An Asynchronous Material Point Method

Yuanming Hu* and Yu Fang*
Tsinghua University
* equal contribution, {yuanmhu,squarefk}@gmail.com

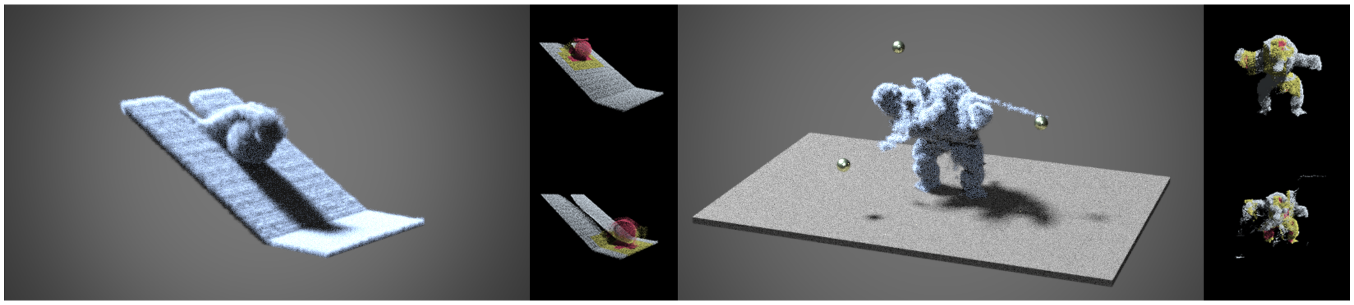

Figure 1: Our Asynchronous Material Point Method (AsyncMPM) leads to efficient simulation of deformable materials. The rolling snowball scene (left) is accelerated by $4.3\times$, while the shooting armadillo scene (right) is accelerated by $6.1\times$. The efficiency is achieved by using a hierachy of different time steps at different regions, so that most particles are not updated unless necessary. Only red and yellow (buffer) particles are involved in updating.

ABSTRACT

We propose a novel asynchronous time integration scheme for the Material Point Method (MPM), which offers temporal adaptivity when objects of different stiffness or velocity coexist. We show via multiple test scenes that our asynchronous MPM leads to $6\times$ speed up over traditional synchronous MPM without sacrificing accuracy.

CCS CONCEPTS

• **Computing methodologies → Physical simulation**;

KEYWORDS

Physically based animation, material point method, asynchronous time integration

ACM Reference format:
Yuanming Hu* and Yu Fang*. 2017. An Asynchronous Material Point Method. In *Proceedings of SIGGRAPH '17 Posters, Los Angeles, CA, USA, July 30 - August 03, 2017,* 2 pages.
DOI: 10.1145/3102163.3102170

1 INTRODUCTION

Since the first adoption of the Material Point Method (MPM) [Stomakhin et al. 2013] in computer graphics, its expressiveness and elegance has drawn much attention from the simulation community. However, efficiency has been an unsolved problem of MPM. Though implicit MPM is generally considered more stable, when combined with large time steps, it usually leads to large numeric errors, resulting in undesired dissipative simulation.

In comparison, explicit MPM is easier to implement and to optimize. Explicit MPM walks around the need for second-order derivatives of potential energy with respect to the deformation gradient, a complex 4-th order tensor required by the implicit version. Also, though fewer iterations are needed in implicit MPM, the sparse system formed in the implicit version is actually quite "dense", containing up to 343 non-zero terms, which severely slow it down. Moreover, to combine multiple materials such as [Klár et al. 2016; Stomakhin et al. 2013], putting together the implicit systems is a non-trivial task. All these considerations underscore the need to better exploiting explicit MPM. To speed it up while retaining its simplicity, we developed an asynchronous version of explicit MPM, named *AsyncMPM*, where temporal adaptivity is offered so that computational resource is focused on stiff regions and higher efficiency is thereby achieved. Two important components of AsyncMPM are partial particle update and the scheduler which determines the particle subset and Δt for updating.

2 PARTIAL PARTICLE UPDATE

The key observation here, is that when using explicit MPM, the time step is limited by stiffest or fastest part from the whole scene, while requiring the rest of the scene to be updated as frequently as these "stiff" parts is wasteful in most simulations. Ideally we should update different regions with different frequency which is the lowest one required for stability and accuracy.

Unlike [Thomaszewski et al. 2008] where a mesh is used, the mesh-free treatment of particles makes partial update in MPM challenging. In AsyncMPM, grids are grouped into larger blocks $B = \{B_1, B_2, ...\}$, and each particle $p \in P$ belongs to exactly one block containing it. Particles contained in block b are denoted as P_b. We only update particles in B_{updating}, a usually very small subset

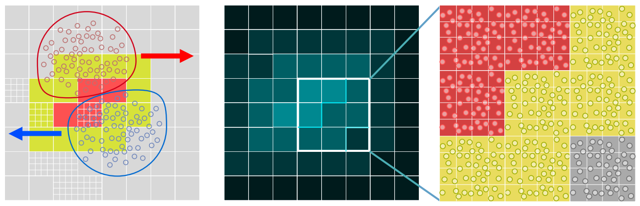

Figure 2: Left: Only (colliding) particles within updating (red) blocks are updated. Particles in surrounding buffer blocks (yellow) are rasterized but not updated. Most particles are not updated (grey), leading to superior efficiency. Middle: darker regions are updated with exponentially lower frequency. Right: particle category is the same as containing block category (updating/buffer/inactive).

of B containing stiff or fast-moving particles. Although particles outside B_{updating} will not be updated, the force of one surrounding B_{updating} will affect those inside. Therefore, it is necessary to also rasterize particles in blocks around them, B_{buffer}, onto the grid. Fig. 2 provides an illustration of different types of blocks and particles. On each particle, we additionally store t_p, standing for the last time when it was updated. When integrating its position or deformation gradient, we use $\Delta t = t - t_p$, and then set $t_p \leftarrow t$.

3 ASYNCHRONOUS SCHEDULING

Central to our asynchronous simulator is the scheduler, which determines frequency of block update adaptively. We use a hierarchy of time steps at different blocks, $\Delta t_b = M_b \times t_\varepsilon$, where M_b is powers-of-two integer multipliers and t_ε is the base time step. Δt_b is constrained by two major factors:

Particle stiffness. Stiffer particles should be given more frequent update for stability. We follow [Thomaszewski et al. 2008] and use

$$p^s = \left\lfloor k_p \middle/ \sqrt{\lambda_p + 2\mu_p} \right\rfloor / t_\varepsilon,$$

where k_p is a constant related to particle volume and λ_p and μ_p are lamé parameters. The block Δt multiplier limited by such stiffness is naturally $M_b^s = \min_{p \in P_b}\{p_s\}$.

Particle relative velocity & "CFL" condition. CFL condition is commonly used for limiting time steps for fluid simulation. However, in MPM, we care more about relative velocity: a snowball in fast transitional motion should not be limited by CFL, while two snowballs colliding with each other should be, so that artifacts like penetration are avoided. Therefore, for each block, the maximum relative velocity within its surrounding $3 \times 3 \times 3$ block neighbourhood $v_{b,\text{relative}}$ is calculated. Additional careful treatment is done near the boundary. Finally, per-block maximum Δt multiplier limited by velocity is simply defined as $M_b^v = \lfloor c_p / v_{b,\text{relative}} \rfloor / t_\varepsilon$. In summary, the block Δt multiplier is $M_b = \min\{M_b^s, M_b^v\}$.

When neighbouring blocks has too different M_b, the numeric error will be out of control. Therefore, we limit M_b of neighbouring blocks to be graded. Looping over all particles and calculate M_b will clearly make AsyncMPM lose its efficiency which comes from partial update, so we only do it incrementally on active blocks. The whole AsyncMPM cycle is shown in Algorithm. 1.

Algorithm 1: Outline of the AsyncMPM Algorithm.

```
while $t < t_{final}$ do
    1. Max. allowed $\Delta t$ multiplier $M' = POT(\min_{b \in B}\{M_b\})$
       $//POT(n)$ stands for the largest power of two below $n$;
    2. Actual time step multiplier $M = M' - T \mod M'$;
    3. $T \rightarrow T + M, t = T t_\varepsilon$;
    4. $B_{\text{updating}} = \{T \mod M_b = 0 | b \in B\}$;
    5. $B_{\text{buffer}}$ = blocks surrounding $B_{\text{updating}}$;
    6. Rasterize $p \in P_b, b \in B_{\text{updating}} \cap B_{\text{buffer}}$;
    7. Update each $p \in P_b, b \in B_{\text{updating}}$ by $\Delta t = t - t_p$
       (position, deform. grad.), and set $t_p \leftarrow t$ (with boundary
       cond.);
    8. Update $M_b, b \in B_{\text{updating}} \cap B_{\text{buffer}}$;
```

4 IMPLEMENTATION AND RESULTS

We use grid blocks of size 8^2 (2D) or 8^3 (3D). In each block, an array storing pointers to its particles is used to maintain P_b. Our code is implemented in C++ based on the open-source library *Taichi*[Hu 2017]. AsyncMPM is now part of *Taichi* and the code is freely available at https://github.com/yuanming-hu/taichi/tree/ampm.

We implemented our novel method and its traditional synchronous version in both 2D and 3D, and compared our method with its traditional synchronous counter part. We also implemented implicit MPM in 2D, and found an careful implementation of explicit MPM combined with APIC [Jiang et al. 2015] leads to even faster simulation in our settings, therefore we omit the comparison with 3D implicit MPM. Various test scenes are evaluated on to demonstrate the speed up. **Simulation results are demonstrated in the supplemental video.** 1.9 – 6.1× speed up over traditional asynchronous MPM is observed, as listed in Table 1. All experiments were done on an Intel i5-4278U (2.6GHz) CPU and 8GB of memory.

Scene	Grid	Particles	Sync.	Async.	speed up
Rolling snowball	320×180	19,222	6.9s	3.5s	1.9×
Bullet into sponge	256×256	111,036	11.2s	3.7s	3.0×
Rolling snowball	$255 \times 255 \times 255$	496,790	652.3s	153.2s	4.3×
Shooting armadillo	$255 \times 255 \times 255$	141,135	125.3s	20.6s	6.1×
Bunny smashing	$127 \times 127 \times 127$	33,553	26.6s	8.6s	3.1×
Snowball collision	$255 \times 255 \times 255$	143,801	526.2s	145.8s	3.6×

Table 1: Settings and time per frame of our test scenes.

REFERENCES

Yuanming Hu. 2017. Taichi - Physically based computer graphics library, http://taichi.graphics. (2017).

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 51.

Gergely Klár, Theodore Gast, Andre Pradhana, Chuyuan Fu, Craig Schroeder, Chenfanfu Jiang, and Joseph Teran. 2016. Drucker-prager elastoplasticity for sand animation. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 103.

Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. 2013. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 102.

Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straßer. 2008. Asynchronous cloth simulation. In *Computer Graphics International*, Vol. 2.