VIII International Conference on Particle-Based Methods
PARTICLES 2023
U. Perego, M. Cremonesi, A. Franci (Eds)

GPUS BASED MATERIAL POINT METHOD FOR COMPRESSIBLE FLOWS

PAOLO J. BAIONI[1], TOMMASO BENACCHIO[2], LUIGI CAPONE[3] AND CARLO DE FALCO[4]

[1,4] MOX, Department of Mathematics, Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano, Italy
e-mail: {paolojoseph.baioni, carlo.defalco}@polimi.it, https://mox.polimi.it/

[2] Weather Research, Danish Meteorological Institute
Lyngbyvej 100, 2100 Copenhagen, Denmark
email: tbo@dmi.dk, www.dmi.dk

[3] Leonardo Labs, Leonardo S.p.A.
Torre, Via Raffaele Pieragostini, 80, 16149 Genova, Italy
email: luigi.capone@leonardo.com, www.leonardo.com

Abstract: *Particle-In-Cell (PIC) methods such as the Material Point Method (MPM) can be cast in formulations suitable to the requirements of data locality and fine-grained parallelism of modern hardware accelerators such as Graphics Processing Units (GPUs). While continuum mechanics simulations have already shown the capabilities of MPM on a wide range of phenomena, the use of the method in compressible gas dynamics is less frequent. This contribution aims to show the potential of a GPU-based MPM parallel implementation for compressible fluid dynamics, as well as to assess the reliability of this approach in reproducing supersonic gas flows against solid obstacles. The results in the paper represent a stepping stone towards a highly parallel, Multi-GPU, MPM-base solver for* $Mach > 1$ *Fluid-Structure Interaction problems.*

Keywords. Material Point Method, GPU, Compressible Flows

1 INTRODUCTION

Parallel computing is a key capability for effective advancement of industrial-grade, large-scale computations, particularly in the fields of compressible computational fluid dynamics and and Fluid-Structure Interaction (FSI) [1]. For these applications, numerical Particle-In-Cell (PIC) methods such as the Material Point Method (MPM) [2, 3] are becoming increasingly relevant, also thanks to the significant growth of computing capabilities made available by modern High Performance Computing architectures [4, 5].
In MPM, continuum media are discretized by means of Lagrangian particles, the *material points*, each one storing every state variable evaluated at its position, enabling the description of a wide range of materials [6, 7, 8] as well as large deformations [9, 10, 11] and fracture phenomena [12]. Notwithstanding its Lagrangian character, MPM also employs a background Cartesian grid to compute differential quantities and solve the motion

equation, thus mediating the particle-particle interactions, taking advantage of both Eulerian and Lagrangian approaches. These features make it an interesting candidate for the development of a highly parallel, Graphics Processing Units (GPUs)-based FSI solver. Indeed, in order to reach high efficiency levels, GPU-based solvers require the algorithm to satisfy data locality, and to be formulated in a parallel way, up to a very fine level of discretization. MPM implementations can naturally achieves these, since: (a) every GPU thread can manage up to very few grid cells; (b) the link between cells and particles can be easily built thanks to the structured character of the grid; and (c) accuracy in the discretization of continuum bodies is obtained by means of the Lagrangian particles, so that the grid does not have to match their boundary.

MPM was first developed from the Fluid Implicit Particle Method [13], a PIC method [14] modified to treat history-dependent materials. Along the years, various enhancements of the method were proposed, solving some of its drawbacks such as grid-crossing error [15] and volumetric locking [16], and improving its accuracy and conservation properties. Among the many, to address grid crossing instability, variants of the method were proposed, such as GIMP [17], CPDI [18], iMPM [19] and IGA-MPM [20], and numerical analysis studies on the use of B-Splines as basis functions in MPM were conduced [21]. An affine and a polynomial mapping was later added to the B-Splines MPM [22, 23] resulting in an angular momentum-conserving model. Other studies explored least squares techniques [24], enabling the recovery of the affine/polynomial splines MPM [25], and the effect of spatial and temporal discretization errors, as well as the application of symplectic integrators [26]. In light of these developments, MPM represents an effective method for the simulation of a wide range of materials and phenomena.

Even though in the literature there are recent GPUs MPM implementations for computer graphics applications [27, 28], such implementations have not yet been applied to compressible fluid dynamics; in fact, when addressing gas dynamics and solids mechanics, MPM is used only for the solids, while the compressible fluid dynamic problem is solved through a second order WENO scheme; moreover, such implementation is run on CPU [29]. There exist also some studies on MPM for compressible flows, as [30, 31], which focus on modelling and expand the method application field, and as [32], which considers the numerical aspects of the topic. Nonetheless, to the best of the authors' knowledge, there are still no studies dedicated to the GPU based, High Performance Computing aspects of MPM implementations for compressible fluid dynamics problems. This contribution aims at illustrating the choices made in the design of a proof-of-concept MPM code to solve compressible gas dynamics problems in the supersonic regime, focusing on computational efficiency and addressing the challenge of optimizing and designing GPUs-based software for modern HPC architectures.

The paper is structured as follows. Sections 2 and 3 recall the analytical and numerical formulation of the method. Sections 4 and 5 show and motivate the chosen algorithm and the current and planned implementation choices. Section 6 illustrates the results obtained against standard test cases, and the final section 7 draws the conclusions of the performed work and prospects the future developments.

2 ANALYTICAL FORMULATION

We are interested in simulating supersonic gas flows, which can be modelled as inviscid fluid in a domain Ω governed by the compressible Euler equations [33]:

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) &= 0 \\ \frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} + p\mathbb{I}) &= 0 \\ \frac{\partial E}{\partial t} + \nabla \cdot ((E + p)\mathbf{v}) &= 0 \end{aligned} \tag{1}$$

where ρ is the density, $\mathbf{v}$ the velocity, p the pressure and E the total energy. In standard MPM, where the total number of Lagrangian material points is kept constant, mass conservation is automatically satisfied by the discretization of continua through such particles; moreover, modelling a perfect gas, energy conservation equation is substituted by the state equation [30, 31]:

$$p = (\gamma - 1)\rho e \tag{2}$$

where e is the specific internal energy and $\gamma = c_p/c_v = 1.4$ for a bi-atomic gas.
By standard arguments, the motion equation in (1) can be re-formulated in the Lagrangian framework as:

$$\rho \dot{\mathbf{v}} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{b} \tag{3}$$

and written in weak form [2, 31]:

$$\int_\Omega \rho \dot{\mathbf{v}} \cdot \mathbf{w} d\Omega = - \int_\Omega \rho \boldsymbol{\sigma} : \nabla \mathbf{w} d\Omega + \int_{\partial\Omega} \rho \boldsymbol{\tau} \cdot \mathbf{w} d\Gamma + \int_\Omega \rho \mathbf{b} \cdot \mathbf{w} d\Omega \tag{4}$$

3 NUMERICAL FORMULATION

As in standard MPM, in order to solve numerically the motion equation, we take a dual Eulerian-Lagrangian approach using both a computational grid and a set of particles. On the Eulerian side, we use a $\mathbb{Q}_1$ discretization, with Lagrangian hat functions, linear in 1D and bilinear in 2D, as basis of the solution and test functions spaces. Therefore, we get the following expressions for mappings among grid and particles [2], where we denote with capital letters the Lagrangian particles variables, with lowercase letters the Eulerian grid ones. For the 2D case we have:

$$\begin{aligned} a = \mathrm{P2G}(A; X, Y) &:= \left[\sum_{k=0}^{Np-1} A_k \; u_i(X_k, Y_k) \right] \\ A = \mathrm{G2P}(a; X, Y) &:= \left[\sum_{i=0}^{(nx+1)(ny+1)-1} a_i \; u_i(X_k, Y_k) \right] \end{aligned} \tag{5}$$

$$\nabla a = \mathrm{P2GD}(A; X, Y) := \left[\sum_{k=0}^{Np-1} A_k \ \nabla u_i(X_k, Y_k)\right] \tag{6}$$
$$\nabla A = \mathrm{G2PD}(a; X, Y) := \left[\sum_{i=0}^{(nx+1)(ny+1)-1} a_i \nabla u_i(X_k, Y_K)\right]$$

where P2G is an acronym for Particles-To-Grid mappings, G2P Grid-To-Particles mappings and D stands for derivative, $u_i(X_k, Y_k)$ denotes the basis function centred on the $i-th$ grid node and evaluated in the $k-th$ particle position, a is a generic grid variable, A is its particle counterpart, nx, ny are the cell numbers in the x and y direction.[1]
In order to model the flow around obstacles, we subtract the inward component of linear momentum and force from the grid nodes whose cell is marked as an obstacle cell, slightly modifying the method first proposed by [35] for granular material, as follows:

$$(m\mathbf{v}_i^o)' = m\mathbf{v}_i^o - \frac{m\mathbf{v}_i^o \cdot \mathbf{n}_i^o - |m\mathbf{v}_i^o \cdot \mathbf{n}_i^o|}{2}\mathbf{n}_i^o, \qquad (\mathbf{f}_i^o)' = \mathbf{f}_i^o - \frac{\mathbf{f}_i^o \cdot \mathbf{n}_i^o - |\mathbf{f}_i^o \cdot \mathbf{n}_i^o|}{2}\mathbf{n}_i^o \tag{7}$$

where the subscript denotes the $i-th$ grid node, the superscript o stands for obstacle and $\mathbf{n}$ is the normal unit vector, positive if directed outward from the obstacle.

4 ALGORITHM DESCRIPTION

Being interested in modelling supersonic flows with MPM on highly parallel architectures, we started from existing MPM algorithm descriptions [36, 37], ported it to CUDA-C and tested the implementation on the Sod shock tube [38] problem (see section 6 for test cases).
In so doing, we also had the chance to make some changes to the algorithm, so as to adapt it to GPUs architecture, see algorithm 1. Specifically, we postponed particles movement to the end of the time loop, in order not to have to store the basis function evaluated at previous particle positions, and computing them on the fly instead - memory fetches are more costly than arithmetic operations on modern computing hardware. Moreover, we turned all the loops involving grid or particle operations into loops on grid cells. In case of particle-based loops, these were turned into inner loops on each cell's particles. This is important to guarantee data locality, especially in fast dynamics fluid simulations, and will be complemented with a proper particles sorting currently under development. The need of keeping data representing geometrically close quantities close in memory becomes particularly evident with a view to the development of a Multi-GPU code based on domain decomposition techniques. With this approach, it will be possible to distribute to different GPUs portions of domain together with the particles contained therein. Finally, we modified the step 2 in [37], where essential and natural boundary conditions are applied to the grid nodes, also adding the obstacle treatment as anticipated in section 3.

[1]It is also possible to derive the same formulation adopting the standard Finite Element procedure, but considering the material points as quadrature points [34].

Algorithm 1 Time loop

```
1: while $t < t_f$ do
2:     Grid reset
3:     Mass, momentum and force P2G: $m = \mathrm{P2G}(M)$, $m\mathbf{v} = \mathrm{P2G}(M\mathbf{V})$, $\mathbf{f} = \mathrm{P2GD}(P)$
4:     Boundary conditions enforcement on $m\mathbf{v}$, $\mathbf{f}$ & obstacle treatment as in (7)
5:     Momentum equation solution on the grid: $(m\mathbf{v})' = m\mathbf{v} + dt\mathbf{f}$
6:     G2P mapping & velocity update: $\overline{\mathbf{V}} = \mathrm{G2P}(\mathbf{v})$, $\mathbf{A} = \mathrm{G2P}(\mathbf{a})$, $\mathbf{V}' = \mathbf{V} + dt * \mathbf{A}$
7:     Momentum P2G & grid velocity computation: $m\mathbf{v} = \mathrm{P2G}(M\mathbf{V})$, $\mathbf{v} = \frac{m\mathbf{v}}{m}$
8:     Boundary condition enforcement on $m\mathbf{v}$, $\mathbf{v}$
9:     Particles properties update:
         9a: $\nabla\mathbf{V} = \mathrm{G2PD}(\mathbf{v})$
         9b: $E' = E + dt\frac{-P}{\rho}\nabla\cdot\mathbf{V}$
         9c: $\rho' = \frac{\rho}{1+dt\nabla\cdot\mathbf{V}}$
         9d: $P' = (\gamma-1)\rho' E'$
10:    Particles moving $\mathbf{X}' = \mathbf{X} + dt\overline{\mathbf{V}}$
11:    CFL condition check & time advance
12: end while
```

5 IMPLEMENTATION

First, a CUDA-C implementation of an existing algorithm for compressible flows [30] was carried out. This language was chosen because of its known performance capabilities, especially on our target architecture, based on NVIDIA A100 GPUs [39]. The implementation strategy has consisted in defining two structures, one for the grid and one for material points, holding the respective variables, with a Structure of Array (SoA) approach, and with member methods decorated as both host and device functions, in order to be able to run them on both CPU and GPU. At the beginning of the main program, a grid variable and a material points variable are constructed on the CPU, and data is copied to a pair of pointers to grid and material points on the GPU. For each step of the algorithm, a CUDA kernel was written and decorated as global only, so that the algorithm executes only on GPU, employing its global memory. These CUDA kernels are essentially as in [37], but without for loops, since they are launched by one CUDA thread per each grid node or particle. For example, the third step is modified from:

```
void step3 (Grid& grd){
 for(idx_t inode = 0; inode < grd.nn; ++inode)
 grd.mv[inode] += grd.f[inode]*dt;}
```

to:

```
__global__ void step3(Grid* grd){
 idx_t inode = blockIdx.x * blockDim.x + threadIdx.x;
 if(inode > NC) return;
 else {grd->mv[inode] += grd->f[inode]*dt;}}
```

where NC denotes the total number of cells. Moreover, when a variable is read and modified by multiple cores simultaneously, atomic operations are performed, so, for example in mass P2G we modified:

```
for(idx_t mpind=0; mpind<mps.n; ++mpind){
 grd.m[mps.inode[mpind]] += mps.m[mpind]*mps.N1[mpind];
 grd.m[mps.inode[mpind]+1] += mps.m[mpind]*mps.N2[mpind];
}
```

into:

```
idx_t mpind = blockIdx.x * blockDim.x + threadIdx.x;
if (mpind>=NC*MPPC) return;
else {
 atomicAdd(&(grd->m[mps->inode[mpind]]),mps->m[mpind]*mps->N1[mpind]);
 atomicAdd(&(grd->m[mps->inode[mpind]+1]),mps->m[mpind]*mps->N2[mpind]);
}
```

where MPPC denotes the number of initial material points per cell. At the end of the time loop, updated data are copied back to the CPU grid variables and material points variables, their GPU counterparts are freed and the final output is written to file by the CPU. By doing so we minimize host-device memory transfers, obtaining a significant speed-up (section 6).
Second, the code has been re-implemented from scratch to address supersonic 2D test cases, using C++17 and its Standard Template Library (STL)-based parallelism for performance portability reasons, since they can be used both on CPUs and GPUs [40]. In this implementation, every kernel operation has been rewritten relying on standard algorithms, passing the parallel vectorized execution policy whenever possible. Regarding the data structure, from recent developments on GPUs computing literature [28, 5], it emerges that the best performance is achieved by adopting an Array of SoA. However, for simplicity, the SoA approach has been kept for the time being, leaving AoSoA for future work. In fact, the current data structure gives a conceptually straightforward extension towards multi-GPU implementations, where a further level of parallelization through an MPI-based domain decomposition will make each GPU hold a SoA as the current one, giving rise globally to an AoSoA data structure.
Finally, a remarkable performance gain has been obtained swapping the P2G and G2P kernels inner loops, looping first on every variable, and then on each grid cell and its inner particles, in order to guarantee better data locality, given the SoA structure.

6 RESULTS

This section contains numerical results obtained with the MPM implementations described in the previous section. First, the efficiency of the GPU-based MPM code for compressible flows is verified in 1D. Second, the STL-based 2D implementation is tested on a supersonic flow case and the results compared with those obtained with an established finite element library.

6.1 Sod shock tube

The first gas dynamics test case is the 1D Riemann problem known as Sod shock tube [38, 41], where two fluids initially at rest are divided by a diaphragm, which is removed at $t = 0$, determining the formation of several discontinuities. Indeed, while the two initial zones shrink progressively towards the domain boundaries, these phenomena are expected:

- a rarefaction wave propagates toward the high density zone, leading to continuity of solutions with discontinuity of derivatives;
- in the position where the diaphragm was, which now is moving towards low densities, a contact discontinuity develops, leading to continuity of pressure and velocity plus discontinuity of density and internal energy density;
- a shock wave propagates towards low density zones, leading to discontinuity of all involved quantities.

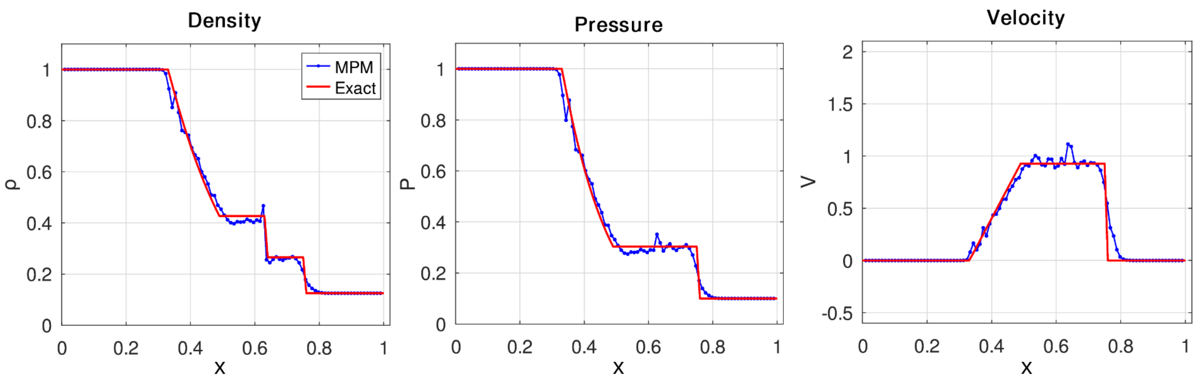

Figure 1: Sod shock tube - 100 cells, 3 material points per cells; exact solution from [41].

While some oscillations in the numerical solution are visible close to the discontinuities, the solution quality is deemed acceptable given the current stage of the implementation, and are in accordance with those in [36, 34].
In addition, comparison of the execution time versus the increasing spatial resolution for the CUDA-C code and the serial C++ code shows that, once the computational effort is relevant enough to overcome the initial cost of moving data to the GPU, the GPU MPM code can be significantly faster than the CPU code (Fig. 2).
Specifically, defining 17 cell numbers as `ncells = 100*(2.^[0:16])`, thus log-scale evenly distributed between 100 and 100×2^{16}, it can be seen that the execution time on CPU and GPU scales linearly with the number of cells, even though the GPU implementation requires an additional initialization time, which is about 10 times the total wall-clock time of the reference CPU run with 100 grid cells. Nonetheless, as resolution increases and computational effort grows, the fine-grained parallelism of MPM makes the GPU

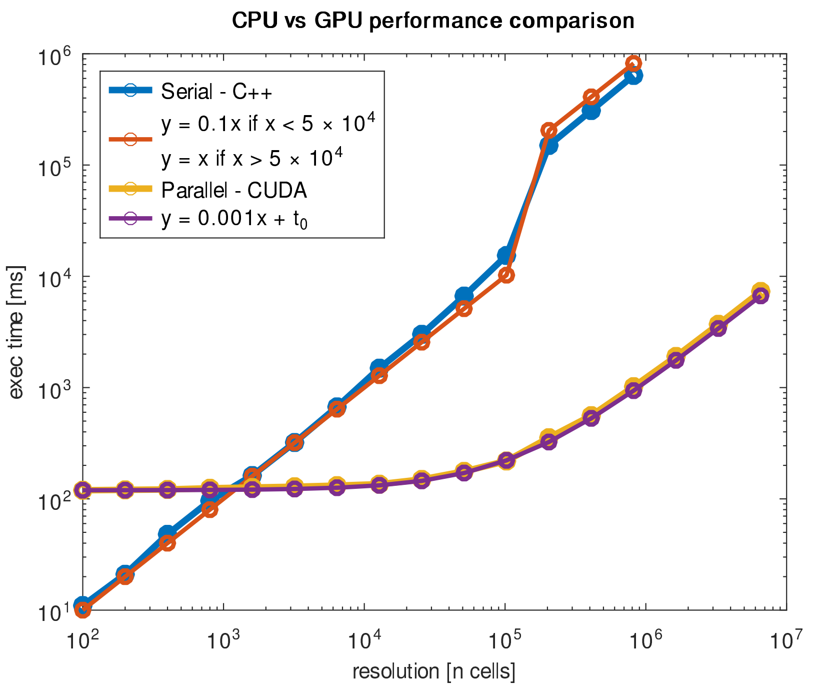

Figure 2: Sod shock tube test, execution time for the 1D MPM code as a function of resolution. Blue and yellow: actual data points. Orange and purple: linear functions represented to ease comparison

implementation faster than the serial CPU implementation by a factor of $100\times$, and even $1000\times$ from 51200 cells on, as the L3 cache of the CPU saturates.

6.2 Supersonic flow past a cylinder and over a rectangular step

Two 2D test cases are shown, both regarding supersonic flows against an obstacle. Denoting with the ∞ subscript the unperturbed conditions, we have density $\rho_\infty = 1.4$, pressure $p_\infty = 1$, specific heat ratio $\gamma = 1.4$, so that initial equilibrium sound speed is $c_{s,\infty} = 1$, velocity $\mathbf{v} = (3,0)$, and thus $Mach_\infty = \frac{v_\infty}{c_{s,\infty}} = 3$ (same units of Sod's problem [38]). Boundary conditions are wind tunnel-like, with slip conditions applied at the horizontal walls, Dirichlet conditions at the inflow boundary and no enforced condition at the outflow boundary - particles leaving the domain from the outflow boundary are re-inserted in a random cell at the inflow boundary with initial, undisturbed conditions.
The first test (Fig. 6.2) concerns the flow in the domain $[0,4]\times[0,2]$ past a cylinder whose cross section is a circle of radius 0.25 centred in $(0.6,1)$; the grid is made by 400×200 cells, 4 material points per cell are initially placed in the fluid domain according to a uniform random distribution. As gas particles encounter the cylinder, a shock wave develops at the front and two oblique shocks in the rear; the first shock is reflected by walls and interacts with the back shocks.
In the second test (Fig. 6.2), the Mach 3 flow proceeds against a $[0.6,3]\times[0,0.2]$ rectangular step; the grid is made by 300×100 cells, material points are initially distributed as in the previous case. Also in the step case, a frontal shock develops, and it is later

reflected from the top wall and successively from the step. In both cases results with the

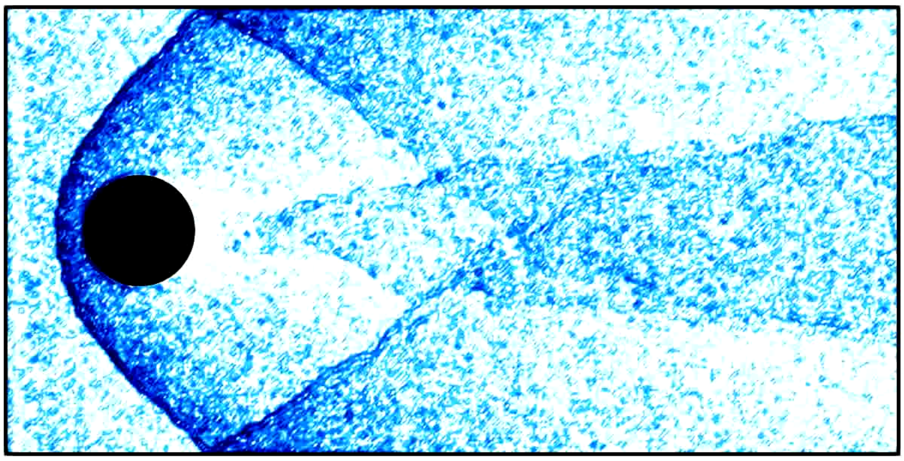

Figure 3: Mach 3 flow past a cylinder, computed log-scale normalized density gradient.

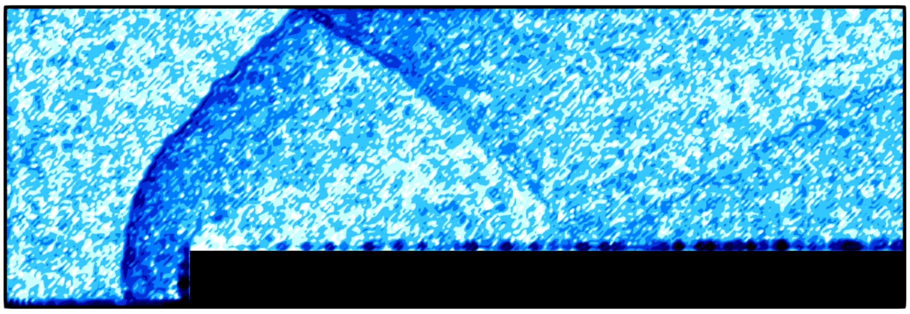

Figure 4: Mach 3 flow past a rectangular step, computed log-scale normalized density gradient.

MPM method are qualitatively comparable with the ones in the literature [33, 42], and the ones obtained with the deal.II finite element library. Specifically, Figure 6.2 compares very well with the result of the step 69 from deal.II tutorial, which uses a comparable resolution to the one used in the MPM simulation.[2]

7 CONCLUSIONS AND NEXT STEPS

This paper has explored the potential of accelerating the Material Point Method on GPU-based hardware via CUDA porting of a standard MPM code for compressible flows. Insight has been given on algorithm adaptations which will provide further benefits, especially when addressing a second level of parallelization through an MPI-based domain decomposition approach on Multi-GPU hardware. Moreover, the simulation of standard test cases of supersonic flow has provided results comparable with the literature and with established software runs with the same polynomial order (first order - Lagrangian hat

[2]https://www.dealii.org/current/doxygen/deal.II/step_69.html#Results.

functions) and number of degrees of freedom. These results show that MPM is able to simulate Mach 3 gas dynamics, giving qualitatively good results with a prototype based on existing implementations [30, 37], and serve as further step towards the design of a highly efficient MPM-based supersonic FSI solver.
On the High Performance Computing side, interesting themes under development concern the porting of the existing code to Multi-GPU architecture via MPI, the realization of scalability studies, and the addition of a CFL-based adaptive time step and a grid-particles adaptive refinement through a quad-tree approach [43, 44]. On the modelling side, more complex test cases will be simulated, eventually leading to a 3D extension and Fluid-Structure Interaction models. In addition, a more accurate higher order B-Splines based method is being developed, and the effect on parallel performance due to related stencil growth and halo overlapping will be the subject of careful investigation.

8 ACKNOWLEDGEMENTS

This research was sponsored by the Leonardo Labs, Leonardo S.p.A., through the PhD thematic scholarship “Multi-GPUs based PIC/MPM methods for Compressible Flows” funded at Politecnico di Milano; the participation to the VIII International Conference on Particle-Based Methods PARTICLES 2023 was funded also by the “Gruppo Nazionale per il Calcolo Scientifico” (GNCS - INdAM). Views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Leonardo Labs or GNCS.

REFERENCES

[1] G. Laskowski, J. Kopriva, V. Michelassi, S. Shankaran, U. Paliath, R. Bhaskaran, Q. Wang, C. Talnikar, Z. Wang, and F. Jia, “Future directions of high fidelity cfd for aerothermal turbomachinery analysis and design,” *46th AIAA Fluid Dynamics Conference*, 2016.

[2] D. Sulsky, Z. Chen, and H. Schreyer, “A particle method for history-dependent materials,” *Computer Methods in Applied Mechanics and Engineering*, vol. 118, no. 1-2, pp. 179–196, 1994.

[3] Y. L. X. Zhang, Z. Chen, *The Material Point Method.* Academic Press, Elsevier, 2017.

[4] F. Hariri, T. Tran, A. Jocksch, E. Lanti, J. Progsch, P. Messmer, S. Brunner, C. Gheller, and L. Villard, “A portable platform for accelerated pic codes and its application to gpus using openacc,” *Computer Physics Communications*, vol. 207, pp. 69–82, 2016.

[5] Y. Fei, Y. Huang, and M. Gao, “Principles towards real-time simulation of material point method on modern gpus,” *CoRR*, vol. abs/2111.00699, 2021.

[6] T. G. Goktekin, A. W. Bargteil, and J. F. O’Brien, “A method for animating viscoelastic fluids,” *ACM Trans. Graph.*, vol. 23, no. 3, p. 463–468, 2004.

[7] A. Stomakhin, C. Schroeder, C. Jiang, L. Chai, J. Teran, and A. Selle, “Augmented mpm for phase-change and varied materials,” *ACM Transactions on Graphics*, vol. 33, no. 4, 2014. cited By 95.

[8] C. Jiang, *The Material Point Method for the Physics-Based Simulation of Solids and Fluid.* PhD thesis, University of California, Los Angeles, 2015.

[9] S. Andersen and L. Andersen, “Modelling of landslides with the material-point method,” *Computational Geosciences*, vol. 14, pp. 137–147, Jan 2010.

[10] Q.-A. Tran and W. Sołowski, "Generalized interpolation material point method modelling of large deformation problems including strain-rate effects – application to penetration and progressive failure problems," *Computers and Geotechnics*, vol. 106, pp. 249–265, 2019.

[11] A. de Vaucorbeil, V. P. Nguyen, and C. R. Hutchinson, "A total-lagrangian material point method for solid mechanics problems involving large deformations," *Computer Methods in Applied Mechanics and Engineering*, vol. 360, p. 112783, 2020.

[12] S. Wang, M. Ding, T. F. Gast, L. Zhu, S. Gagniere, C. Jiang, and J. M. Teran, "Simulation and visualization of ductile fracture with the material point method," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 2, jul 2019.

[13] J. Brackbill and H. Ruppel, "Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions," *Journal of Computational Physics*, vol. 65, no. 2, pp. 314–343, 1986.

[14] M. W. Harlow, Francis H.; Evans and J. Harris, David E., "The particle-in-cell method for two-dimensional hydrodynamic problems," *Report LAMS-2082 of the Los Alamos Scientifc Laboratory*, 1956.

[15] R. Tielen, E. Wobbes, M. Möller, and L. Beuth, "A high order material point method," *Procedia Engineering*, vol. 175, pp. 265–272, 2017. Proceedings of the 1st International Conference on the Material Point Method (MPM 2017).

[16] W. M. Coombs, T. J. Charlton, M. Cortis, and C. E. Augarde, "Overcoming volumetric locking in material point methods," *Computer Methods in Applied Mechanics and Engineering*, vol. 333, pp. 1–21, 2018.

[17] S. G. Bardenhagen and E. M. Kober, "The generalized interpolation material point method," *Computer Modeling in Engineering & Sciences*, vol. 5, no. 6, pp. 477–496, 2004.

[18] A. Sadeghirad, R. M. Brannon, and J. Burghardt, "A convected particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations," *International Journal for Numerical Methods in Engineering*, vol. 86, no. 12, pp. 1435–1456, 2011.

[19] D. Sulsky and M. Gong, *Improving the Material-Point Method*, pp. 217–240. Cham: Springer International Publishing, 2016.

[20] G. Moutsanidis, C. C. Long, and Y. Bazilevs, "Iga-mpm: The isogeometric material point method," *Computer Methods in Applied Mechanics and Engineering*, vol. 372, p. 113346, 2020.

[21] M. Steffen, R. M. Kirby, and M. Berzins, "Analysis and reduction of quadrature errors in the material point method (mpm)," *International Journal for Numerical Methods in Engineering*, vol. 76, no. 6, pp. 922–948, 2008.

[22] C. Jiang, C. Schroeder, and J. Teran, "An angular momentum conserving affine-particle-in-cell method," *Journal of Computational Physics*, vol. 338, pp. 137–164, 2017.

[23] C. Fu, Q. Guo, T. Gast, C. Jiang, and J. Teran, "A polynomial particle-in-cell method," *ACM Trans. Graph.*, vol. 36, nov 2017.

[24] E. Wobbes, M. Möller, V. Galavi, and C. Vuik, "Conservative taylor least squares reconstruction with application to material point methods," *International Journal for Numerical Methods in Engineering*, vol. 117, no. 3, pp. 271–290, 2019.

[25] Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang, "A moving least squares material point method with displacement discontinuity and two-way rigid body coupling," *ACM Transactions on Graphics*, vol. 37, no. 4, p. 150, 2018.

[26] M. Berzins, "Energy conservation and accuracy of some mpm formulations," *Computational Particle Mechanics*, Feb 2022.

[27] M. Gao, X. Wang, K. Wu, A. Pradhana, E. Sifakis, C. Yuksel, and C. Jiang, "Gpu optimization of material point methods," *ACM Trans. Graph.*, vol. 37, dec 2018.

[28] X. Wang, Y. Qiu, S. R. Slattery, Y. Fang, M. Li, S.-C. Zhu, Y. Zhu, M. Tang, D. Manocha, and C. Jiang, "A massively parallel and scalable multi-gpu material point method," *ACM Trans. Graph.*, vol. 39, jul 2020.

[29] Y. Cao, Y. Chen, M. Li, Y. Yang, X. Zhang, M. Aanjaneya, and C. Jiang, "An efficient b-spline lagrangian/eulerian method for compressible flow, shock waves, and fracturing solids," *ACM Trans. Graph.*, vol. 41, may 2022.

[30] A. R. York, D. Sulsky, and H. L. Schreyer, "Fluid–membrane interaction based on the material point method," *International Journal for Numerical Methods in Engineering*, vol. 48, pp. 901–924, 2000.

[31] Y.-C. Su, J. Tao, S. Jiang, Z. Chen, and J.-M. Lu, "Study on the fully coupled thermodynamic fluid–structure interaction with the material point method," *Computational Particle Mechanics*, vol. 7, pp. 225–240, Mar 2020.

[32] L. T. Tran, J. Kim, and M. Berzins, "Solving time-dependent pdes using the material point method, a case study from gas dynamics," *International Journal for Numerical Methods in Fluids*, vol. 62, no. 7, pp. 709–732, 2010.

[33] J.-L. Guermond, M. Nazarov, B. Popov, and I. Tomas, "Second-order invariant domain preserving approximation of the euler equations using convex limiting," *SIAM Journal on Scientific Computing*, vol. 40, no. 5, pp. A3211–A3239, 2018.

[34] A. de Vaucorbeil, V. P. Nguyen, S. Sinaie, and J. Y. Wu, "Chapter two - material point method after 25 years: Theory, implementation, and applications," in *Advances in Applied Mechanics* (S. P. Bordas and D. S. Balint, eds.), vol. 53, pp. 185–398, Elsevier, 2020.

[35] S. Bardenhagen, J. Brackbill, and D. Sulsky, "The material-point method for granular materials," *Computer Methods in Applied Mechanics and Engineering*, vol. 187, no. 3, pp. 529–541, 2000.

[36] A. York, "Development of modifications to the material point method for the simulation of thin membranes, compressible fluids, and their interactions," 1997.

[37] Z. Chen and R. M. Brannon, "An evaluation of the material point method," *Sandia National Lab. Technical Report*, 2 2002.

[38] G. A. Sod, "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws," *Journal of Computational Physics*, vol. 27, no. 1, pp. 1–31, 1978.

[39] M. Breyer, A. Van Craen, and D. Pflüger, "A comparison of sycl, opencl, cuda, and openmp for massively parallel support vector machine classification on multi-vendor hardware," in *International Workshop on OpenCL*, IWOCL'22, (New York, NY, USA), Association for Computing Machinery, 2022.

[40] J. Latt, C. Coreixas, and J. Beny, "Cross-platform programming model for many-core lattice boltzmann simulations," *PLOS ONE*, vol. 16, pp. 1–29, 04 2021.

[41] E. F. Toro, *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer, 2009.

[42] M. Maier and M. Kronbichler, "Efficient parallel 3d computation of the compressible euler equations with an invariant-domain preserving second-order finite-element scheme," *ACM Trans. Parallel Comput.*, vol. 8, sep 2021.

[43] H. Tan and J. A. Nairn, "Hierarchical, adaptive, material point method for dynamic energy release rate calculations," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 19, pp. 2123–2137, 2002.

[44] Y.-J. Cheon and H.-G. Kim, "An adaptive material point method coupled with a phase-field fracture model for brittle materials," *International Journal for Numerical Methods in Engineering*, vol. 120, no. 8, pp. 987–1010, 2019.