Author responses to the Interactive discussion on "An explicit GPU-based material point method solver for elastoplastic problems (ep2-3De v1.0)" in the Geoscientific Model Development (GMD) Journal

Emmanuel Wyser, Yury Alkhimenkov, Michel Jaboyedoff, Yury Podladchikov

August 28, 2021

The referee comments appear in black, whereas our responses appear in blue and the changes made in the revised manuscript appear in red.

1 Reviewer #1

The authors implemented an explicit GPU-based solver within the material point method framework and tested using two- and three-dimensional problems. Results seem to agree with the expected values validating this MPM - GPU architecture. I would like to suggest the publication of this work. Nonetheless, some minor points should be addressed first.

We would like to acknowledge the reviewer for the time spent on the revision of our work.

Comment # 1 The authors mentioned that this GPU architecture speeds up information transfer between nodes and material points. As stated, this is one of the most computationally expensive operations in MPM. Nevertheless, finding material points new location after the mesh returns to its original position is another process that is computationally expensive (in many cases more expensive than information transfer between nodes and material points). I would like to know if the GPU architecture proposed also improves this step. If yes, the author could indicate it in the paper. If not, it would be interesting if the author discusses the possibility of combining some techniques (e.g. Pruijn N.S. 2016) together with GPU's to improve MPM computations.

Pruijn N.S. 2016. The improvement of the material point method by increasing efficiency and accuracy. TU Delft Master Thesis.

Reply # 1 The reviewer points out the computationally expansive operation of finding material point's new locations after the mesh has been reset at the end of a time step. We used a regular background mesh (as opposed to triangular mesh or non-constant element size), therefore, it is straightforward to find material point's new location. To find in which element e a material point p is located, we use the following equation

$$e = (\texttt{floor}((z_p - z_{\min})/\Delta z) + n_{el,z} \times \texttt{floor}((x_p - x_{\min})/\Delta x)) + n_{el,x} \times n_{el,z}\texttt{floor}((y_p - y_{\min})/\Delta y), \quad (1)$$

where $n_{el,x}$ and $n_{el,z}$ are the number of elements along $x-$ and $z-$directions, $x_{\min}$, $y_{\min}$ and $z_{\min}$ are the minimum x, y, and z coordinate of nodes. However, this is far less trivial when irregular background mesh is used and such equation can not be used any more. Such concern is out of the scope of our contribution since our implementation only consider a regular background mesh. This concern also explain why we selected a regular background mesh.

We looked at the reference suggested by the reviewer. From an overlook of the mentioned research work, we think a GPU-based implementation of the brute force method is possible, but this would require deeper

investigations. This could be the subject of future studies. We strongly think such method could benefit from the computational power of GPUs.

Change # 1 -

Comment # 2 The authors include a damping value D in the simulations. This value is not well validated. It seems that several simulations were needed to find it, giving the idea that D is not related to the material properties and geometry and is more of an artificial way to reach the desirable results. The authors should elaborate better on the reasons for using this specific damping value.

Reply # 2 It is true that the damping value is not well validated. From a broader perspective and to our knowledge, no studies thoroughly investigated and quantified the influence of damping. However, the common range between 0.05 and 0.15 is usually selected by researchers (e.g., Wang et al., 2016b; Wang et al., 2016a) for dynamic analysis. This range was found sufficient to damp out dynamic oscillations while not producing spurious plastic yielding or an over-damped system, as mentioned in Wang et al., 2016a. As such, we decided to use a damping value of $D = 0.1$, since reasonable propagations were obtained and no spurious plastic yielding were noticed.

As raised by the reviewer, we will elaborate better on the reasons of this specific value and will clarify accordingly the lack of validation of this damping value within the main body of the text. We will also mention the need for future studies addressing this concern.

Change # 2 -

Comment # 3 In line 32, the authors mentioned that "The background mesh can be reset". As far as I know, the background mesh must be reset. I recommend changing the verb "can" for a better one.

Reply # 3 We agree with the reviewer. We will change the verb in the revised manuscript.

Change # 3 L.32 The background mesh is reset

Comment # 4 I am wondering if the variables in line 75 are the same as in equations 2 and 3 since different punctuations were used (e.g. Å≪ and û).

Reply # 4 We recognize here that this is a mathematical typo. Variables in line 75 are the same as in Eqs. 2 and 3. We will fix this in the revised version of the manuscript.

Change # 4 (L.75) $\hat{u} \rightarrow \bar{u}$ and $\hat{\tau} \rightarrow \bar{\tau}$

Comment # 5 Finally, I recommend reading the paper again to correct some typos detected..

Reply # 5 We acknowledge the reviewer's recommendation and we will read the manuscript again to correct remaining typos

Change # 5 -

Reply # 6 We additionally extended the original single GPU code and implemented a multi-GPU version using the message passing interface standard. We provide the new section below, which then will be included (with some simplifications) into the manuscripts during the revision stage.

The Multi-GPU Code Implementation

Introduction

One of the major limitation of `ep2-3De v1.0` is the on-chip memory. We demonstrated that an implementation of the material point framework quickly reaches the hardware limit of GPUs, even on modern architectures. It is then essential to overcome this limit in order to resolve larger computational domain with a greater amount of material points.

Here, we address this concern by implementing a distributed memory parallelisation using the message passing interface (MPI) standard. However, we limit our implementation efforts by considering 1) a one-dimensional GPU topology, 2) no computation/communication overlaps, and 3) only mesh-related quantities are shared amongst GPUs, i.e., the material points are not transferred between GPUs during a simulation. We also selected a non-adaptative time step to avoid the collection of the material point's velocities located in different GPUs at the beginning of each calculation cycle.

Available computational resources

The multi-GPU simulations are performed on the supercomputer Octopus running on a CentOS with the latest CUDA version v11. The multi-GPU simulations are run on the two different systems. The first one is an Nvidia DGX-1 - like node hosting 8 Nvidia Tesla V100 Nvlink (32 GB) GPUs, 2 Intel Xeon Silver 4112 (2.6 GHz) CPUs. The second one is composed of 32 nodes, each featuring 4 Nvidia GeForce GTX Titan X Maxwell (12 GB) GPUs, 2 Intel XEON E5-2620V3 4112 (2.4 GHz) CPUs. To summarize the computational resources in use, Table 1 presents the main characteristics of the GPUs used in this study.

Table 1: List of the graphical processing units (GPUs) used for multi-GPU simulations.

GPU	Architecture	On-chip memory [GB]
8×V100	Volta	8×32
128×GTX Titan X	Maxwell	128×12

Model 2a

To avoid frequent material point's transfers amongst the GPUs, we consider an overlap of 8 elements between neighbouring meshes, i.e., 9 nodes. This results in a one-dimensional GPU topology, for which both material points and meshes are distributed along the $y-$direction of the global computational domain (see Figs. 1 & 2). Arranging GPUs along this direction allows to overcome the need to transfer material points amongst GPUs, provided that the material point's displacement is not greater than the buffer zone, i.e., the element overlap. The evaluation of the multi-GPU implementation is based on the Model 2a, with slight modifications, i.e., the number of element along the y-direction is largely increased. The size of the physical domain $l_z \times l_x \times l_y$ is, at most, 12 m × 64 m × (64×2048) m.

Model 2a: multi-GPU performances

We consider two distributed computing systems for parallel GPU computation, using up to 8 Tesla V100 (Volta architecture) or 128 Geforce GTX Titan X (Maxwell architecture). All numerical simulations are performed using a single-arithmetic precision (i.e., $n_\mathrm{p} = 4$ bytes). This allows to increase the maximum number of material points and mesh dimensions. In addition, our GPU implementation relies on the usage of the built-in function `atomicAdd()`. It does not support the double-precision floating-point format FP64 for GPUs with compute capabilities lower than 6.0, i.e., the Maxwell architecture amongst others.

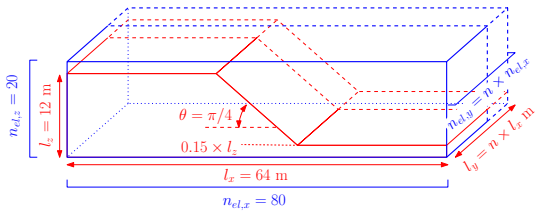

Figure 1: Geometry for the earth slump. For the multi-GPU implementation, the number of element along the $y-$direction can be largely increased, i.e., $n = 2048$.

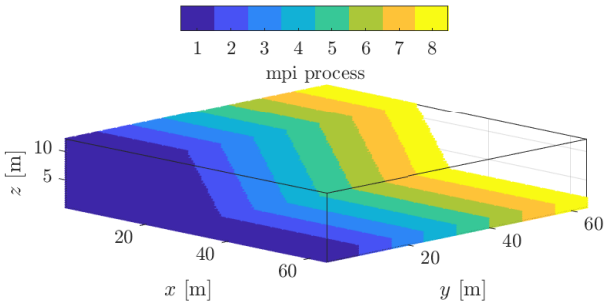

Figure 2: Domain partition of the material points amongst 8 GPUs. Combined with an overlap of 8 elements along the y-direction, material points can moderately move while still residing within the same GPU during the whole simulation.

Note that, unlike the Tesla V100, the Geforce GTX Titan X only delivers an effective memory throughput of $\text{MTP}_{\text{eff}} \approx 100$ GBs^{-1}. This corresponds to 38 % of its hardware limit. This was already reported by Räss et al., 2019; Alkhimenkov et al., 2021 and, it could be attributed to its older Maxwell architecture (Gao et al., 2018). This performance drop is even more severe, mainly due to the use of built-in functions like `atomicAdd()`.

Computing system: up to 8 Tesla V100

We first performed parallel simulations with a moderate number of GPUs, i.e., up to 8 Tesla V100 NVlink (32 GB). The respective wall-clock times are reported in Fig. 3. We report a wall-clock time of ≈ 110 s for $n_{mp} \approx 10^8$. For the same amount of material points, we report a roughly weak scaling between the number of GPUs and the wall-clock time. If n_{mp} is increased by a factor 2, 4 or 8, the wall-clock time is roughly similar to the baseline, i.e., $n_{\text{GPU}} = 1$.

Such weak scaling is more obvious when inspecting the MTP_{eff} measured (see Fig. 4), i.e., the total sum of MTP_{eff} across all the GPUs. Based on the memory throughput of 1 GPU, an estimation of a perfect weak scaling is possible. For 8 GPUs, it should correspond to $\text{MTP}_{\text{eff}} = 4824$ GBs^{-1}, whereas we report $\text{MTP}_{\text{eff}} = 4538$ GBs^{-1}. This gives a parallel efficiency of $\approx 94\%$ and, an effective speed-up of $7.5\times$. Similar observations are made for $n_{\text{GPU}} = 2$ and $n_{\text{GPU}} = 4$.

Computing system: up to 128 Geforce GTX Titan X

We investigate a parallel GPU computing using up to 128 Geforce GTX Titan X. This allows to address even larger geometries, as showed in Fig 5 where a geometry of nearly $n_{mp} \approx 8 \cdot 10^8$ is resolved in less than 8

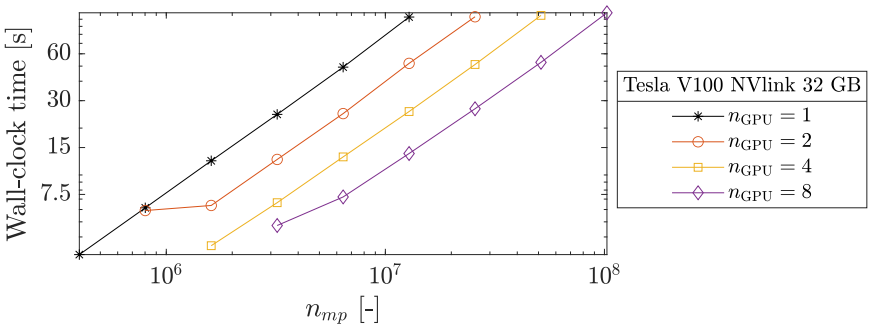

Figure 3: Wall-clock time for 1, 2, 4 and 8 Tesla V100 GPUs.

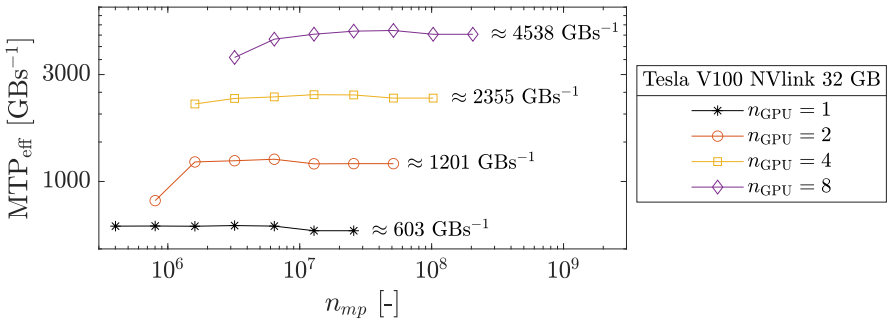

Figure 4: Sum across the GPUs involved of the MTP_{eff}. We roughly report a weak scaling between the number of GPUs and the overall effective memory throughput.

minutes. The first observation is that, for parallel computing up to 64 GPUs, the wall-clock time evolution is smooth. For 128 GPUs, the wall-clock time is chaotic for fewer material points whereas it stabilizes as the number of material points increases. We suspect the absence of computation/communication overlaps to be the main reason of this erratic behaviour. The communication between many GPUs requires careful synchronization between GPUs which can be hidden under computation/communication overlap. The total size of the overlap is constant, regardless of the $y-$dimension. As the number of material points increases, the time spent on computation becomes larger compared to the time spent on exchanges between GPUs and the wall-clock time stabilizes.

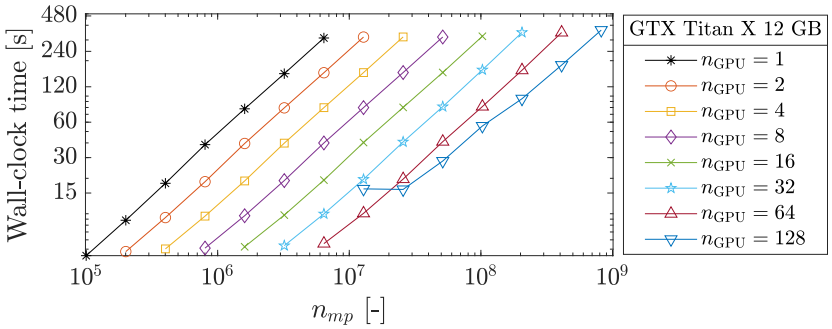

Figure 5: Wall-clock time reported for up to 128 Geforce GTX Titan X GPUs and up to $n_{mp} \approx 8 \cdot 10^8$.

Another observation is the effective memory throughput (see Fig. 6). When considering a perfect weak scaling, one should measure an effective memory throughput $\text{MTP}_{\text{eff}} = 12800$ GBs^{-1} for 128 GPUs whereas we report only $\text{MTP}_{\text{eff}} = 10953$ GBs^{-1}. This gives a parallel efficiency of $\approx 85\%$ and, an effective speed-up of $\approx 110\times$. When using less GPUs, the parallel efficiency is higher, i.e., 98 % for 8 GPUs.

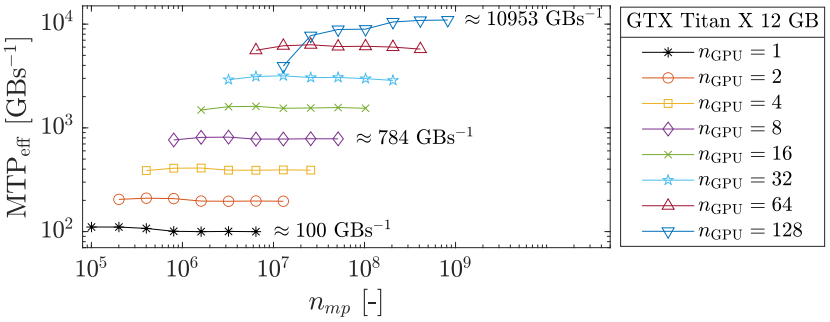

Figure 6: MTP_{eff} sum across the GPUs involved.

Discussion

Even tough the simplifications made alleviate the on-chip memory limitation, the type of problem, which can be addressed, is reduced. As an example, investigating high-resolution three-dimensional granular collapses is not possible under the assumptions made, because of small displacement required along the $y-$direction. This is incompatible with three-dimensional granular collapses. Hence, this motivates future deeper investigations toward a more versatile multi-GPU implementation. In addition, we report a slight drop of the parallel efficiency, as the number of GPUs increases. Future works should be directed toward a parallel strategy that hides communication latency, as proposed in Räss et al., 2019; Räss et al., 2020; Alkhimenkov et al., 2021.

However, such multi-GPU implementation is particularly well-suited to resolve highly-detailed three-dimensional shear-banding. We also reported decent wall-clock times (less than 8 minutes) for simulations with nearly a billion material points. One could argue that limiting the material point method to small displacement is a non-sense. Essentially, finite element codes are better suited for small strain analysis. However, this gives interesting insights on a multi-GPU implementation of the material point framework on a GPU supercomputer.

References

Alkhimenkov, Y., L. Räss, L. Khakimova, B. Quintal, and Y. Podladchikov (2021). "Resolving wave propagation in anisotropic poroelastic media using graphical processing units (GPUs)". In: *Journal of Geophysical Research: Solid Earth* n/a.n/a. e2020JB021175 2020JB021175, e2020JB021175. DOI: `https://doi.org/10.1029/2020JB021175`.

Gao, M., X. Wang, K. Wu, A. Pradhana, E. Sifakis, C. Yuksel, and C. Jiang (Dec. 2018). "GPU Optimization of Material Point Methods". In: *ACM Trans. Graph.* 37.6. DOI: `10.1145/3272127.3275044`.

Räss, L, T. Duretz, and Y. Podladchikov (2019). "Resolving hydromechanical coupling in two and three dimensions: spontaneous channelling of porous fluids owing to decompaction weakening". In: *Geophysical Journal International* 218.3, pp. 1591–1616.

Räss, L., A. Licul, F. Herman, Y. Y. Podladchikov, and J. Suckale (2020). "Modelling thermomechanical ice deformation using an implicit pseudo-transient method (FastICE v1. 0) based on graphical processing units (GPUs)". In: *Geoscientific Model Development* 13.3, pp. 955–976.

Wang, B., P. J. Vardon, and M. A. Hicks (2016a). "Investigation of retrogressive and progressive slope failure mechanisms using the material point method". In: *Computers and Geotechnics* 78, pp. 88–98. DOI: `https://doi.org/10.1016/j.compgeo.2016.04.016`.

Wang, B., P. J. Vardon, M. A. Hicks, and Z. Chen (2016b). "Development of an implicit material point method for geotechnical applications". In: *Computers and Geotechnics* 71, pp. 159–167. DOI: `https://doi.org/10.1016/j.compgeo.2015.08.008`.