Part I

Application

Introduction

The first part presented the tools needed to understand and target performance in HPC. The second part presented a 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 conclude 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. 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

General problem

1.1 Introduction

In this section we give details on our choices for the generic application confronted to both computation and communication walls on irregular context. This problem, Smoothed Particle Hydrodynamics, is described on the physics aspect and the difficulties involved in the resolution on supercomputers.

1.2 Smoothed Particle Hydrodynamics

1.2.1 General description

Smoothed Particle Hydrodynamics (SPH) is an explicit numerical mesh-free Lagrangian method used to solve hydrodynamical partial differential equations (PDEs) by discretized it 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 Fig. 1.1, computes the evolution of physical quantities for every particle regarding its neighbors in the radius of its smoothing length h. The particles in this

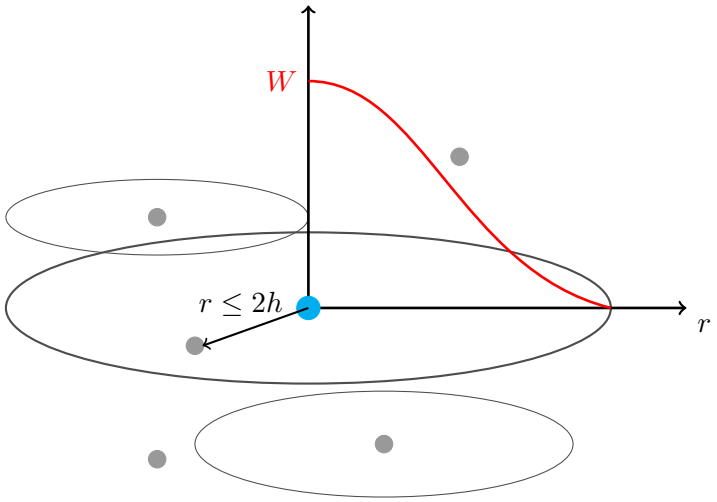

Figure 1.1: SPH kernel W and smoothing length h representation

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 particle by:

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

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 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 is restricted to low-order rate of convergence on certain 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 mass, energy and momentum of an ideal fluid [LL59] such that:

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \vec{v}, \quad \frac{du}{dt} = \left(\frac{P}{\rho^2}\right) \frac{d\rho}{dt}, \quad \frac{d\vec{v}}{dt} = -\frac{\nabla P}{\rho} \tag{1.2}$$

with ρ the density, P the pressure, u the internal energy and v the velocity, where $d/dt = \partial_t + \vec{v} \cdot \nabla$ 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 different way but 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. Here, we use $\epsilon = 0.01h^2$, $\alpha = 1.0$, 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:

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

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:

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

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 **??**.

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.

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

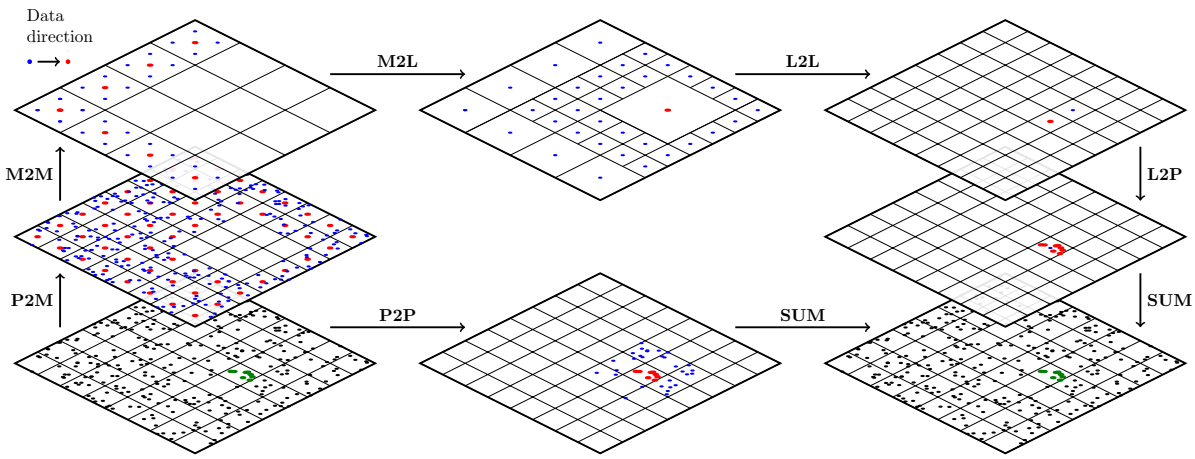

Figure 1.2: Fast Multipole Method schematics. Particles to Multipole (P2M), Multipole to Multipole (M2M), Multipole to Particles (M2P), Multipole to Local (M2L), Local to Local (L2L) and Particles to Particles (P2P). Schematic inspired from [YB11]

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.

1.2.2 Gravitation

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

$$\vec{a_g} = m\vec{g} \tag{1.14}$$

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

$$\vec{f}_{ai} = \sum_j -G\frac{m_i m_j}{|\vec{r_i} - \vec{r_j}|^3}\vec{r_{ij}} \tag{1.15}$$

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]. In this method we compute the gravitation up a approximations. The user can refine those approximation changing parameters.

This method is based on Taylor series. The gravitation function of equation 1.15 can be approximate on a particle at position $\vec{r}$ by the gravitation computed at the centroid at position $\vec{r_c}$:

$$\vec{f}(\vec{r}) = \vec{f}(\vec{r_c}) + ||\frac{\partial \vec{f}}{\partial \vec{r}}|| \cdot (\vec{r} - \vec{r_c}) + \frac{1}{2}(\vec{r} - \vec{r_c})^\intercal \cdot ||\frac{\partial \vec{f}}{\partial \vec{r}\partial \vec{r}}|| \cdot (\vec{r} - \vec{r_c}) \tag{1.16}$$

From equation 1.15 we compute the term $||\frac{\partial \vec{f}}{\partial \vec{r}}||$:s

$$\frac{\partial \vec{f}}{\partial \vec{r}} = -\sum_p \frac{m_p}{|\vec{r_c} - \vec{r_p}|^3} \begin{bmatrix} 1 - \frac{3(x_c - x_p)(x_c - x_p)}{|\vec{r_c} - \vec{r_p}|^2} & -\frac{3(y_c - y_p)(x_c - x_p)}{|\vec{r_c} - \vec{r_p}|^2} & -\frac{3(z_c - z_p)(x_c - x_p)}{|\vec{r_c} - \vec{r_p}|^2} \\ -\frac{3(x_c - x_p)(y_c - y_p)}{|\vec{r_c} - \vec{r_p}|^2} & 1 - \frac{3(y_c - y_p)(y_c - y_p)}{|\vec{r_c} - \vec{r_p}|^2} & -\frac{3(z_c - z_p)(y_c - y_p)}{|\vec{r_c} - \vec{r_p}|^2} \\ -\frac{3(x_c - x_p)(z_c - z_p)}{|\vec{r_c} - \vec{r_p}|^2} & -\frac{3(y_c - y_p)(z_c - z_p)}{|\vec{r_c} - \vec{r_p}|^2} & 1 - \frac{3(z_c - z_p)(z_c - z_p)}{|\vec{r_c} - \vec{r_p}|^2} \end{bmatrix} \tag{1.17}$$

And we propose a compact version of the matrix with:

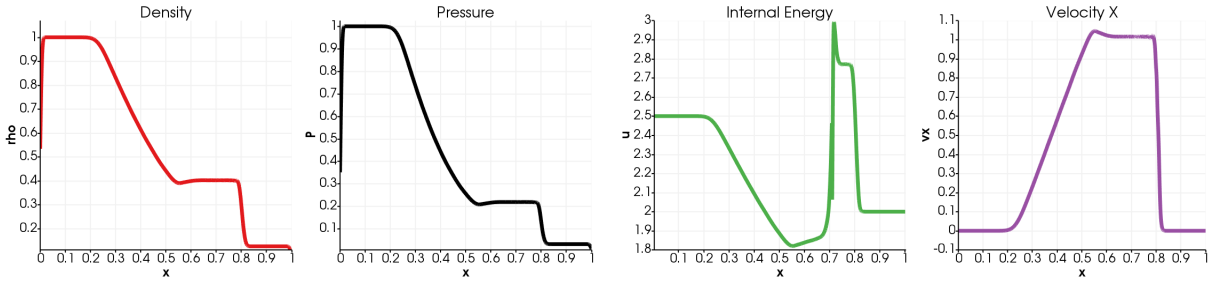

Figure 1.3: Sod shock tube with FleCSPH

$$||\frac{\partial f^a}{\partial r^b}|| = -\sum_c \frac{m_c}{|\vec{r}-\vec{r_c}|^3}\Big[\delta_{ab} - \frac{3.(r^a - r_c^a)(r^b - r_c^b)}{|\vec{r}-\vec{r_c}|^2}\Big] \tag{1.18}$$

With δ_{ab} the kronecker delta:

$$\delta_{ab} = \begin{cases} 1, & \text{if } a = b. \\ 0, & \text{if } a \neq b. \end{cases} \tag{1.19}$$

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:

$$\begin{aligned} ||\frac{\partial^2 f^a}{\partial r^b \partial r^c}|| = -\sum_c \frac{3m_c}{|\vec{r}-\vec{r_c}|^5}\Big[\frac{5(r^a - r_c^a)(r^b - r_c^b)(r^c - r_c^c)}{|\vec{r}-\vec{r_c}|^2} - \\ \Big(\delta_{ab}(r^c - r_c^c) + \delta_{bc}(r^a - r_c^a) + \delta_{ac}(r^b - r_c^b)\Big)\Big] \end{aligned} \tag{1.20}$$

The method is summed up in figure with the different equations. We consider Centers Of Mass, COM, to be the centroid of particles based on their position. In several steps the information is first transmitted to the COMs, computing their position and mass.

1.2.3 Problems

The previous equations are generic and describe the behavior of SPH method. In order to check our

Sod shock tube

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

$$(\rho, v, p)_{t=0} = \begin{cases} (1.0, 0.0, 1.0) & \text{if} \quad 0 < x \leq 0.5 \\ (0.125, 0.0, 0.1) & \text{if} \quad 0.5 < x < 1.0 \end{cases} \tag{1.21}$$

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

$$P(\rho, u) = (\Gamma - 1)\rho u \tag{1.22}$$

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 Fig. 1.3 and shows physically correct results.

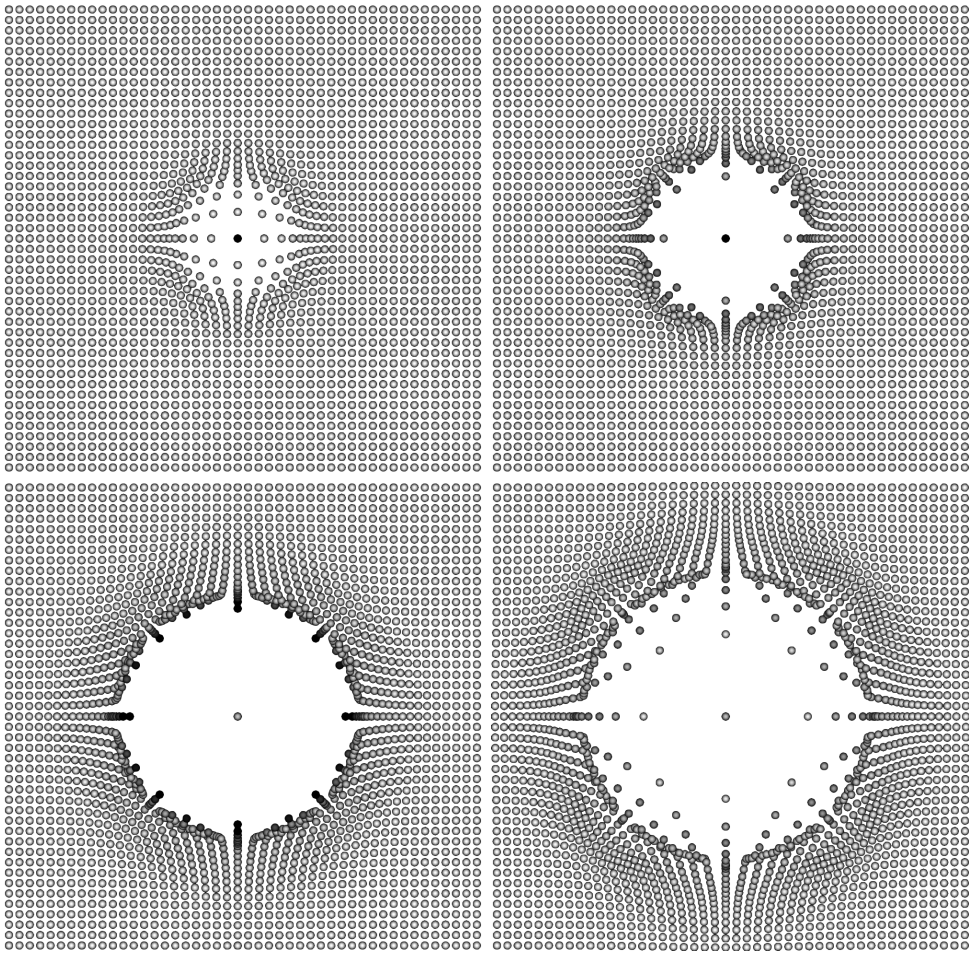

Figure 1.4: Sedov Blast Wave with FleCSPH at respectively $t = 0.01$, $t = 0.03$, $t = 0.06$ and $t = 0.1$

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 Fig. 1.4 and shows physically correct results.

Fluid flow

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:

$$P = B\Big[\big(\frac{\rho}{\rho_0}\big)^{\gamma} - 1\Big] \tag{1.23}$$

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

For this experiment, realistic boundaries conditions were needed. Several methods are possible with SPH we focused on the main ones, the mirror particles [LP91] or the dummies particles

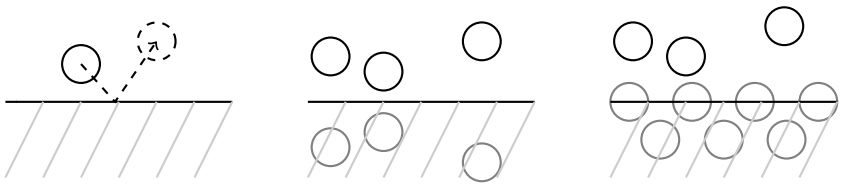

Figure 1.5: Boundaries conditions with respectively: mathematics, mirror and dummies particles

implementation [AHA12].

For the current implementation, we used the dummies particles method. 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.

Astrophysics: neutron stars coalescence

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].

Solving Lane-Emden Equation We need to determine the density function based on the radius.

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

$$\frac{d^2\theta}{d\xi^2} + \frac{2}{\xi}\frac{d\theta}{d\xi} + \theta^n = 0 \tag{1.24}$$

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 simulation.

For $n = 1$ the solution of equation 1.24 is:

$$\theta(\xi) = \frac{sin(\xi)}{\xi} \tag{1.25}$$

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

$$\theta(\xi) = \Big(\frac{\rho(\xi)}{\rho_c}\Big)^{\frac{1}{n}} = \frac{\rho(\xi)}{\rho_c} \tag{1.26}$$

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

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

With K a proportionality constant.

From the previous equations we can write the stellar radius R as:

$$R = \sqrt{\frac{K(n+1)}{4\pi G}}\rho_c^{(1-n)/2}\xi_1 = \sqrt{\frac{K}{2\pi G}} \times \xi_1 \tag{1.27}$$

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

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

$$K = \frac{R^2 2\pi G}{\xi_1^2} \tag{1.28}$$

	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

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 $R = M = G = 1$: $K' = K/(R^2G)$, $m_i' = m_i/M$, $h_i' = h_i/R$, $\vec{x_i}' = \vec{x_i}/R$

1.3 Conclusion

Chapter 2

FleCSPH

2.1 Introduction

2.2 LANL and FleCSI

FleCSI [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 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.

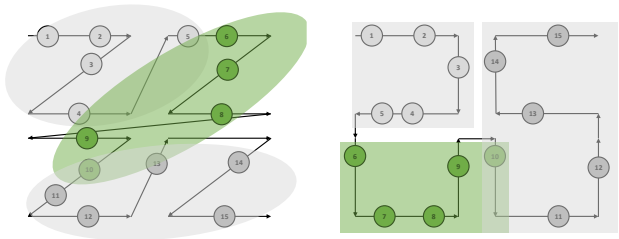

Figure 2.1: Morton and Hilbert space filling curves

2.3 FleCSPH implementation

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 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 Fig. 2.1. 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 Fig. 2.1, 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

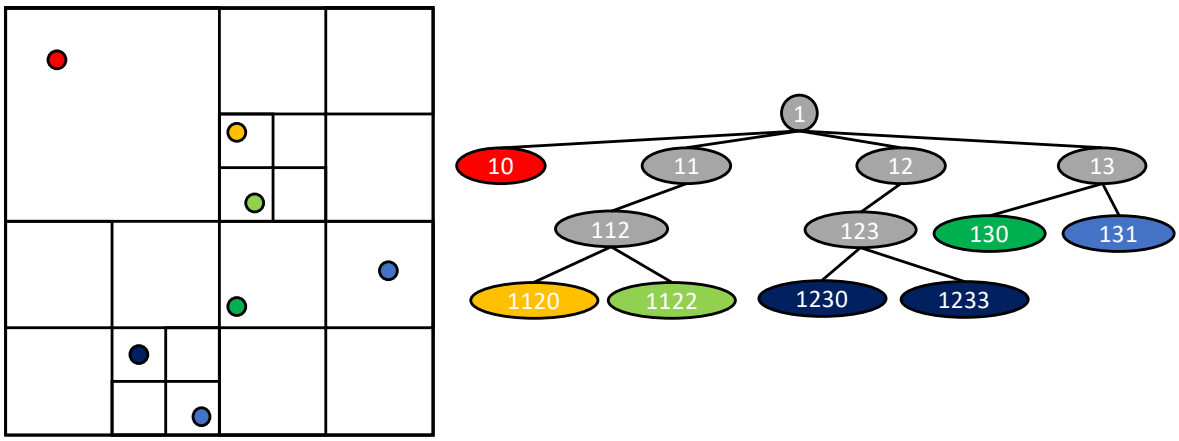

Figure 2.2: Quadtree, space and data representation

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 $2^{-64} \approx$ 1e+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 Fig. 2.2 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 Fig. 2.2.

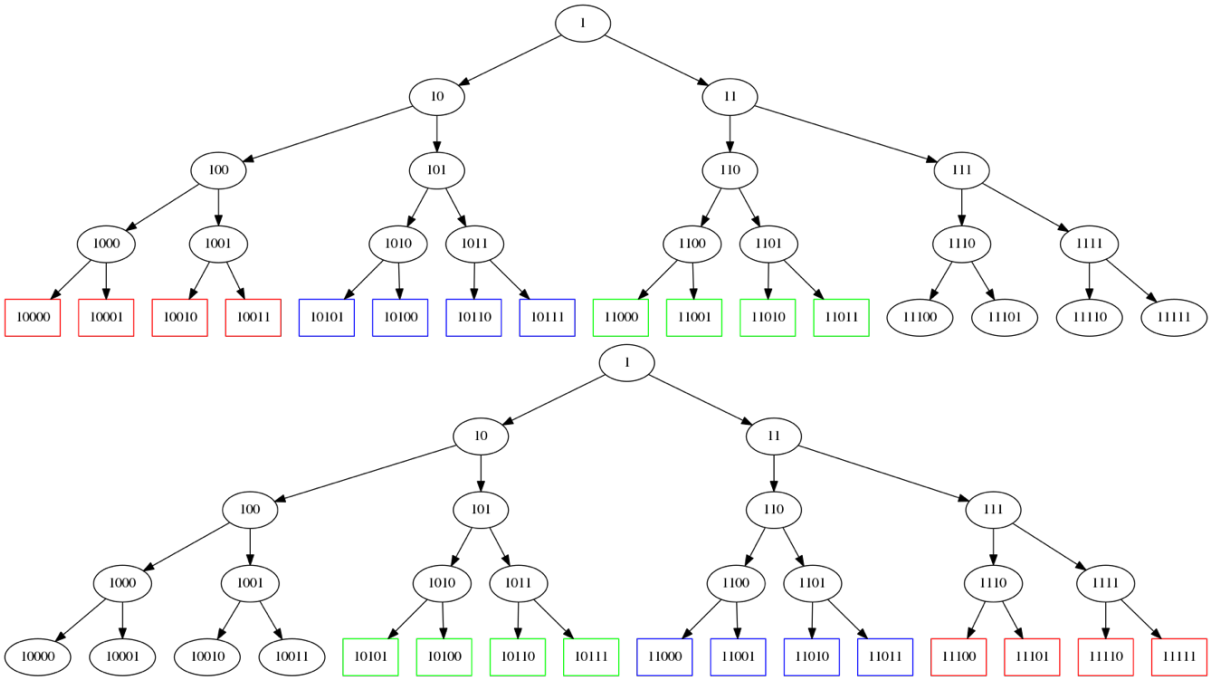

Figure 2.3: Binaries tree for a 2 processes system. Exclusive, Shared and Ghosts particles resp. red, blue, green.

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.

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. In this way we will not need a full tree traversal for each particle but a full tree traversal for every group of particles.

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

Algorithm 2 Tree search algorithm

```
1: procedure FIND_NODES
2:     stack stk ← root
3:     while not_empty(stk) do
4:         branch b ← 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 ← n
19:    while not_empty(stk) do
20:        branch b ← 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 ← root
35:    node-list nl ← ∅
36:    while not_empty(stk) do
37:        branch b ← stk.pop()
38:        if b is leaf then
39:            for each particles p of b do
40:                if within() then
41:                    nl ← 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 ← 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
```

are able to sort them using the keys and, in case of collision keys, using their unique ID. This 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 Fig. 2.3.

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 usage. The first requirement is to allow the user to work directly with the Paraview visualization tool and splash[1] [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.

2.4 Conclusion

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

Conclusion

In this part we presented the

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.