as $c_s = \sqrt{\frac{\partial p}{\partial \rho}}$. The values of ϵ, α, and β have to be set regarding the problem targeted. As an example we used for the Sod shock tube problem: $\epsilon = 0.01h^2$, $\alpha = 1.0$, and $\beta = 2.0$.

There are many possibilities for the smoothing function, called the kernel. As an example the Monaghan's cubic spline kernel is given by:

$$W(\vec{r_{ij}}, h) = \frac{\sigma}{h^D} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & \text{if} \quad 0 \le q \le 1 \\ \frac{1}{4}(2-q)^3 & \text{if} \quad 1 \le q \le 2 \\ 0 & \text{otherwise} \end{cases} \tag{1.12}$$

where $q = r/h$, r the distance between the two particles, D is the number of dimensions and σ is a normalization constant with the values:

$$\sigma = \begin{cases} \frac{2}{3} & \text{for 1D} \\ \frac{10}{7\pi} & \text{for 2D} \\ \frac{1}{\pi} & \text{for 3D} \end{cases} \tag{1.13}$$

In the computation of forces we also need to apply the gradient of the smoothing kernel. This is, in our example, for the cubic spline kernel:

$$\vec{\nabla}_i W(\vec{r_{ij}}, h) = \frac{\sigma}{h^{D+1}} \times \begin{cases} (-\frac{3}{h} + \frac{9}{4h}q)\vec{r_{ij}}, & \text{si } 0 \le \frac{r}{h} < 1 \\ (\frac{-3}{r} + \frac{3}{h} - \frac{3}{4h}q)\vec{r_{ij}}, & \text{si } 1 \le \frac{r}{h} < 2 \\ 0, & \text{si } \frac{r}{h} \ge 2 \end{cases} \tag{1.14}$$

Figure 1.2 is a representation of the cubic spline kernel with $\sigma = 1$ and $h = 1$. The abscissa axis represent r, distance between the particles and the ordinate the value of the smoothing kernel. When the support of the function, 2 is reached the particles are ignored $W = 0$, represented in red on the figure.

To sum up, the SPH resolution scheme and its routines are presented on algorithm 1. The Equation of State (EOS) and the integration are problem dependent and will be define for each test case in section 1.3.
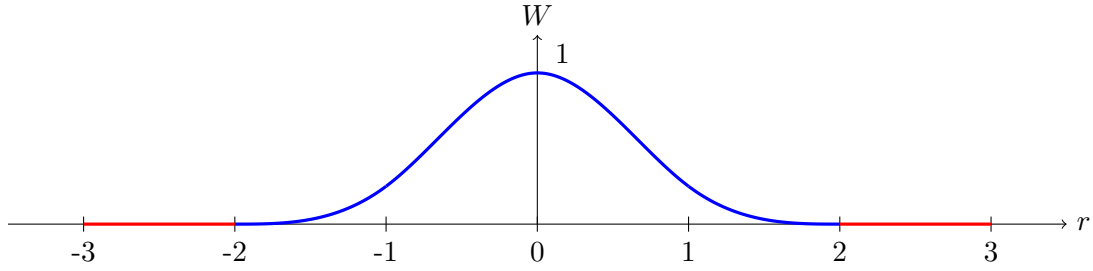
Figure 1.2: Cubic spline kernel example with $\sigma = 1$ and $h = 1$

Algorithm 1 SPH loop algorithm

1: **while** not last step **do**
2:     Compute density for each particle (1.3)
3:     Compute pressure using EOS
4:     Compute acceleration from pressure forces (1.9)
5:     Compute change of internal energy for acceleration (1.8)
6:     Advance particles after integration
7: **end while**

The main downside for the implementation of this method is the requirement for local computation on every particle. The particles have to be grouped locally to perform the computation of (1.3), (1.8) and (1.9). A communication step is needed before and after (1.3) to get the local physical data to be able to compute (1.8) and (1.9). The tree data structure allows us to perform $O(Nlog(N))$ neighbor search but also add a domain decomposition and distribution layer.

1.2.2   Gravitation

For classical problems like fluid flow the gravitation can directly be applied on the particles with the force:

$$\vec{a_g} = m\vec{g} \tag{1.15}$$

In order to consider astrophysics problems we need to introduce self-gravitation and gravitation. Each particle imply an action on the others based on its distance and mass. The equation of gravitation for a particle i with j other particles is:

$$\vec{f_{ai}} = \sum_j -G\frac{m_i m_j}{|\vec{r_i} - \vec{r_j}|^3}\vec{r_{ij}} \tag{1.16}$$

This computation involve an $O(N^2)$ complexity and thus is not applicable directly. We applied the method called Fast Multipole Method, FMM and discussed in [BG97]. This method is perfectly adapted to our tree representation of the domain and particles.

This method aim to compute the gravitation up to an approximation determined by the user. Details are given in figure 1.3, from left bottom to right bottom for a group of particles. We identify three main actors in this method:

Particles: the bodies on which we need to compute the gravitation regarding the other particles.   In figure 1.3 we separate the green particles, on which we are computing the gravitation, from the other, blue, particles.

Multipoles: the center of mass for a group of particles.   In our example they regroup the mass and barycenter of the sub-particles. There are several level of multipoles: particles' multipole and multipoles' multipole.

Locals: the center of mass for the reduction on the particles concerned in this walk. They have the same behavior as the multipoles but the information goes down to particles instead of
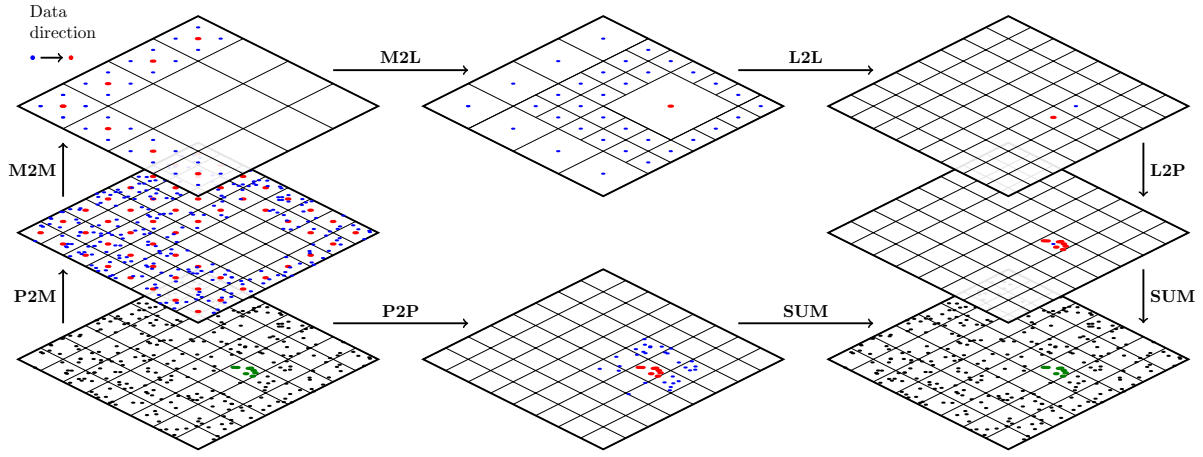
Figure 1.3: Fast Multipole Method schematics. Particles to Multipole (P2M), Multipole to Multipole (M2M), Multipole to Particles (M2P), Multipole to Local (M2L), Local to Local (L2L) and Particles to Particles (P2P). Schematic inspired from [YB11]

up.

In order to compute the gravitation for a group of particles in the domain the steps are:

Particles to Particles (P2P): For the particles that are close, use the direct $O(N^2)$ algorithm. This is the part that grow if the user desires more accurate results.

Particles to Multipoles (P2M): Gather the data of all the sub-particles to the centers of mass, the multipoles. This is the first layer of the tree, the leaves.

Multipoles to Multipole (M2M): Gather the data of multipoles on higher level of the tree from the leaves to the root.

Multipoles to Local (M2L): Compute the gravitation part of all the distant multipole to the local.

Local to Local (L2L): Go down in the tree and spread the component to sub-locals.

Local to Particles (L2P): When a leaf of the tree is reached, compute the application of the local for all sub-particles.

Summation: At the end of the computation for both P2P and L2P the two interactions can be summed up to compute the gravitation applied to the particles.

This scheme have to be repeat for every group of particles. The P2M-M2M steps are done just once before the FMM method for all the groups of particles. For the choice between either P2P or M2L we use a criterion call MAC, Multipole Acceptance Criterion. In this study we used an angle between the local center of mass and the edge of distant multipole. If the angle fits the criterion we use the current multipole, otherwise we go lower in the tree to consider smaller multipole. If the criterion never match, we are too close and consider P2P.

For the P2P step, the classical gravitation computation is used, like presented in equation 1.16. But for the interaction with distant multipoles, we use a Taylor series.
The gravitation function of equation 1.16 can be approximate on a particle at position $\vec{r}$ by the gravitation computed at the centroid at position $\vec{r_c}$:

$$\vec{f}(\vec{r}) = \vec{f}(\vec{r_c}) + ||\frac{\partial \vec{f}}{\partial \vec{r}}|| \cdot (\vec{r} - \vec{r_c}) + \frac{1}{2}(\vec{r} - \vec{r_c})^\intercal \cdot ||\frac{\partial \vec{f}}{\partial \vec{r} \partial \vec{r}}|| \cdot (\vec{r} - \vec{r_c}) \qquad (1.17)$$

From equation 1.16 we compute the term $||\frac{\partial \vec{f}}{\partial \vec{r}}||$:

$$\frac{\partial \vec{f}}{\partial \vec{r}} = -\sum_p \frac{m_p}{|\vec{r_c} - \vec{r_p}|^3} \begin{bmatrix} 1 - \frac{3(x_c-x_p)(x_c-x_p)}{|\vec{r_c}-\vec{r_p}|^2} & -\frac{3(y_c-y_p)(x_c-x_p)}{|\vec{r_c}-\vec{r_p}|^2} & -\frac{3(z_c-z_p)(x_c-x_p)}{|\vec{r_c}-\vec{r_p}|^2} \\ -\frac{3(x_c-x_p)(y_c-y_p)}{|\vec{r_c}-\vec{r_p}|^2} & 1 - \frac{3(y_c-y_p)(y_c-y_p)}{|\vec{r_c}-\vec{r_p}|^2} & -\frac{3(z_c-z_p)(y_c-y_p)}{|\vec{r_c}-\vec{r_p}|^2} \\ -\frac{3(x_c-x_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} & -\frac{3(y_c-y_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} & 1 - \frac{3(z_c-z_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} \end{bmatrix} \quad (1.18)$$

And we propose a compact version of the matrix with:

$$||\frac{\partial f^a}{\partial r^b}|| = -\sum_c \frac{m_c}{|\vec{r} - \vec{r_c}|^3} \left[\delta_{ab} - \frac{3.(r^a - r_c^a)(r^b - r_c^b)}{|\vec{r} - \vec{r_c}|^2} \right] \quad (1.19)$$

With δ_{ab} the Kronecker delta:

$$\delta_{ab} = \begin{cases} 1, & \text{if } a = b. \\ 0, & \text{if } a \neq b. \end{cases} \quad (1.20)$$

We note that a and b variate from 0 to 2 and $r^0 = x$, $r^1 = y$, and $r^2 = z$ as usual sense. For the term $||\frac{\partial \vec{f}}{\partial \vec{r} \partial \vec{r}}||$ we give the compact version by:

$$||\frac{\partial^2 f^a}{\partial r^b \partial r^c}|| = -\sum_c \frac{3m_c}{|\vec{r} - \vec{r_c}|^5} \left[\frac{5(r^a - r_c^a)(r^b - r_c^b)(r^c - r_c^c)}{|\vec{r} - \vec{r_c}|^2} - \right.$$
$$\left. \left(\delta_{ab}(r^c - r_c^c) + \delta_{bc}(r^a - r_c^a) + \delta_{ac}(r^b - r_c^b) \right) \right] \quad (1.21)$$

The equations 1.19 and 1.21 are use during the M2L step. Then to go down in the tree and apply the gravitation to locals and then particles in L2L and L2P we use equation 1.17.

This methods impose a lot of communications and exchanges between the processes. In our distributed version the particles are separate for each processes. Indeed, as each of them will hold part of the particles, the multipole in M2L computation imposes to share data. The P2P computation will face issues on the edge of each sub-domain. The irregular behavior is also present for the choice of the multipole to consider during the M2L step based on the MAC criterion.

1.3   Test cases with SPH

The generic SPH method and the gravitation computation, either simple or with FMM, allow us to run standard example to check the physical accuracy of our code. We choose four mains examples in this purpose: the Sod shock tube, the Sedov blast wave, fluid flow simulation and astrophysical examples like binary neutron stars coalescence. We present those problems in this section and their characteristics regarding the computer science difficulties involved.

1.3.1   Simple tests: tree and communications

Sod shock tube

The Sod shock tube is the test consisting of a one-dimensional Riemann problem with the following initial parameters [Sod78].

$$(\rho, v, p)_{t=0} = \begin{cases} (1.0, 0.0, 1.0) & \text{if} \quad 0 < x \leq 0.5 \\ (0.125, 0.0, 0.1) & \text{if} \quad 0.5 < x < 1.0 \end{cases} \quad (1.22)$$

In our code, we use the same initial data as in section 1.2.1 with ideal gas EOS such as:

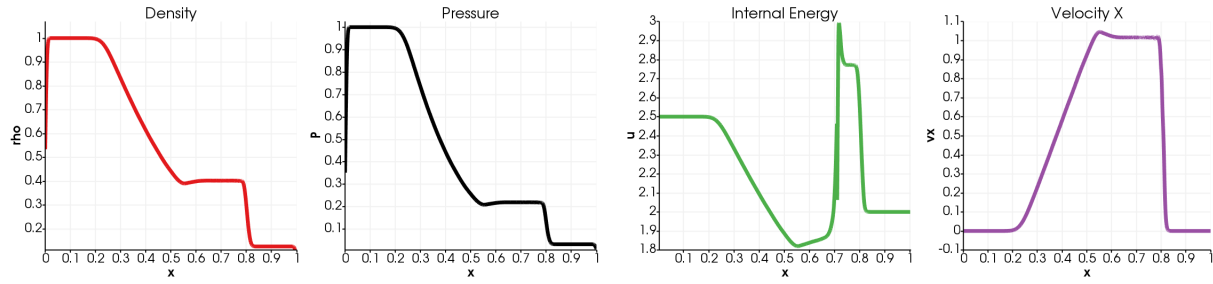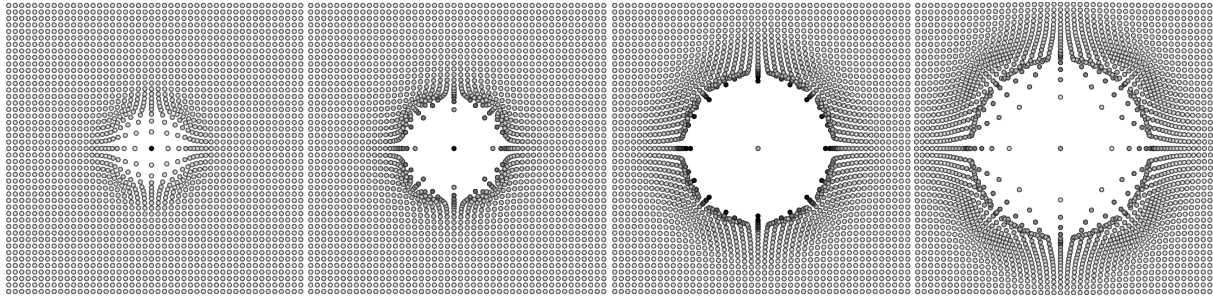$$P(\rho, u) = (\Gamma - 1)\rho u \quad (1.23)$$

Figure 1.4: Sod shock tube with FleCSPH

Figure 1.5: Sedov Blast Wave with FleCSPH at respectively $t = 0.01$, $t = 0.03$, $t = 0.06$ and $t = 0.1$

where Γ is the adiabatic index of the gas, we set $\Gamma = 5/3$.

This test is used to check the physical accuracy of the code and thus the tree search itself. A simulation of our Sod shock experimentation is presented on figure 1.4 and shows physically correct results. The first difficulty of this problem is the tree repartition, the physics behind is not complicated and does not involve specific optimizations.

Sedov blast wave

A blast wave is the pressure and flow resulting from the deposition of a large amount of energy in a small very localized volume. There are different versions of blast wave test and we consider comparing it with the analytic solution for a point explosion as given by Sedov [Sed46], making the assumption that the atmospheric pressure relative to the pressure insider the explosion negligible. Here, we test 2D blast wave. In this simulation, we use ideal gas EOS with $\Gamma = 5/3$ and we are assuming that the undistributed area is at rest with a pressure $P_0 = 1.0^{-5}$. The density is constant ρ_0, also in the pressurized region.

An example of our Sedov Blast wave experimentation is presented on figure 1.5 and shows physically correct results. This problem have the same conclusion as the Sod shock tube, his purpose is the test the tree behavior with 2 dimensions.

1.3.2 High number of data and specific formulations

After performing the tests regarding the physics reliability, we worked on fluid flow problem in 2D and 3D to reach high number of particles. The details can be found in [GGRC+12]. This test is based on an ideal EOS given by:

$$P = B\left[\left(\frac{\rho}{\rho_0}\right)^\gamma - 1\right] \tag{1.24}$$

with $\gamma = 7$ and $B = c_0 \rho_0 / \gamma$ being $\rho_0 = 1000 \, kg.m^{-3}$ the reference density.

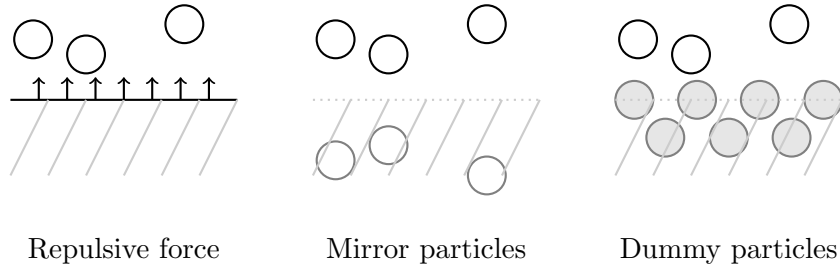Repulsive force   Mirror particles   Dummy particles

Figure 1.6: Different boundaries condition methods

Boundary conditions

For this experiment, realistic boundaries conditions were needed. Several methods are possible with SPH and we focused on the main ones: the repulsive wall, the mirror particles [LP91] and the dummies particles implementation [AHA12]. Those boundaries conditions implementation are presented in figure 1.6.

On our first implementation we used repulsive forces, but they imposed a lot of computation during the integration step and does not handle all the shape of boundaries easily.

For the current implementation we used the dummies particles method. This method is accurate and allows us to consider all the shape of boundaries. The wall particles are just considered as normal particles, with specific equations, and their quantities are evolved during the run. The main difference is that their position does not evolve at the end of the step. They are identified in the code with a specific type, provided during the data generation.

In this fluid flow example we were able to go up to a high number of particles with 2 or 3 dimensions. The handling of boundaries conditions added a pressure on the irregularity behavior.

1.3.3   SPH and gravitation

The final aim of our tests is to simulate astrophysical events. We are interested in one of the most important event recently discovered. Last year the Laser Interferometer Gravitational Wave, LIGO, detected the first gravitational wave generated by binary neutron stars merging [AAA+17b] and also more complexes event with Binary Black Holes coalescence in [AAA+17a]. We decided to conduct tests on Binary Neutron Star, BNS, coalescence. This problem represent all the aspect needed in our benchmark with the SPH method but also the self-gravitation and gravitation computation. In order to perform realistic simulations we consider two stars that are stable. We give details on the way to generate those data and the physics of the fusion itself. A lot of work have been involve in this understanding since it is quite out of the field of the primary researches.

Solving Lane-Emden Equation

In order to consider BNS we first consider two individual stars. The star is characterized by its radius and mass but also the density repartition inside. Indeed, we have to determine the density function based on the radius. For this purpose we used the so called Land-Emden equation.

REFERENCE As we consider the star as a polytropic fluid, we use the equation of Lane-Emden which is a form of the Poisson equation:

$$\frac{d^2\theta}{d\xi^2} + \frac{2}{\xi}\frac{d\theta}{d\xi} + \theta^n = 0 \tag{1.25}$$

With ξ and θ two dimensionless variables. There is only exact solutions for a polytropic index $n = 0.5$, 1 and 2. In our work we use a polytropic index of 1 which can correspond to a

	NS$_1$	NS$_2$	NS$_3$	NS$_4$
Radius (cm)	$R = G = M = 1$	1500000	1400000	960000
K	0.636619	95598.00	83576.48	39156.94

Table 1.1: Examples of the proportionality constant K regarding the star stellar radius R

NS simulation.

For $n = 1$ the solution of equation 1.25 is:

$$\theta(\xi) = \frac{sin(\xi)}{\xi} \tag{1.26}$$

We note $\xi_1 = \pi$, the first value of ξ with $\theta(\xi) = 0$. $\theta(\xi)$ is also defined as:

$$\theta(\xi) = \left(\frac{\rho(\xi)}{\rho_c}\right)^{\frac{1}{n}} = \frac{\rho(\xi)}{\rho_c} \tag{1.27}$$

With ρ_c the internal density of the star and ρ the density at a determined radius. ξ is defined as:

$$\xi = \sqrt{\frac{4\pi G}{K(n+1)}\rho_c^{(n-1)/n}} \times r = \sqrt{\frac{2\pi G}{K}} \times r = Ar \text{ (for } n = 1 \text{ and } A = \sqrt{\frac{2\pi G}{K}})$$

With K a proportionality constant, r the radius and G the gravitational constant.

From the previous equations we can write the stellar radius R, the total radius of the star, as:

$$R = \sqrt{\frac{K(n+1)}{4\pi G}}\rho_c^{(1-n)/2}\xi_1 = \sqrt{\frac{K}{2\pi G}} \times \xi_1 \text{ (for } n = 1) \tag{1.28}$$

We note that for $n = 1$ the radius does not depend of the central density.

Here as an example we use dimensionless units as $G = R = M = 1$ (for the other results we use CGS with $G = 6.674 \times 10^{-8}cm^3g^{-1}s^{-2}$) We can compute K as:

$$K = \frac{R^2 2\pi G}{\xi_1^2} \tag{1.29}$$

Figure 1.1 give example of radius and the proportionality constant K.

Then we deduce the density function of r as :

$$\rho(\xi) = \frac{sin(A \times r)}{A \times r} \times \rho_c \text{ with } A = \sqrt{\frac{2\pi G}{K}}$$

As we know the total Mass M, the radius R and the gravitational constant G we can compute the central density as:

$$\rho_c = \frac{MA^3}{4\pi(sin(AR) - ARcos(AR))}$$

Then we normalize the results to fit $R = M = G = 1$: $K' = K/(R^2G)$, $m'_i = m_i/M$, $h'_i = h_i/R$, $\vec{x}_i' = \vec{x}_i/R$

Generating BNS data

The initial data are based on a cubic lattice within a sphere of radius R. The density function, based on radius, $\rho(\vec{r})$ is known using the result of the Lane-Emden equation (we use polytropic index $n = 1$ here). The mass associate to each particle i of the total N particles:

$$m_i = \frac{\rho(\vec{r_i})}{n_r} \text{ with } n_r = \frac{3N}{4\pi R^3}$$

The smoothing length is define constant and the same for all particles for all the simulation:

$$h = \frac{1}{2}\sqrt{\frac{3N_N}{4\pi n}}$$

Here we choose N_N, the average number of neighbors, to be 100.

Roche lobe problem: To create Hydrostatic Equilibrium Models we use a different equation of motion. This version use Roche Lobe:

$$\frac{d\vec{v_i}}{dt} = \frac{\vec{F}_i^{Grav}}{m_i} + \frac{\vec{F}_i^{Hydro}}{m_i} + \vec{F}_i^{Roche} - \frac{\vec{v_i}}{t_{relax}} \tag{1.30}$$

With $t_{relax} \leq t_{osc} \sim (G\rho)^{-1/2}$ and where $\vec{F}_i^{Roche}$ is:

$$\vec{F}_i^{Roche} = \mu(3+q)x_i\hat{\vec{x}} + \mu q y_i \hat{\vec{y}} - \mu z_i \hat{\vec{z}}$$

With μ to be determined (for us $\mu = 0.069$) and $q = \frac{M'}{M} = 1$ as the two polytropes have the same total mass. This is apply to each star to get the equilibrium and the simulate the tidal effect.

Darwin problem: This is the way we use to generate the final simulation. The equation of motion for the relaxation is now:

$$\frac{d\vec{v_i}}{dt} = \frac{\vec{F}_i^{Grav}}{m_i} + \frac{\vec{F}_i^{Hydro}}{m_i} + \vec{F}_i^{Rot} - \frac{\vec{v_i}}{t_{relax}} \tag{1.31}$$

With t_{relax} same as before and $\vec{F}_{Rot}$ defined by:

$$\vec{F}_{Rot} = \Omega^2(x_i\vec{x} + y_i\vec{y}) \tag{1.32}$$

With $\Omega = \sqrt{\frac{G(M+M')}{a^3}}$, $L_z = Q_{zz}\Omega$ and $Q_{zz} = \sum_i(x_i^2 + y_i^2)$. At $t = 0$ we compute the total angular moment L_z which stay constant. Using it during the relaxation we can compute Ω as: $\Omega = \frac{L_z}{Q_{zz}}$ just by recomputing Q_{zz}. Here the scheme is in N^2 but just for the relaxation step.

For this relaxation we use two stars generated as before, applying equation of motion 1.31. Using a as the distance between the two polytropes (Here $a = 2.9$ for $R = 1$) and $\vec{x}$ going for the center of the first to the second star, and $\vec{z}$ is like the rotation vector.

After the generation we are able to perform BNS coalescence. This provides us a code using SPH, self-gravity and gravitation.

1.4   Conclusion

As the SPH method is used in a large panel of fields from astrophysics to fluid mechanic, there are numerous related works. We can cite a code developed in the LANL, 2HOT [War13] that introduced the Hashed Oct Tree structure used in our implementation. There is also GADGET-2 [Spr05], GIZMO [Hop14] and the most recent publication is GASOLINE [WKQ17] based on PKDGRAV, a specific tree+gravity implementation. Several implementations already implement GPU code and tree construction and traversal, one can cite GOTHIC [MU17], presenting gravitational tree code accelerated using the latest Fermi, Kepler and Maxwell architectures. But a lot of GPU accelerated work still focused on fluid problems and not on astrophysical problems [HKK07, CDB+11]. We also note that these implementations focus on SPH problems and does not provide a general purpose and multi-physics framework like we intent to provide through FleCSPH and FleCSI.

We chose application to test the tree architecture itself up to a high number of particles using the fluid flow. Plus sur les problemes informatiques

Chapter 2

FleCSPH

2.1 Introduction

The previous part describes the method we want to implement and presents the wall in the smoothed particle hydrodynamics method. We showed that this application meets the communication and computation wall in an irregular behavior context. FleCSPH is a complex application that, beside of being interesting for our purpose, need to be accurate on the physics aspect to be use by the domain scientists from LANL. I also note that this is my first work on developing a production code from scratch.

This section gives details about the FleCSPH framework. We first present FleCSI, the base project in the Los Alamos National Laboratory on which FleCSPH is based. For FleCSPH we give details on the domain decomposition strategy used with Morton Ordering. The tree traversal algorithm choices are then explained. We show the first results on the multi-CPU version of FleCSPH. In the last part we present our multi-GPU implementation of FleCSPH and the advantage using accelerators.

2.2 FleCSI

FleCSI[1] [BMC16] is a compile-time configurable framework designed to support multi-physics application development. It is developed at the Los Alamos National Laboratory as part of the Los Alamos Ristra project. As such, FleCSI provides a very general set of infrastructure design patterns that can be specialized and extended to suit the needs of a broad variety of solver and data requirements. FleCSI currently supports multi-dimensional mesh topology, geometry, and adjacency information, as well as n-dimensional hashed-tree data structures, graph partitioning interfaces, and dependency closures.

FleCSI introduces a functional programming model with control, execution, and data abstractions that are consistent both with MPI and with state-of-the-art, task-based runtimes such as Legion[BTSA12] and Charm++[KK93]. The abstraction layer insulates developers from the underlying runtime, while allowing support for multiple runtime systems including conventional models like asynchronous MPI.

The intent is to provide developers with a concrete set of user-friendly programming tools that can be used now, while allowing flexibility in choosing runtime implementations and optimization that can be applied to future architectures and runtimes.

FleCSI's control and execution models provide formal nomenclature for describing poorly understood concepts such as kernels and tasks. FleCSI's data model provides a low-buy-in approach that makes it an attractive option for many application projects, as developers are not locked into particular layouts or data structure representations.

FleCSI currently provides a parallel but not distributed implementation of Binary, Quad and
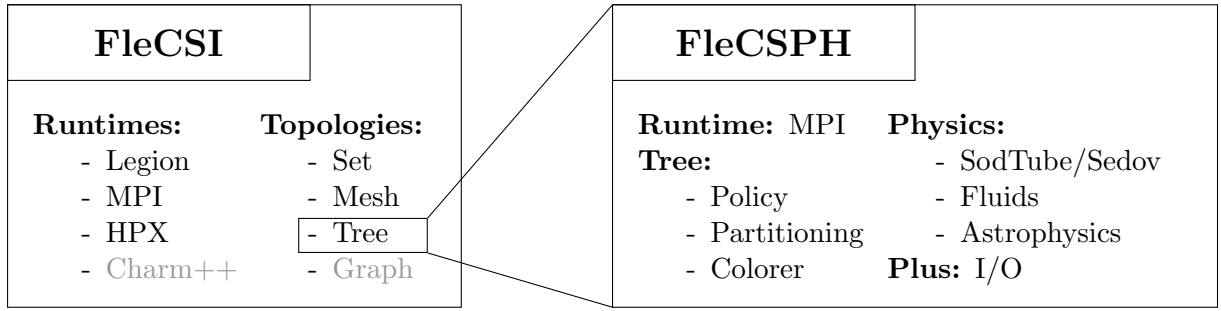
---

[1] `http://github.com/laristra/flecsi`

Figure 2.1: FleCSI and FleCSPH frameworks

Oct-tree topology. This implementation is base on space filling curves domain decomposition, the Morton order. The current version allows the user to specify the code main loop and the data distribution requested. The data distribution feature is not available for the tree data structure needed in our SPH code and we provide it in the FleCSPH implementation. The next step will be to incorporate it directly from FleCSPH to FleCSI as we reach a decent level of performance. As FleCSI is an on-development code the structure may change in the future and we keep track of these updates in FleCSPH.

Based on FleCSI the intent is to provide a binary, quad and oct-tree data structure and the methods to create, search and share information for it. In FleCSPH this will be dedicated, apply and tested on the SPH method. In this part we first present the domain decomposition, based on space filling curves, and the tree data structure. We describe the HDF5 files structure used for the I/O. Then we describe the distributed algorithm for the data structure over the MPI processes.

2.3   Distributed SPH on multi-core architecture

FleCSPH[2] is a framework initially created as a part of FleCSI. The purpose of FleCSPH is to provide a data distribution and communications patterns to use the tree topology provided in FleCSI applied to SPH. The final purpose of FleCSPH is to complement the tree topology and provide the colorer to FleCSI. The SPH seems to be a very good candidate to benchmark the binary, quad and oct tree topology. The demand is high for astrophysics simulation regarding the recent discoveries of the LIGO and the SPH method allows us to generate those simulations. Figure 2.1 present how FleCSI and FleCSPH are integrated. FleCSPH is base on the tree topology of FleCSI and will follow the same structure define in FleCI. The ideal runtime in FleCSI is Legion but this in-development code does not allows us to do more than static data distribution. This is why we decided to work with the MPI runtime in FleCSPH. Those MPI functions can then be integrated to FleCSI to generate group of particles and labeled them, the coloring.

Figure 2.2 present the file systems of the github repository. We use the tools from Cinch[4] developed at LANL for the CMake and the makefile generation. It also provide the GoogleTests API for our unit tests. The FleCSPH code is currently public and available on github under the *laristra* (Los Alamos Ristra) project. The continuous integration is ensured by using Travis based on Docker provided in the Docker folder. In addition to Travis for the unit tests we use use tools like CodeCov[5] for the code coverage and SonarQube[6] for the quality gates. In the current version we use external libraries: HDF5, H5hut and a specific library for I/O based on H5hut. Those elements are present and installed using scripts in *third-part-library* folder.

---

[2]`http://github.com/laristra/flecsph`
[4]`http://github.com/laristra/cinch`
[5]`http://codecov.io`
[6]`http://sonarqube.org/`

```
FleCSPH
   ├─ @cinch³ ──────── Utilities and configuration tools for cmake buils
   ├─ include ──────── Tree/Branches/Particles policies and physics
   ├─ app
   │    ├─ sedov
   │    ├─ sodtube
   │    ├─ fluid_3D
   │    └─ BNS
   ├─ data ──────── Example data for unit tests
   ├─ doc
   ├─ docker ──────── Dockerfiles for Travis+CodeCov+Sonarqube
   ├─ mpisph
   │    ├─ body_system.hpp
   │    ├─ tree_colorer.hpp
   │    └─ tree_fmm.hpp
   └─ third-part-library ── Required libraries like HDF5/H5Hut
```
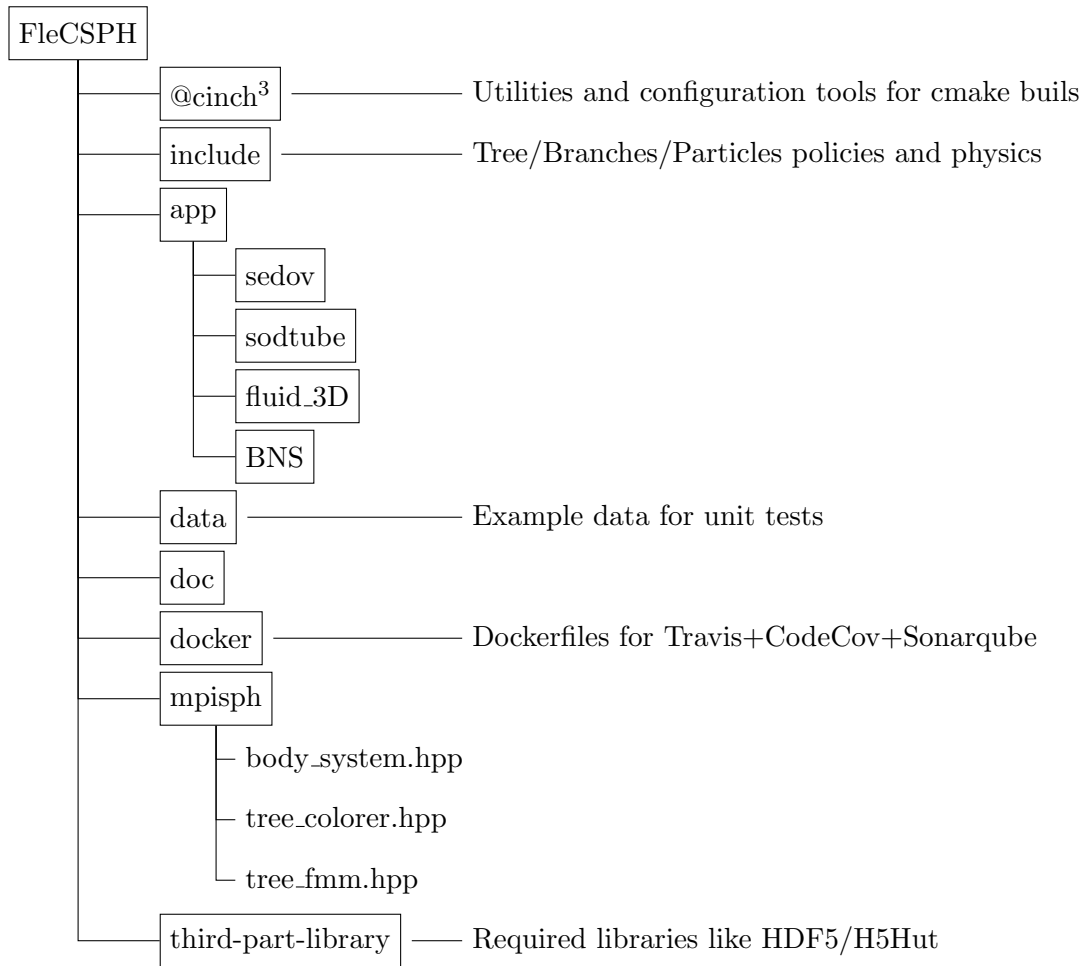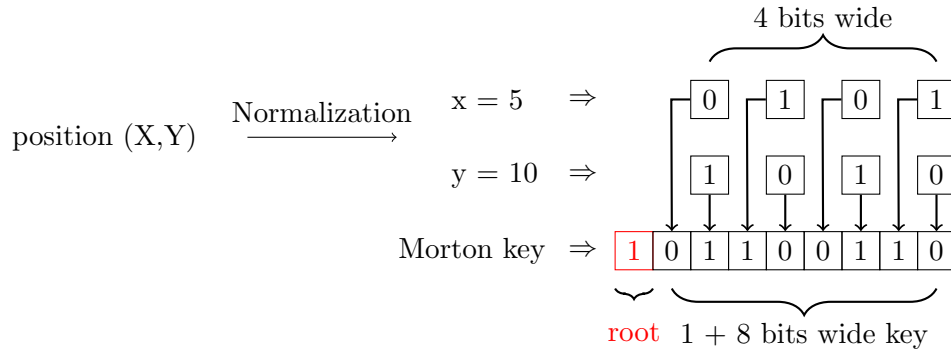
Figure 2.2: FleCSPH structures and files

Figure 2.3: Morton order key generation

For the first implementation we present the code without considering accelerators. We intent to provide an efficient multi-CPU distributed code. The description starts with the domain decomposition strategy which is a basic element for the tree implementation. We explain the tree data structure for the construction and the search of particles along with the distribution strategy.

2.3.1 Domain decomposition

The number of particles can be high and represent a huge amount of data that does not fit in a single node memory. This implies the distribution of the particles over several computational
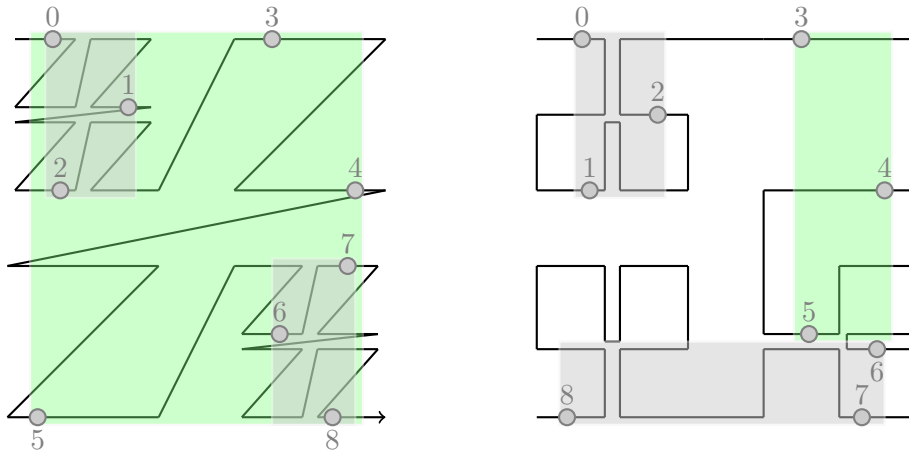
Figure 2.4: Morton and Hilbert space filling curves

nodes. As the particles moves during the simulation the static distribution is not possible and they have to be redistributed at some point in the execution. Furthermore, this distribution need to keep local particles in the same computation node to optimize the exchanges and computation itself.

A common approach is to distribute the particles over computational nodes using *space filling curves* like in [War13, Spr05, BGF$^+$14]. It intends to assign to each particle a key which is based on its spacial coordinates, then sorting particles based on those keys keeps particles grouped locally. Many space filling curves exists: Morton, Hilbert, Peano, Moore, Gosper, etc.

This domain decomposition is used in several layers for our implementation. On one hand, to spread the particles over all the MPI processes and provide a decent load balancing regarding the number of particles. On the other hand, it is also used locally to store efficiently the particles and provide a $O(Nlog(N))$ neighbor search complexity, instead of $O(N^2)$, using a tree representation describe in part 2.3.2.

Several space filling curves can fit our purposes:

The Morton curves

[Mor66], or Z-Order, is the most spread method. This method can produce irregular shape domain decomposition like shown in green on figure 2.4. The main advantage is to be fast to compute, the key is made by interlacing directly the X, Y and Z bits without rotations.

The Hilbert curves

[Sag12] are constructed by interlacing bits but also adding rotation based on the Gray code. This work is based on the Peano curves and also called Hilber-Peano. The construction is more complicated than Morton but allows a better distribution.

On figure 2.4, the Morton (left) and Hilbert (right) space-filling curves are represented in this example. The particles are distributed over 3 processes. The set of particles of the second process appears in green. As we can see there are discontinuities on the Morton case due to the Z-order "jump" over the space. This can lead to non-local particles and over-sharing of particles that will not be needed during the computation. In the Hilbert curve, the locality over the processes is conserved.

In this first implementation of FleCSPH we used the Morton ordering due to the computational cost. The next step of this work is to compare the computation time of different space filing curves.

Technically the keys are generated for each particle at each iteration because their position is expected to change over time. To be more efficient, the keys can stay the same during several
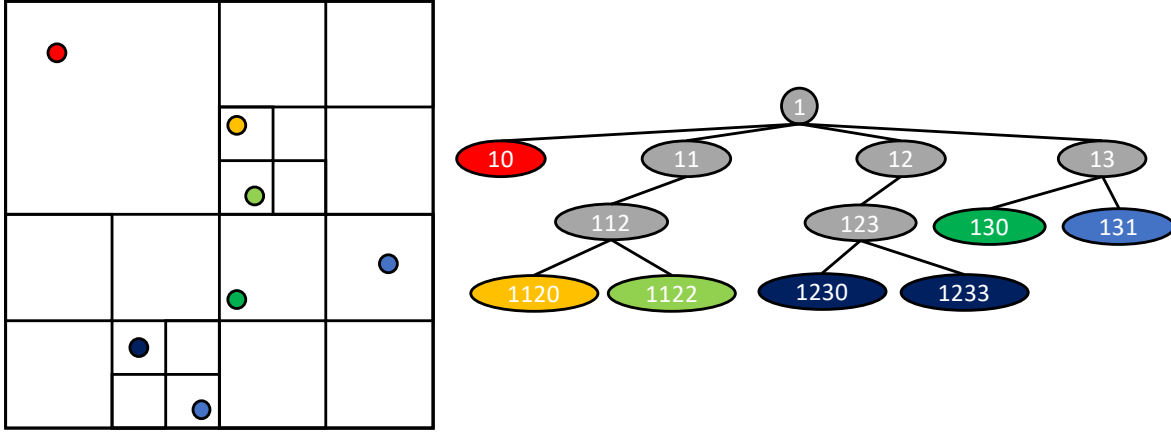
Figure 2.5: Quadtree, space and data representation

steps and the final comparison can be made on the real particles positions. This increase the search time but allows less tree reconstructions.

We use 64 bits to represent the keys to avoid conflicts. The FleCSI code allows us to use a combination of memory words to reach the desired size of keys (possibly more than 64 bits) but this will cost in memory occupancy. The particle keys are generated by normalizing the space and then converting the floating-point representation to a 64 bits integer for each dimensions. Then the Morton interlacing is done and the keys are created. Unfortunately in some arrangements, like isolated particles, or scenarios with very close particles, the keys can be either badly distributed or duplicate keys can appear. Indeed, if the distance between two particles is less than $2^{-64} \approx 1\mathrm{e}{+}20$, in a normalized space, the key generated through the algorithm will be the same. This problem is then handle during the particle sort and then the tree generation. In both case two particles can be differentiate based on their unique ID generated at the beginning execution.

2.3.2 Hierarchical trees

The method we use for the tree data structure creation and research comes from Barnes-Hut trees presented in [BH86, Bar90]. By reducing the search complexity from $O(N^2)$ for direct summation to $O(Nlog(N))$ it allows us to do very large simulations with billions of particles. It also allows the use of the tree data structure to compute gravitation using multipole methods.

We consider binary trees, for 1 dimension, quad-trees, for 2 dimensions, and oct-trees, for 3 dimensions. The construction of those trees is directly based on the domain decomposition using keys and space-filling curve presented in section 2.3.1.

As explain in the previous section, we use 64 bits keys. That give us up to 63, 31 and 21 levels in the tree for respectively 1, 2 and 3 dimensions. As presented on figure 2.5 the first bit is use to represent the root of the tree, 1. This allows us to have up to 2^{63} different keys and unique particles.

Tree generation

After each particle get distributed on its final process using its space-filling curve key, we can recursively construct the tree. Each particle is added and the branches are created recursively if there is an intersection between keys. Starting from the root of key "1" the branches are added at each levels until the particles are reached. An example of a final tree is shown on figure 2.5.

Tree search

When all the particles have been added, the data regarding the tree nodes are computed with a bottom up approach. Summing up the mass, position called Center of Mass (COM), and the boundary box of all sub-particles of this tree node.

For the search algorithm the basic idea would be to do a tree traversal for all the particles and once we reach a particle or a node that interact with the particle smoothing length, add it for computation or in a neighbor list. Beside of being easy to implement and to use in parallel this algorithm requires a full tree traversal for every particle and will not take advantage of the particles' locality.
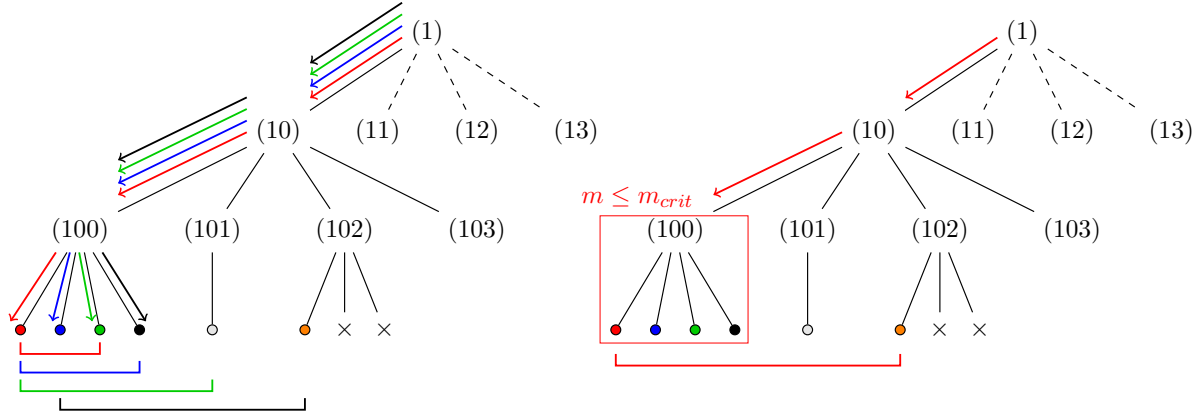
Figure 2.6: Neighbors search using a tree traversal per particle vs a group of particle and computing an interaction list

Our search algorithm, presented on Algorithm 2, is a two step algorithm like in Barnes trees: First create the interaction lists and then using them on the sub-tree particles. In the first step we look down for nodes with a target sub-mass of particles *tmass*. Then for those branches we compute an interaction list and continue the recursive tree search. When a particle is reached, we compute the physics using the interaction list as the neighbors. The interaction list is computing using an opening-angle criterion comparing the boundary box and an user define angle. This way we will not need a full tree traversal for each particle but a full tree traversal for every group of particles. On figure 2.6 we present the classical and the two steps algorithm. We see that the first method, on left, force to do one walk per particle, compute the interaction list and then apply to particles. On the left, the two step method, only perform one tree traversal for the whole block of particles, compute the interaction list and then processed to the local computation. Indeed, this implies a check of particle distance during the computation since all the particles from the interaction list are not useful for every particle.

2.3.3   Distribution strategies

The previous section presented the tree data structure that can be use locally on every node. The distribution layer is added on top of it, keeping each sub-tree on the computation nodes. The current version of FleCSPH is still based on synchronous communications using the Message Passing Interface (MPI).

The main distributed algorithm is presented on algorithm 3:

Particle distribution

The sort step, line 5, is base on a distributed quick sort algorithm. The keys are generated using the Morton order described in part 2.3.1. As we associate a unique number to each particle we are able to sort them using the keys and, in case of collision keys, using their unique ID. This

---

Algorithm 2 Tree search algorithm

1: **procedure** FIND_NODES
2:     stack $stk \leftarrow$ root
3:     **while** not_empty(stk) **do**
4:         branch $b \leftarrow stk$.pop()
5:         **if** b is leaf **then**
6:             **for each** particles p of b **do**
7:                 $apply\_sub\_tree(p,interaction\_list(p))$
8:             **end for**
9:         **else**
10:             **for each** child branch c of b **do**
11:                 stk.push(c)
12:             **end for**
13:         **end if**
14:     **end while**
15: **end procedure**
16:
17: **procedure** APPLY_SUB_TREE(node n, node-list nl)
18:     stack $stk \leftarrow n$
19:     **while** not_empty(stk) **do**
20:         branch $b \leftarrow stk$.pop()
21:         **if** b is leaf **then**
22:             **for each** particles p of b **do**
23:                 $apply\_physics(p,nl)$
24:             **end for**
25:         **else**
26:             **for each** child branch c of b **do**
27:                 stk.push(c)
28:             **end for**
29:         **end if**
30:     **end while**
31: **end procedure**
32:
33: **function** INTERACTION_LIST(node n)
34:     stack $stk \leftarrow$ root
35:     node-list $nl \leftarrow \emptyset$
36:     **while** not_empty(stk) **do**
37:         branch $b \leftarrow stk$.pop()
38:         **if** b is leaf **then**
39:             **for each** particles p of b **do**
40:                 **if** within() **then**
41:                     $nl \leftarrow nl + p$
42:                 **end if**
43:             **end for**
44:         **else**
45:             **for each** child branch c of b **do**
46:                 **if** $mac(c,angle)$ **then**
47:                     $nl \leftarrow nl + c$
48:                 **else**
49:                     stk.push(c)
50:                 **end if**
51:             **end for**
52:         **end if**
53:     **end while**
54: **end function**

Algorithm 3 Main algorithm

1: **procedure** SPECIALIZATION_DRIVER(input data file f)
2:     Read f in parallel
3:     Set physics constant from f
4:     **while** iterations **do**
5:         Distribute the particles using distributed quick sort
6:         Compute total range
7:         Generate the local tree
8:         Share branches
9:         Compute the ghosts particles
10:         Update ghosts data
11:         *PHYSICS*
12:         Update ghosts data
13:         *PHYSICS*
14:         Distributed output to file
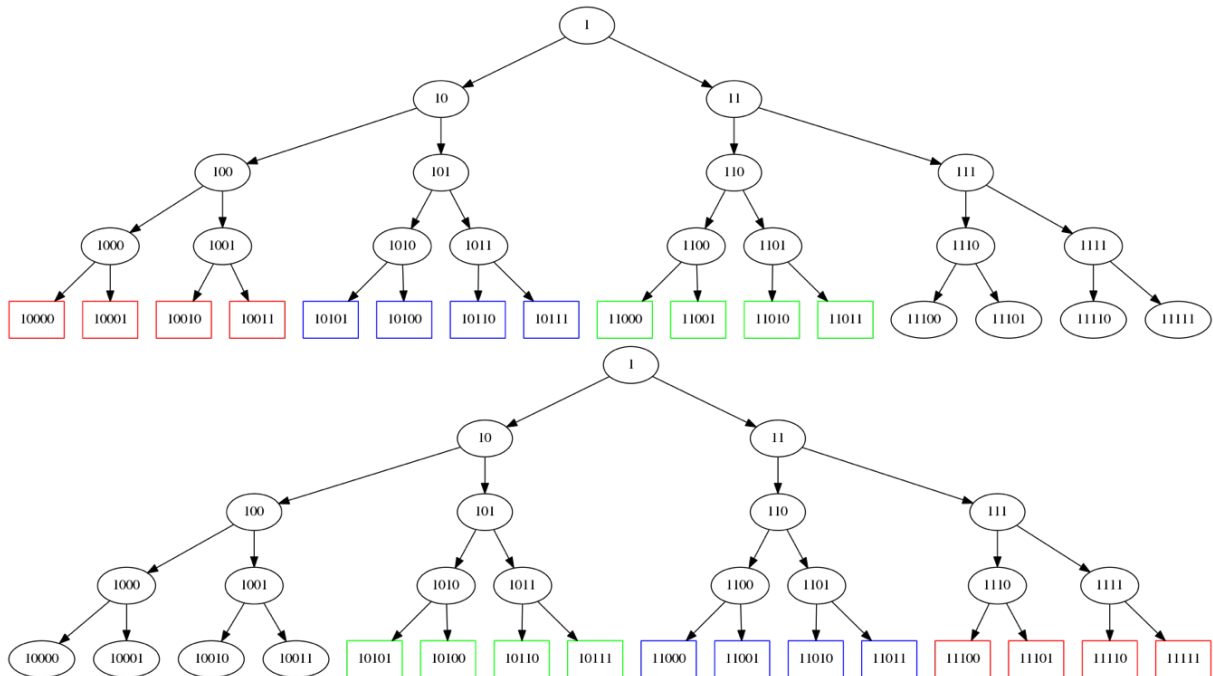15:   **end while**
16: **end procedure**

Figure 2.7: Binaries tree for a 2 processes system. Exclusive, Shared and Ghosts particles resp. red, blue, green.
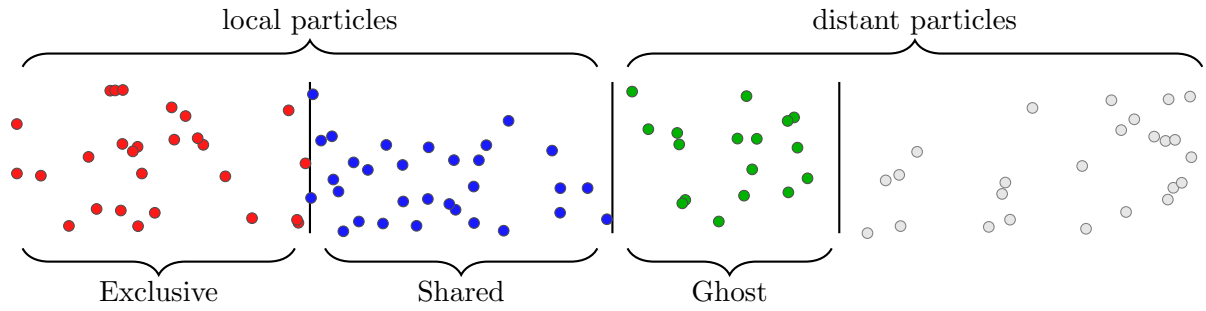
Figure 2.8: Particles "coloring" with local particles: exclusive, shared and distant particles useful during run: ghost particles

gives us a global order for the particles. Each process sends to a master node (or submaster for larger cases) a sample of its keys. We determined this size to be 256 KB of key data per process for our test cases but it can be refined for larger simulations. Then the master determines the general ordering for all the processes and shares the pivots. Then each process locally sort its local keys and, in a global communication step, the particles are distributed to the process on which they belong. This algorithm gives us a good partition in term of number of particles. But some downside can be identified:

- The ordering may not be balanced in term of number of particles per processes. But by optimizing the number of data exchanged to the master can lead to better affectation.

- The load balance also depend on the number of neighbors of each particle. If a particle get affected a poor area with large space between the particles, this can lead to bad load balancing too.

This is why we also provide another load balancing based on the particles neighbors. Depending on the user problem, the choice can be to distribute the particles on each processes regarding the number of neighbors, having the same amount of physical computation to perform on each processes.

After this first step, the branches are shared between the different processes, line 8. Every of them send to its neighbors several boundaries boxes, defined by the user. Then particles from the neighbors are computed, exchanged and added in the local tree. Those particles are labeled as NON_LOCAL particles. At this point a particle can be referenced as: EXCLUSIVE: will never be exchanged and will only be used on this process; SHARED: may be needed by another process during the computation; GHOSTS: particles information that the process need to retrieve from another process. An example is given for 2 processes on figure 2.8 and on a tree data structure on figure 2.7.

Exchange Shared and Ghosts particles

The previous distribution shares the particles and the general information about neighbors particles. Then each process is able to do synchronously or asynchronously communications to gather distant particles. In the current version of FleCSPH an extra step is required to synchronously share data of the particles needed during the next tree traversal and physics part. Then after this step, the ghosts data can be exchanged as wanted several times during the same time step.

2.3.4 I/O

Regarding the high number of particles, an efficient, parallel and distributed I/O implementation is required. Several choices were available but we wanted a solution that can be specific for our
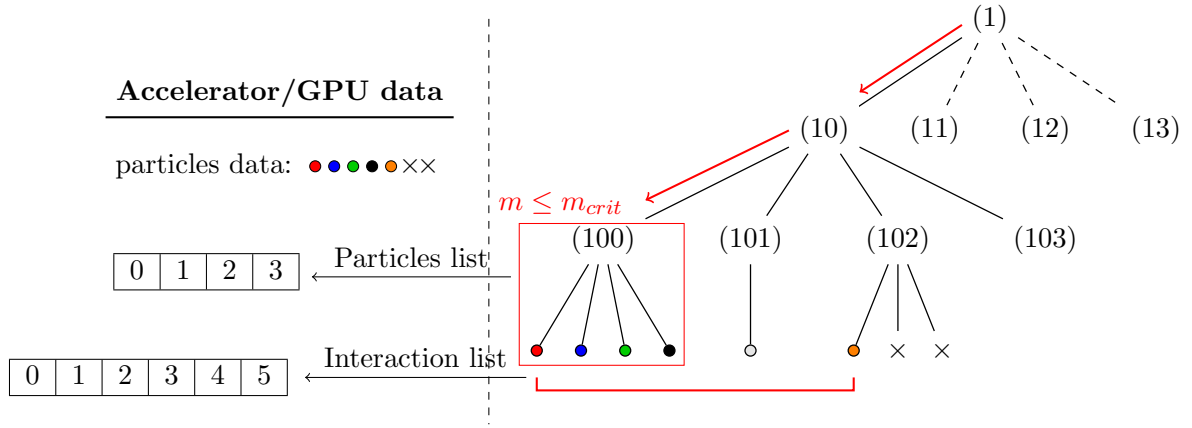
Figure 2.9: Task resolution using GPUs

usage. The first requirement is to allow the user to work directly with the Paraview visualization tool and splash[7] [Pri07].

We base this first implementation on HDF5 [FCY99] file structure with H5Part and H5Hut [HAB+10]. HDF5 support MPI runtime with distributed read and write in a single or multiple files. We added the library H5hut to add normalization in the code to represent global data, steps, steps data and the particles data for each steps. The I/O code was developed internally at LANL and provides a simple way to write and read the data in H5Part format. The usage of H5Hut to generate H5part data files allows us to directly read the output in Paraview without using a XDMF descriptor like requested in HDF5 format.

2.3.5   Further-works

The FleCSPH code for this version is based on synchronous communication. This reduces the scalability performances.

2.4   Distributed SPH on hybrid architectures

We constructed an efficient and reliable SPH code working on multi-CPU clusters. In this section we present our multi-GPU implementation and compare it with our multi-CPU code. We kept the same data structure, distribution strategy and code architecture in the accelerator code. Several options are possible for the accelerator implementation but we wanted to keep the code usable and working for the domain scientists to work on. As the current version of FleCSI does not allow utilization of accelerator like GPU for the data structure, we decided to embedded the GPU code directly on FleCSPH. We provided the same approach we studies in the Langford problem offloading part of the tree computation on the accelerators.

2.4.1   Distribution strategies

During our study we The first one is to implement the whole tree traversal and data representation on GPU. This strategy imposes several downside especially for asynchronous communication.

Figure 2.9 present the choose distribution of tasks with the accelerator. The tree traversal itself stay on the host processor and lower part of the tree are offloaded to the accelerators.

---

[7]http://users.monash.edu.au/~dprice/splash/

2.4.2   Physics on accelerators

2.5   Results and comparison

In this part we compare the results of the multi-GPU version and the multi-CPU version of FleCSPH. We decided to conduct the tests on the 2D and 3D fluids.

2.6   Conclusion

In this part we presented the FleCSPH framework base on FleCSI. We gave details on the code structure and the strategies taken for the multi-CPU version. We presented the tree traversal, divided in two steps in order to generated better interaction lists for particles.

The results of the multi-CPU code are presented and shows good results in both side, physics and computer science.

In the last section we presented the multi-GPU code. We showed that

Conclusion

This last application complete our metrics to target hybrid supercomputers. The SPH method

Bibliography

[AAA+17a] Benjamin P Abbott, R Abbott, TD Abbott, F Acernese, K Ackley, C Adams, T Adams, P Addesso, RX Adhikari, VB Adya, et al. Gw170814: A three-detector observation of gravitational waves from a binary black hole coalescence. *Physical review letters*, 119(14):141101, 2017.

[AAA+17b] Benjamin P Abbott, Rich Abbott, TD Abbott, Fausto Acernese, Kendall Ackley, Carl Adams, Thomas Adams, Paolo Addesso, RX Adhikari, VB Adya, et al. Gw170817: observation of gravitational waves from a binary neutron star inspiral. *Physical Review Letters*, 119(16):161101, 2017.

[AHA12] S Adami, XY Hu, and NA Adams. A generalized wall boundary condition for smoothed particle hydrodynamics. *Journal of Computational Physics*, 231(21):7057–7075, 2012.

[Bar90] Joshua E Barnes. A modified tree code: don't laugh; it runs. *Journal of Computational Physics*, 87(1):161–170, 1990.

[BG97] Rick Beatson and Leslie Greengard. A short course on fast multipole methods. *Wavelets, multilevel methods and elliptic PDEs*, 1:1–37, 1997.

[BGF+14] Jeroen Bédorf, Evghenii Gaburov, Michiko S Fujii, Keigo Nitadori, Tomoaki Ishiyama, and Simon Portegies Zwart. 24.77 pflops on a gravitational tree-code to simulate the milky way galaxy with 18600 gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 54–65. IEEE Press, 2014.

[BH86] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.

[BMC16] Ben Bergen, Nicholas Moss, and Marc Robert Joseph Charest. Flexible computational science infrastructure. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 2016.

[BTSA12] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. Legion: Expressing locality and independence with logical regions. In *Proceedings of the international conference on high performance computing, networking, storage and analysis*, page 66. IEEE Computer Society Press, 2012.

[CDB+11] Alejandro C Crespo, Jose M Dominguez, Anxo Barreiro, Moncho Gómez-Gesteira, and Benedict D Rogers. Gpus, a new tool of acceleration in cfd: efficiency and reliability on smoothed particle hydrodynamics methods. *PloS one*, 6(6):e20685, 2011.

[DRZR17] Zili Dai, Huilong Ren, Xiaoying Zhuang, and Timon Rabczuk. Dual-support smoothed particle hydrodynamics for elastic mechanics. *International Journal of Computational Methods*, 14(04):1750039, 2017.

[FCY99]   Mike Folk, Albert Cheng, and Kim Yates. Hdf5: A file format and i/o library for high performance computing applications. In *Proceedings of Supercomputing*, volume 99, pages 5–33, 1999.

[GGRC⁺12]   Moncho Gomez-Gesteira, Benedict D Rogers, Alejandro JC Crespo, Robert A Dalrymple, Muthukumar Narayanaswamy, and José M Dominguez. Sphysics–development of a free-surface fluid solver–part 1: Theory and formulations. *Computers & Geosciences*, 48:289–299, 2012.

[GM77]   Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

[GM82]   RA Gingold and JJ Monaghan. Kernel estimates as a basis for general particle methods in hydrodynamics. *Journal of Computational Physics*, 46(3):429–453, 1982.

[HAB⁺10]   Mark Howison, Andreas Adelmann, E Wes Bethel, Achim Gsell, Benedikt Oswald, et al. H5hut: A high-performance i/o library for particle-based simulations. In *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*, pages 1–8. IEEE, 2010.

[HKK07]   Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed particle hydrodynamics on gpus. In *Computer Graphics International*, pages 63–70. SBC Petropolis, 2007.

[Hop14]   Philip F Hopkins. Gizmo: Multi-method magneto-hydrodynamics+ gravity code. *Astrophysics Source Code Library*, 2014.

[KK93]   Laxmikant V Kale and Sanjeev Krishnan. Charm++: a portable concurrent object oriented system based on c++. In *ACM Sigplan Notices*, volume 28, pages 91–108. ACM, 1993.

[LL59]   L. D. Landau and E. M. Lifshitz. *Fluid mechanics*. 1959.

[LP91]   Larry D Libersky and AG Petschek. Smooth particle hydrodynamics with strength of materials. In *Advances in the free-Lagrange method including contributions on adaptive gridding and the smooth particle hydrodynamics method*, pages 248–257. Springer, 1991.

[LSR16]   SJ Lind, PK Stansby, and Benedict D Rogers. Incompressible–compressible flows with a transient discontinuous interface using smoothed particle hydrodynamics (sph). *Journal of Computational Physics*, 309:129–147, 2016.

[Luc77]   Leon B Lucy. A numerical approach to the testing of the fission hypothesis. *The astronomical journal*, 82:1013–1024, 1977.

[MG83]   J.J Monaghan and R.A Gingold. Shock simulation by the particle method sph. *Journal of Computational Physics*, 52(2):374 – 389, 1983.

[Mor66]   Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966.

[MU17]   Yohei Miki and Masayuki Umemura. Gothic: Gravitational oct-tree code accelerated by hierarchical time step controlling. *New Astronomy*, 52:65–81, 2017.

[Pri07]   Daniel J Price. Splash: An interactive visualisation tool for smoothed particle hydrodynamics simulations. *Publications of the Astronomical Society of Australia*, 24(3):159–173, 2007.

[RDZR16] Huilong Ren, Zili Dai, Xiaoying Zhuang, and Timon Rabczuk. Dual-support smoothed particle hydrodynamics. *arXiv preprint arXiv:1607.08350*, 2016.

[Ros09] Stephan Rosswog. Astrophysical smooth particle hydrodynamics. *New Astronomy Reviews*, 53(4):78 – 104, 2009.

[Sag12] Hans Sagan. *Space-filling curves*. Springer Science & Business Media, 2012.

[Sed46] Leonid I Sedov. Propagation of strong shock waves. *Journal of Applied Mathematics and Mechanics*, 10:241–250, 1946.

[Sod78] Gary A Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1 – 31, 1978.

[Spr05] Volker Springel. The cosmological simulation code gadget-2. *Monthly Notices of the Royal Astronomical Society*, 364(4):1105–1134, 2005.

[War13] Michael S Warren. 2hot: an improved parallel hashed oct-tree n-body algorithm for cosmological simulation. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 72. ACM, 2013.

[WKQ17] James W Wadsley, Benjamin W Keller, and Thomas R Quinn. Gasoline2: a modern smoothed particle hydrodynamics code. *Monthly Notices of the Royal Astronomical Society*, 471(2):2357–2369, 2017.

[YB11] Rio Yokota and Lorena A Barba. Treecode and fast multipole method for n-body simulation with cuda. In *GPU Computing Gems Emerald Edition*, pages 113–132. Elsevier, 2011.