https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

Fast and efficient MATLAB-based MPM solver (fMPMM-solver v1.0)

Emmanuel Wyser[1], Michel Jaboyedoff[1,2], and Yury Y. Podladchikov[1,2]

[1]Institute of Earth Sciences, University of Lausanne, 1015 Lausanne, Switzerland
[2]Swiss Geocomputing Centre, University of Lausanne, 1015 Lausanne, Switzerland

Correspondence: Emmanuel Wyser (manuwyser@gmail.com)

Abstract. In this contribution, we present an efficient MATLAB-based implementation of the material point method (MPM) and its most recent variants.

MPM has gained popularity over the last decade, especially for problems in solid mechanics in which large deformations are involved, i.e., cantilever beam problems, granular collapses and even large-scale snow avalanches. Although its numerical accuracy is lower than that of the widely accepted finite element method (FEM), MPM has been proven useful in overcoming some of the limitations of FEM, such as excessive mesh distortions.

We demonstrate that MATLAB is an efficient high-level language for MPM implementations that solve elasto-dynamic and elasto-plastic problems, such as the cantilever beam and granular collapses, respectively. We report a computational efficiency factor of 20 for a vectorized code compared to a classical iterative version. In addition, the numerical efficiency of the solver surpassed those of previously reported MPM implementations in Julia, ad minima 2.5 times faster under a similar computational architecture.

1 Introduction

The material point method (MPM), developed in the 1990s (Sulsky et al. (1994)), is an extension of a particle-in-cell (PIC) method to solve solid mechanics problems involving massive deformations. It is an alternative to Lagrangian approaches (updated Lagrangian finite element method) that is well suited to problems with large deformations involved in geomechanics, granular mechanics or even snow avalanche mechanics. Wang et al. (2016b); Vardon et al. (2017) investigated elasto-plastic problems of strain localization of slumping processes relying on an explicit or implicit MPM formulation. Similarly, Bandara et al. (2016); Bandara and Soga (2015) proposed a poro-elasto-plastic MPM formulation to study levee failures induced by pore pressure increases. Additionally, Dunatunga and Kamrin (2017, 2015); Wieckowski (2004); Bardenhagen et al. (2000) proposed a general numerical framework of granular mechanics, i.e., silo discharge or granular collapses. More recently, Gaume et al. (2018) proposed a unified numerical model in the finite deformation framework to study the whole process, i.e., from failure to propagation, of slab avalanche releases. The core idea of MPM is to discretize a continuum with material points carrying state variables (e.g., mass, stress, and velocity). The latter are mapped (accumulated) to the nodes of a regular

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

or irregular background FE mesh, on which an Eulerian solution to the momentum balance equation is explicitly advanced forward in time.

Nodal solutions are then mapped back to the material points, and the mesh can be discarded. The mapping from material points to nodes is ensured using the standard FE hat function that spans over an entire element (Bardenhagen and Kober (2004)). This avoids a common flaw of FEM, which is an excessive mesh distortion. We will refer to this first variant as the standard material point method (sMPM).

MATLAB© allows a rapid code prototyping but at the expense of significantly lower computational performance than C or C++. However, Sinaie et al. (2017) recently demonstrated that an MPM implementation in Julia offers significantly higher performances when compared to a similar implementation in MATLAB. An efficient MATLAB implementation of FEM called MILAMIN (Million a Minute) was proposed by Dabrowski et al. (2008) that was capable of solving two-dimensional problems with one million unknowns in one minute on a modern computer with a reasonable architecture.

Since MPM and FEM share common grounds, we aim at increasing the performances of MATLAB up to what was reported by Sinaie et al. (2017), combining the most recent and accurate versions of MPM: the explicit GIMPM (generalized interpolation material point method, Bardenhagen and Kober (2004)) and the explicit convected particle domain interpolation with second-order quadrilateral domains (CPDI2q, Sadeghirad et al. (2013, 2011)) variants with some of the numerical considerations from the FEM solver MILAMIN.

In this manuscript, we present an implementation of an efficiently vectorized explicit MPM solver (fMPMM-solver, which v1.0 is available for download from Bitbucket at: https://bitbucket.org/ewyser/fmpmm-solver/src/master/), taking advantage of vectorization capabilities of MATLAB©, i.e., extensive use of built-in functions such as `repmat( )`, `reshape( )`, `sum( )`, `sparse( )` or `accumarray( )`. We compare our results with i) classical MPM benchmarks, i.e., the elastic collapse under self-weight, and ii) validate our in-house code with existing experimental results, i.e., granular collapses. We conclude by showcasing examples of applications to landslide mechanics.

2 Overview of the Material Point Method (MPM)

2.1 A Material Point Method implementation

The material point method (MPM), originally proposed by Sulsky et al. (1995, 1994) in an explicit format, is an extension of the particle-in-cell (PIC) method.

The key idea is to solve the weak form of the momentum balance equation on an FE mesh while state variables (e.g., stress, velocity or mass) are stored at Lagrangian points discretizing the continuum, i.e., the material points, which can move according to the deformation of the grid Dunatunga and Kamrin (2017). MPM could be regarded as a finite element solver in which integration points (material points) are allowed to move (Guilkey and Wiess (2003)) and are thus not always located at the Gauss-Legendre location within an element, resulting in higher quadrature errors and poorer integration estimates, especially when using low-order basis functions Steffen et al. (2008b, a).

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions
Open Access
EGU

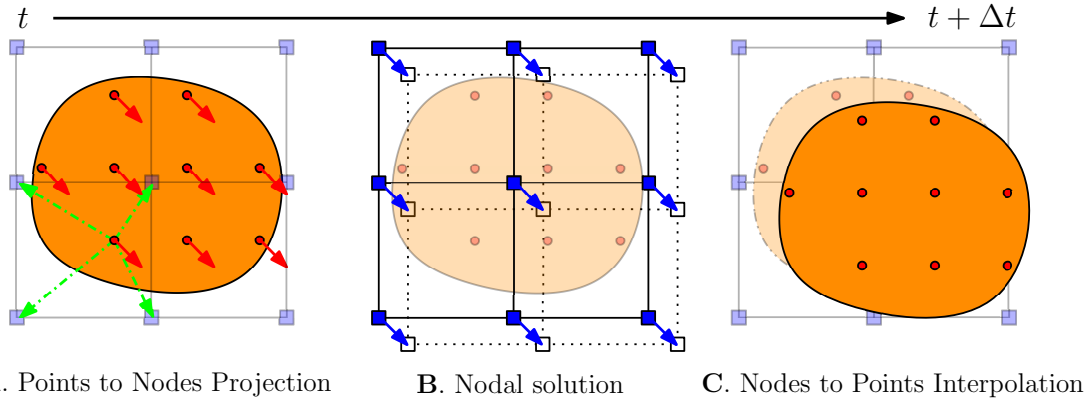

Figure 1. Typical calculation cycle of an MPM solver for an homogeneous velocity field, inspired by Dunatunga and Kamrin (2017). The continuum (orange) is discretized into a set of Lagrangian material points (red dots), at which state variables or properties (e.g., mass, stress, and velocity) are defined. The latter are mapped to an Eulerian finite element mesh made of nodes (blue square).

Since the 1990's, several variants were introduced to resolve a number of numerical issues. The generalized interpolation material point method (GIMPM) was first presented by Bardenhagen and Kober (2004) and proposed a generalization of the basis and gradient functions that were convoluted with a characteristic domain function of the material point. A major flaw in sMPM is the lack of continuity of the gradient basis function, resulting in spurious oscillations of internal forces as soon as a material point crosses an element boundary while entering into its neighbour. This is referred to as cell-crossing instabilities due to the C_0 continuity of the gradient basis functions, and it is minimized by the GIMPM variant. This variant allows the material point's domain to deform (cpGIMPM) or not (uGIMPM) according to its deformation gradient tensor (see Coombs et al. (2020)). Hence, GIMPM is categorized as a domain-based material point method, unlike the later development of the B-spline material point method (BSMPM, see Gaume et al. (2018); Stomakhin et al. (2013); Steffen et al. (2008b)) which cures cell-crossing instabilities using B-spline functions as basis functions. Whereas in sMPM only nodes belonging to an element contribute to a given material point, GIMPM requires an extended nodal connectivity, i.e., the nodes of the element enclosing the material point and the nodes belonging to the adjacent elements (see Fig. 2). Last, the convected particle domain interpolation proposed by Sadeghirad et al. (2013, 2011) accounts for the deformation and the rotation of the material point's domain, the latter being considered either as a deforming parallelogram (CPDI) or as a deforming quadrilateral (CPDI2q). These variants require some additional considerations during the vectorization task.

We choose the explicit GIMPM variant with the modified update stress last scheme (MUSL, see Nairn (2003); Bardenhagen et al. (2000) for a detailed discussion), i.e., the stress of material point is updated after the nodal solutions are obtained. The updated momentum of the material point is then mapped back a second time to the nodes in order to obtain an updated nodal velocity, further used to calculate gradient terms such as the strains or the deformation gradient of the material point. The explicit format also implies the well-known restriction on the time step, which is limited by the Courant-Friedrich-Lewy (CFL)

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions
Open Access
EGU

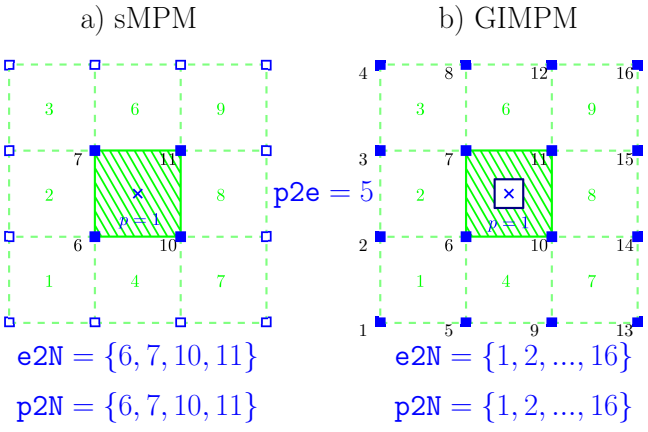

Figure 2. Nodal connectivities of a) standard MPM and b) GIMP variants. The material point's location is marked by the blue cross. Note that for sMPM (and similarly BSMPM) the particle domain does not exist, unlike for GIMP (the blue square enclosing the material point). Nodes associated with the material point are denoted by filled blue squares, and the element number appears in green in the center of the element. The connectivity array between the material point and the element is `p2e`, the array between the element and its associated nodes is `e2N` and the array between the material point and its associated nodes is `p2N`.

condition to ensure numerical stability. The domain update is based on the diagonal components of the stretch tensor of the material point (Charlton et al. (2017)), which ensures optimal convergence properties according to Coombs et al. (2020).

Additionally, we implemented a CPDI/CPDI2q version (in an explicit and quasi-static implicit format) of the solver. However, in this paper, we do not present the theoretical background of the CPDI variant nor the implicit implementation of an MPM-based solver, and therefore, interested readers are referred to the original contributions of Sadeghirad et al. (2013, 2011) and Charlton et al. (2017); Iaconeta et al. (2017); Beuth et al. (2008); Guilkey and Wiess (2003), respectively.

3 MATLAB-based MPM implementation

3.1 Structure of the MPM solver

The solver procedure is shown in Fig. 3. A typical MPM calculation cycle consists of the three following steps Wang et al. (2016a):

1. A Mapping phase, during which properties of the material point (mass, momentum or stress) are mapped to the nodes.
2. An Updated-Lagrangian FEM (UL-FEM) phase, during which the momentum equations are solved on the nodes and the solution is explicitly advanced forward in time.
3. A Convection phase, during which i) the nodal solutions are interpolated back to the material points, and ii) the properties of the material point are updated.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

In the `main.m` script, both functions `meSetup.m` and `mpSetup.m`, respectively, define the geometry and related quantities, particularly the nodal connectivity array, i.e., the `e2N` array (Simpson (2017)). The latter stores the nodes associated with a given element. As such, a material point p located in an element e can immediately identify which nodes N it is associated with.

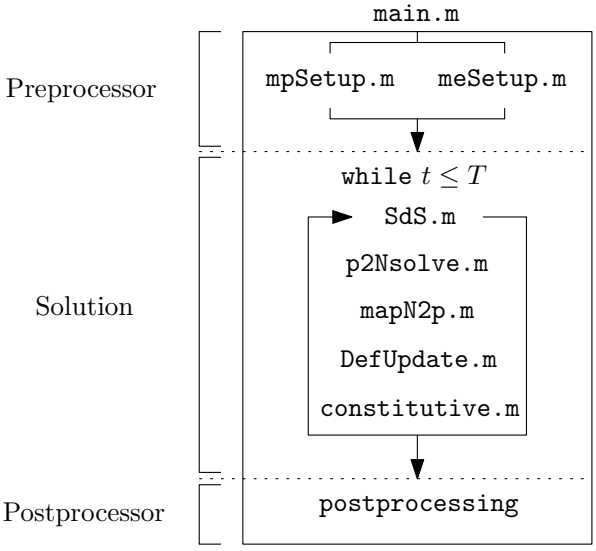

Figure 3. Workflow of the explicit GIMPM solver and the calls to functions within a calculation cycle. The role of each function is described in the text.

After initialization, a while loop solves the elasto-dynamic (or elasto-plastic) problem until a time criterion T is reached. This time criterion could be restricted to the time needed for the system to reach an equilibrium, or if the global kinetic energy of the system has reached a threshold.

At the beginning of each cycle, a connectivity array `p2e` between the material points and their respective element (a material point can only reside in a single element) is created. Since i) the nodes associated with the elements and ii) the elements enclosing the material points are known, it is possible to obtain the connectivity array `p2N` between the material points and their associated nodes, e.g., `p2N=e2N(p2e,:)` in a MATLAB syntax (see Fig. 2 for an example of these connectivity arrays). This array is of dimension (n_p, n_{Ne}), with n_p the total number of material points, n_{Ne} the total number of nodes associated with an element (16 in two-dimensional problems) and $n_{i,j}$ the node number where i corresponds to the material point and j corresponds to its j-th associated nodes, which results in the following:

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions
Open Access
EGU

$$\texttt{p2N} = \begin{pmatrix} n_{1,1} & n_{1,2} & \cdots & n_{1,n_{Ne}} \\ n_{2,1} & n_{2,2} & \cdots & n_{2,n_{Ne}} \\ \vdots & \vdots & \ddots & \vdots \\ n_{n_p,1} & n_{n_p,2} & \cdots & n_{n_p,n_{Ne}} \end{pmatrix} \tag{1}$$

The following functions are called successively during one cycle:

1. `SdS.m` calculates the basis functions and the gradient of the basis functions and assembles the strain-displacement matrix.

2. `p2Nsolve.m` projects the quantities of the material point (e.g., mass and momentum) to the associated nodes, solves the equations of motion and sets Dirichlet boundary conditions.

3. `mapN2p.m` interpolates nodal solutions (acceleration and velocity) to the material points with a double mapping procedure (see Zhang et al. (2017) or Nairn (2003) for a clear discussion of USF, USL and MUSL algorithms).

4. `DefUpdate.m` updates incremental strains and deformation-related quantities (e.g., the volume of the material point or the domain half-length) at the level of the material point based on the remapping of the updated material point momentum.

5. `constitutive.m` calls two functions to solve for the constitutive elasto-plastic relation, i.e.,

   (a) `elastic.m`, which predicts an incremental objective stress-rate (the Jaumann stress-rate is selected for its ease of implementation and its acknowledged accuracy) considering a purely elastic step, further corrected by

   (b) `plastic.m`, which corrects the trial stress by a plastic correction if the material has yielded.

When a time criterion is met, the calculation cycle stops and further post-processing tasks (visualization, data exportation) can be performed.

The numerical simulations are conducted using MATLAB© R2018a within a Windows 7 64-bit environment on an Intel Core i7-4790 (4th generation CPU with 4 physical cores of base frequency at 3.60 GHz up to a maximum turbo frequency of 4.00 GHz) with 8 MB cache and 16 GB DDR3 RAM (clock speed 800 MHz).

3.2 Vectorization

3.2.1 Basis Functions and derivatives

The GIMPM basis function (Coombs et al. (2018); Steffen et al. (2008b); Bardenhagen and Kober (2004)) results from the convolution of a characteristic particle function χ_p (i.e., the material point spatial extent or domain) with the standard basis

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions
Open Access
EGU

function $N_I(x_p)$ of the mesh, which results in:

$$S_I(x_p) = \begin{cases} 1 - (4x^2 + l_p^2)/(4hl_p) & \text{if } |x| < l_p/2 \\ 1 - |x|/h & \text{if } l_p/2 \leq |x| < h - l_p/2 \\ (h + l_p/2 - |x|)^2/(2hl_p) & \text{if } h - l_p/2 \leq |x| < h + l_p/2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

with l_p the length of the material point domain, h the mesh spacing, $x = x_p - x_I$ where x_p is the coordinate of a material point and x_I the coordinate of its associated node I. The basis function of a node I with its material point p is constructed for a two-dimensional model, as follows:

$$S_I(\boldsymbol{x}_p) = S_I(x_p) S_I(y_p) \tag{3}$$

for which the derivative is defined as:

$$\nabla S_I(\boldsymbol{x}_p) = (\partial_x S_I(x_p) S_I(y_p), S_I(x_p) \partial_y S_I(y_p)) \tag{4}$$

Within the GIMPM variant, uGIMPM (undeformed GIMPM) and cpGIMPM (contiguous particle GIMPM) can be chosen to update the material point domain length l_p^0, i.e., $l_p = l_p^0$ for the undeformed GIMPM, $l_{i,p} = \det(F_{jk}) l_{i,p}^0$ or $l_{i,p} = F_{ii} l_{i,p}^0$ where F_{ii} is the diagonal component of the deformation gradient for the cpGIMPM. However, Charlton et al. (2017) recommend using the diagonal components $U_{ii} = (F_{kj} F_{ki})^{0.5}$ of the stretch part of the deformation gradient instead of the deformation gradient. Hence, the variant we use relies either on the determinant or on the stretch part of the deformation gradient.

Similar to the FEM, the strain-displacement matrix $\boldsymbol{B}$ consists of the derivatives of the basis function and is assigned to each material point, which results in the following:

$$\boldsymbol{B}(\boldsymbol{x}_p) = \begin{pmatrix} \partial_x S_1 & 0 & \cdots & \partial_x S_{n_{Ne}} & 0 \\ 0 & \partial_y S_1 & \cdots & 0 & \partial_y S_{n_{Ne}} \\ \partial_y S_1 & \partial_x S_1 & \cdots & \partial_y S_{n_{Ne}} & \partial_x S_{n_{Ne}} \end{pmatrix} \tag{5}$$

where n_{Ne} is the total number of associated nodes to an element e, in which a material point p resides.

The algorithm outlined in Code Fragment 1 (the function `[mpD] = SdS(meD,mpD,p2N)` called at the beginning of each cycle, see Fig. 4) represents the vectorized solution of the computation of basis functions and their derivatives in just one time step, which avoids any for-loop requirement.

Table 1 lists the variables used in Code fragments 1 & 2 (Figs. 4 & 5).

Coordinates of the material points `mpD.x(:,1:2)` are first replicated and then subtracted by their associated nodes coordinates, e.g., `meD.x(p2N)` or `meD.y(p2N)` (Lines 3 or 4 in 4). This yields the array `D` with the same dimension of `p2N`. This array of distance between the points and their associated nodes is sent as an input to the local function `[N,dN] = NdN(D,h,lp)`, which computes 1D basis functions and derivatives through matrix piecewise operations (operator `.*`) (either

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

in Line 4 for x coordinates or Line 6 for y coordinates in Fig. 4). Note that the use of one single array `D` avoids a redundant usage of memory.

Given the piecewise Eq. 2, three logical arrays (`c1`, `c2` and `c3`) are defined (Lines 28-30 in Fig. 4), whose elements are either 1 (the condition is true) or 0 (the condition is false). Three arrays of basis functions are calculated (`N1`, `N2` and `N3`, Lines 32-34) according to Eq. 3. The array of basis functions `N` is obtained through a summation of the elementwise multiplications of these temporary arrays with their corresponding logical arrays (Line 35 in Fig. 4). The same holds true for the calculation of the gradient basis function (Lines 37-40 in Fig. 4). Furthermore, it is faster to use logical arrays as multipliers of precomputed basis function arrays rather than using these in a conditional indexing statement, e.g., `N(c2==1) = 1-abs(dX(c2==1))./h`. The performance gain is significant, i.e., a 30 % gain over a calculation cycle.

Table 1. Description of variables of the structure arrays for the mesh `meD` and the material point `mpD` used in Code Fragment 1 & 2 shown in Figs. 4 & 5. `nDF` stores the local and global number of degrees of freedom, i.e., `nDF=[nNe,nN*DoF]`. The constant `nstr` is the number of stress components, according to the standard definition of the Cauchy stress tensor using the Voigt notation, e.g., $\boldsymbol{\sigma}_p = (\sigma_{xx}, \sigma_{yy}, \sigma_{xy})$.

Variable		Description	Dimension
meD.	nNe	nodes per element	(1)
	nN	number of nodes	(1)
	DoF	degree of freedom	(1)
	nDF	number of DoF	(1,2)
	h	mesh spacing	(1,DoF)
	x	node coordinates	(nN,1)
	y	node coordinates	(nN,1)
	m	node mass	(nN,1)
	p	node momentum	(nDF(2),1)
	f	node force	(nDF(2),1)
mpD.	n	number of points	(1)
	l	domain half-length	(np,DoF)
	V	volume	(np,1)
	m	mass	(np,1)
	x	point coordinates	(np,DoF)
	p	momentum	(np,DoF)
	s	stress	(np,nstr)
	S	basis function	(np,nNe)
	dSx	derivative in x	(np,nNe)
	dSy	derivative in y	(np,nNe)
	B	B matrix	(nstr,nDF(1),np)

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

```
function [mpD] = SdS(meD,mpD,p2N)
%% COMPUTE (X,Y)-BASIS FUNCTION
D        = (repmat(mpD.x(:,1),1,meD.nNe) - meD.x(p2N))              ;%
[Sx,dSx] = NdN(D,meD.h(1),repmat(mpD.l(:,1),1,meD.nNe))             ;%
D        = (repmat(mpD.x(:,2),1,meD.nNe) - meD.y(p2N))              ;%
[Sy,dSy] = NdN(D,meD.h(2),repmat(mpD.l(:,2),1,meD.nNe))             ;%
%-------------------------------------------------------------------%

%% CONVOLUTION OF BASIS FUNCTIONS
mpD.S    =  Sx.* Sy                                                 ;%
mpD.dSx  = dSx.* Sy                                                 ;%
mpD.dSy  =  Sx.*dSy                                                 ;%
%-------------------------------------------------------------------%

%% B MATRIX ASSEMBLY
iDx            = 1:meD.DoF:meD.nDF(1)-1                             ;%
iDy            = iDx+1                                              ;%
mpD.B(1,iDx,:) = mpD.dSx'                                           ;%
mpD.B(2,iDy,:) = mpD.dSy'                                           ;%
mpD.B(4,iDx,:) = mpD.dSy'                                           ;%
mpD.B(4,iDy,:) = mpD.dSx'                                           ;%
%-------------------------------------------------------------------%

end
function [N,dN]=NdN(dX,h,lp)
%% COMPUTE BASIS FUNCTIONS
lp = 2*lp                                                           ;%
c1 = ( abs(dX)< (  0.5*lp)                        )                 ;%
c2 = ((abs(dX)>=(  0.5*lp)) & (abs(dX)<(h-0.5*lp)))                 ;%
c3 = ((abs(dX)>=(h-0.5*lp)) & (abs(dX)<(h+0.5*lp)))                 ;%
% BASIS FUNCTION
N1 = 1-((4*dX.^2+lp.^2)./(4*h.*lp))                                 ;%
N2 = 1-(abs(dX)./h)                                                 ;%
N3 = ((h+0.5*lp-abs(dX)).^2)./(2*h.*lp)                             ;%
N  = c1.*N1+c2.*N2+c3.*N3                                           ;%
% BASIS FUNCTION GRADIENT
dN1= -((8*dX)./(4*h.*lp))                                           ;%
dN2= sign(dX).*(-1/h)                                               ;%
dN3=-sign(dX).*(h+0.5*lp-abs(dX))./(h*lp)                           ;%
dN = c1.*dN1+c2.*dN2+c3.*dN3                                        ;%
%-------------------------------------------------------------------%
end
```

Figure 4. Code Fragment 1 shows the vectorized solution to the calculation of the basis functions and their gradients within the function `SdS.m`.

3.2.2 Integration of internal forces

Another computationally expensive operation for MATLAB© is the mapping (or accumulation) of the material point contributions to their associated nodes. It is performed by the function `p2Nsolve.m` in the workflow of the solver.

The standard calculations for the material point contributions to the lumped mass m_I, the momentum $\boldsymbol{p}_I$, the external $\boldsymbol{f}_I^e$ and internal $\boldsymbol{f}_I^i$ forces are given by:

$$m_I = \sum_{p \in I} S_I(\boldsymbol{x}_p) m_p \tag{6}$$

$$\boldsymbol{p}_I = \sum_{p \in I} S_I(\boldsymbol{x}_p) m_p \boldsymbol{v}_p \tag{7}$$

$$\boldsymbol{f}_I^e = \sum_{p \in I} S_I(\boldsymbol{x}_p) m_p \boldsymbol{b}_p \tag{8}$$

$$\boldsymbol{f}_I^i = \sum_{p \in I} v_p \boldsymbol{B}^T(\boldsymbol{x}_p) \boldsymbol{\sigma}_p \tag{9}$$

with m_p the material point mass, $\boldsymbol{v}_p$ the material point velocity, $\boldsymbol{b}_p$ the body force applied to the material point and $\boldsymbol{\sigma}_p$ the material point Cauchy stress tensor in the Voigt notation.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

Once the mapping phase is achieved, the equations of motions are explicitly solved forward in time on the mesh. Nodal accelerations and velocities are given by:

$$\boldsymbol{a}_I^{t+\Delta t} = m_I^{-1}(\boldsymbol{f}_I^e - \boldsymbol{f}_I^i) \tag{10}$$

$$\boldsymbol{v}_I^{t+\Delta t} = m_I^{-1}\boldsymbol{p}_I + \Delta t \boldsymbol{a}_I^{t+\Delta t} \tag{11}$$

Finally, boundary conditions are applied to the nodes that belong to the boundaries.

```
function [meD] = p2Nsolve(meD,mpD,g,dt,l2g,p2N,BC)
%% INITIALIZATION
% NODAL VECTOR INITIALIZATION
meD.m(:) = 0.0; meD.mr(:) = 0.0 ; meD.f(:) = 0.0 ; meD.d(:) = 0.0   ;%
meD.a(:) = 0.0;  meD.p(:) = 0.0 ; meD.v(:) = 0.0 ; meD.u(:) = 0.0   ;%
%-------------------------------------------------------------------------%

%% CONTRIBUTION TO NODES
% PREPROCESSING
m = reshape(mpD.S.*repmat(mpD.m,1,meD.nNe)      ,mpD.n*meD.nNe   ,1);%
p = reshape([mpD.S.*repmat(mpD.p(:,1),1,meD.nNe);...
             mpD.S.*repmat(mpD.p(:,2),1,meD.nNe)],mpD.n*meD.nDF(1),1);%
f = reshape([mpD.S.*0.0                          ;...
             mpD.S.*repmat(mpD.m,1,meD.nNe).*-g ],mpD.n*meD.nDF(1),1);%
fi= squeeze(sum(permute(mpD.B  ,[2 1 3]).*...
             repmat((permute(mpD.s',[3 2 1])),meD.nDF(1),1),2).*...
             repmat( permute(mpD.V ,[3 2 1]) ,meD.nDF(1),1))         ;%
% CONTRIBUTION FROM p TO N
meD.m = accumarray(p2N(:),m,[meD.nN     1])                          ;%
meD.p = accumarray(l2g(:),p,[meD.nDF(2) 1])                          ;%
meD.f = accumarray(l2g(:),f,[meD.nDF(2) 1])                          ;%
for n = 1:meD.nNe                                                     %
    l = [(meD.DoF*p2N(:,n)-1);(meD.DoF*p2N(:,n))]                    ;%
    meD.f = meD.f - accumarray(l,[fi(n*meD.DoF-1,:)';...
                                  fi(n*meD.DoF  ,:)'],[meD.nDF(2) 1]);%
end                                                                   %
%-------------------------------------------------------------------------%

%% SOLVE EXPLICIT MOMENTUM BALANCE EQUATION
% UPDATE GLOBAL NODAL INFORMATIONS
iDx        = 1:meD.DoF:meD.nDF(2)-1                                  ;%
iDy        = iDx+1                                                   ;%
% COMPUTE GLOBAL FORCE VECTORS
meD.d(iDx) = sqrt(meD.f(iDx).^2+meD.f(iDy).^2)                       ;%
meD.d(iDy) = meD.d(iDx)                                              ;%
meD.f      = meD.f - meD.vd*meD.d.*sign(meD.p)                       ;%
% UPDATE GLOBAL MOMENTUM VECTOR
meD.p      = meD.p + dt*meD.f                                        ;%
% COMPUTE GLOBAL ACCELERATION AND VELOCITY VECTORS
meD.mr     = reshape(repmat(meD.m',meD.DoF,1),meD.nDF(2),1)          ;%
iD         = meD.mr==0                                               ;%
meD.a      = meD.f./meD.mr                                           ;%
meD.v      = meD.p./meD.mr                                           ;%
meD.a(iD)  = 0.0                                                     ;%
meD.v(iD)  = 0.0                                                     ;%
% BOUNDARY CONDITIONS: FIX DIRICHLET BOUNDARY CONDITIONS
BCx =  [BC.xi;BC.xs]                                                 ;%
BCy =  [BC.yi      ]                                                 ;%
meD.a([meD.DoF*BCx-1;meD.DoF*BCy])=0.0                               ;%
meD.v([meD.DoF*BCx-1;meD.DoF*BCy])=0.0                               ;%
meD.a(meD.DoF*(BCy+1))=-meD.a(meD.DoF*(BCy-1))                       ;%
meD.v(meD.DoF*(BCy+1))=-meD.v(meD.DoF*(BCy-1))                       ;%
clear m p f fi iDx iDy iD BCx BCy                                    ;%
%-------------------------------------------------------------------------%

end
```

Figure 5. Code Fragment 2 shows the vectorized solution to the nodal projection of material point quantities (e.g., mass and momentum) within the local function `p2Nsolve.m`. The core of the vectorization process is the extensive use of the built-in function of MATLAB© `accumarray( )`, for which we detail the main features in the text.

The vectorized solution comes from the use of the built-in function `accumarray( )` of MATLAB© combined with `reshape( )` and `repmat( )`. Similar to the function `sparse( )`, `accumarray( )` collects, in a vector $\boldsymbol{e}$, and accumulates, into a second vector $\boldsymbol{e'}$, all values in $\boldsymbol{e}$ that have identical subscripts, whose values are contained in a vector $\boldsymbol{s}$.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions
Open Access
EGU

Such an operation corresponds to the summation operator in Eqs. 6-9. The core of the vectorization is to use `p2N` as the vector (i.e., flattening the array `p2N(:)` results in a row vector) of subscripts with `accumarray`, which accumulates material point contributions (e.g., mass or momentum) that share the same node.

In the function `p2Nsolve` (Code Fragment 2 shown in Fig. 5), the first step is to initialize nodal vectors (mass, momentum, forces, etc.) to zero (Lines 4-5 in Fig. 5). Then, temporary vectors (`m`, `p`, `f` and `fi`) of material point contributions (namely, mass, momentum, and external and internal forces) are generated (Lines 10-17 in Fig. 5). The accumulation (nodal summation) is performed (Lines 19-26 in Fig. 5) using either the flattened `p2n(:)` or `l2g(:)` (e.g., the global indices of nodes) as the vector of subscripts. Note that for the accumulation of material point contributions of internal forces, a short for-loop iterates over the associated node (e.g., from 1 to `meD.nNe`) of every material point to accumulate their respective contributions.

To calculate the temporary vector of internal forces (`fi` at Lines 15-17 in Fig. 5), the first step consists of the matrix multiplication of the strain-displacement matrix `mpD.B` with the material point stress vector `mpD.s`. The vectorized solution is given by i) elementwise multiplications of a permutation `permute(mpD.B,[2 1 3])` (e.g., the transpose operator for multidimensional arrays) with a replication of the transposed stress vector `repmat(permute(mpD.s',[3 2 1]),meD.nDF(1),1)`, whose result is then ii) summed by means of the built-in function `sum( )` along the columns and finally multiplied by a replicated transpose of the material point volume vector, e.g., `repmat(permute(mpD.V,[3 2 1]),meD.nDF(1),1)`.

To illustrate the efficiency of such a procedure, we have developed an iterative and vectorized solution of $\boldsymbol{B}(\boldsymbol{x}_p)^T\boldsymbol{\sigma}_p$ with an increasing n_p and survey the computational time spent on solving the problem. We report a gain in computational time (close to an order of magnitude faster for one order of increase in magnitude of n_p for the vectorized solution).

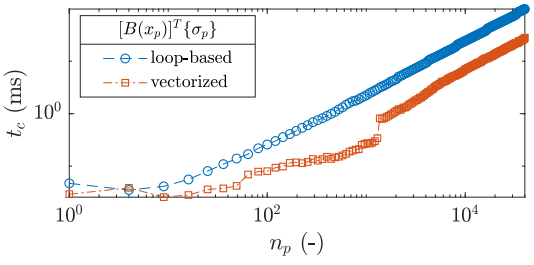

Figure 6. Computational time needed to solve a matrix multiplication between a multidimensional array and a vector with an increasing number of the third dimension.

3.2.3 Update of material point properties

Finally, we propose a vectorization of the function `mapN2p.m` that i) interpolates updated nodal solutions to the material points (velocities and coordinates) and ii) the double mapping (DM or MUSL) procedure (see Fern et al. (2019)). The material

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

point velocity is defined as an interpolation of the solution of the updated nodal accelerations, given by:

$$\boldsymbol{v}_p^{t+\Delta t} = \boldsymbol{v}_p^t + \Delta t \sum_{I=1}^{n_{Ne}} S_I(\boldsymbol{x}_p) \boldsymbol{a}_I^{t+\Delta t} \tag{12}$$

The material point updated momentum is found by $\boldsymbol{p}_p^{t+\Delta t} = m_p \boldsymbol{v}_p^{t+\Delta t}$. The double mapping procedure (MUSL) of the nodal velocity consists of the remapping of the updated material point momentum on the mesh, divided by the nodal mass, which yields the nodal incremental displacements $\Delta \boldsymbol{u}_I$ when multiplied by an increment of time, both given by:

$$\boldsymbol{v}_I^{t+\Delta t} = m_I^{-1} \sum_{p \in I} S_I(\boldsymbol{x}_p) \boldsymbol{p}_p^{t+\Delta t} \tag{13}$$

$$\Delta \boldsymbol{u}_I = \Delta t \boldsymbol{v}_I^{t+\Delta t} \tag{14}$$

and for which boundary conditions are enforced.

Finally, the material point coordinates are updated based on the following:

$$\boldsymbol{x}_p^{t+\Delta t} = \boldsymbol{x}_p^t + \sum_{I=1}^{n_{Ne}} S_I(\boldsymbol{x}_p) \Delta \boldsymbol{u}_I \tag{15}$$

To solve for the interpolation of the updated nodal solutions to the material points, we rely on a combination of elementwise matrix multiplication between the array of basis functions `mpD.S` with the global vectors through a transform of the `p2N` array, i.e., `iDx=meD.DoF*p2N-1` and `iDy=iDx+1` (Lines 3-4 in Code Fragment 3 in Fig. 7), which are used to access to x and y components of global vectors.

When accessing global nodal vectors by means of `iDx` and `iDy`, the resulting arrays are naturally of the same size as `p2N` and are therefore dimension-compatible with `mpD.S`. For instance, a summation along the columns (e.g., the associated nodes of material points) of an elementwise multiplication of `mpD.S` with `meD.a(iDx)` results in an interpolation of the x-component of the global acceleration vector to the material points.

This procedure is used for the velocity update (Lines 6-7 in Fig. 7) and for the material point coordinate update (Line 11 in Fig. 7). A remapping of the nodal momentum is carried out (Lines 17 to 20 in Fig. 7), which allows calculating the updated nodal incremental displacements (Line 22 in Fig. 7). Finally, boundary conditions of nodal incremental displacements are enforced (Lines 29-32 in Fig. 7).

4 Results

4.1 Convergence: elastic compaction under self-weight of a column

Following the convergence analysis proposed by Coombs and Augarde (2020); Wang et al. (2019); Charlton et al. (2017), we analyse an elastic column of an initial height $l_0 = 10$ m subjected to an external load (e.g. the gravity). The initial geometry is shown in Fig. 8. The background mesh is made of a bi-linear four-noded quadrilateral, and roller boundary conditions are

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

```
function [meD,mpD] = mapN2p(meD,mpD,dt,l2g,p2N,BC)
%% PROJECT BACK NODAL ACCELERATION AND VELOCITY TO MATERIAL POINT
iDx        = meD.DoF*p2N-1                                          ;%
iDy        = iDx+1                                                  ;%
% VELOCITY UPDATE
mpD.v(:,1) = mpD.v(:,1)+dt*sum(mpD.S.*meD.a(iDx),2)                 ;%
mpD.v(:,2) = mpD.v(:,2)+dt*sum(mpD.S.*meD.a(iDy),2)                 ;%
% MOMENTUM UPDATE
mpD.p      = mpD.v.*repmat(mpD.m,1,meD.DoF)                         ;%
% COORDINATE UPDATE
mpD.x      = mpD.x+dt*[sum(mpD.S.*meD.v(iDx),2),...
                       sum(mpD.S.*meD.v(iDy),2)]                   ;%
clear iDx iDy                                                       ;%
%-------------------------------------------------------------------%

%% NODAL MOMENTUM WITH MP MOMENTUM (MUSL OR DOUBLE MAPPING)
meD.p(:) = 0.0                                                      ;%
P        = reshape([mpD.S.*repmat(mpD.p(:,1),1,meD.nNe);...
                    mpD.S.*repmat(mpD.p(:,2),1,meD.nNe)],...
                    mpD.n*meD.nDF(1),1                  )           ;%
meD.p    = accumarray(l2g(:),P,[meD.nDF(2) 1])                      ;%
meD.u    = dt*(meD.p./meD.mr)                                       ;%
iD       = meD.mr==0                                                ;%
meD.u(iD)= 0.0                                                      ;%
clear iD                                                            ;%
%-------------------------------------------------------------------%

%% BOUNDARY CONDITIONS: FIX DIRICHLET BOUNDARY CONDITIONS
BCx      = [BC.xi;BC.xs]                                            ;%
BCy      = [BC.yi      ]                                            ;%
meD.u([meD.nDF*BCx-1;meD.nDF*BCy])=0.0                              ;%
meD.u([meD.nDF*(BCy+1)          ])=-meD.u([meD.nDF*(BCy-1)])        ;%
clear BCx BCy                                                       ;%
%-------------------------------------------------------------------%

end
```

Figure 7. Code Fragment 3 shows the vectorized solution for the interpolation of nodal solutions to material points with a double mapping procedure (or MUSL) within the function `mapN2p.m`.

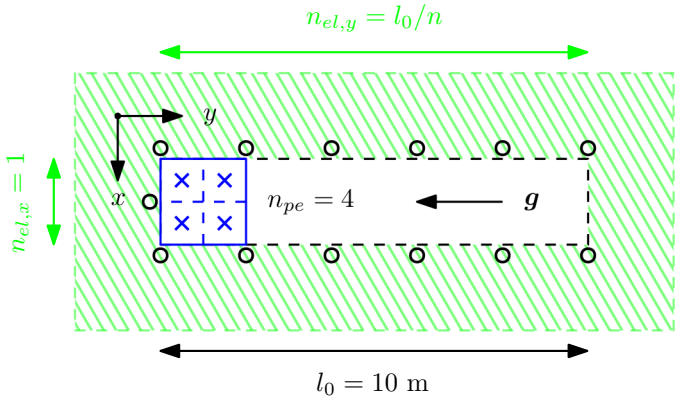

Figure 8. Initial geometry of the column.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

applied on the base and the sides of the column, initially populated by 2^2 material points per element. The column is 1 element wide and n elements tall.

To consistently apply the external load for the explicit solver, we follow the recommendation of Bardenhagen and Kober (2004), i.e., a quasi-static solution (given an explicit integration scheme is chosen) is obtained if the total simulation time is equal to 40 elastic wave transit times. The material has an elastic modulus $E = 1 \cdot 10^6$ Pa and a Poisson's ratio $\nu = 0$ with a density $\rho = 80$ kg m^{-3}. The gravity g is increased from 0 to its final value, i.e., $g = 9.81$ m s^{-2}. We performed additional implicit quasi-static simulations (named iCPDI) in order to consistently discuss the results with respect to what was reported in Coombs and Augarde (2020). The external force is consistently applied over 25 equal load steps. The vertical normal stress is given by the analytical solution (Coombs and Augarde (2020)) $\sigma_{yy}(y_0) = \rho g(l_0 - y_0)$ where l_0 is the initial height of the column and y_0 is the initial position of a point within the column.

The error between the analytical and numerical solutions is as follows:

$$\text{error} = \sum_{p=1}^{n_p} \frac{||(\sigma_{yy})_p - \sigma_{yy}(y_p)||(V_0)_p}{(\rho g l_0) V_0} \tag{16}$$

where $(\sigma_{yy})_p$ is the vertical stress of a material point p of an initial volume $(V_0)_p$ and V_0 is the initial volume of the column, i.e., $V_0 = \sum_{p=1}^{n_p} (V_0)_p$.

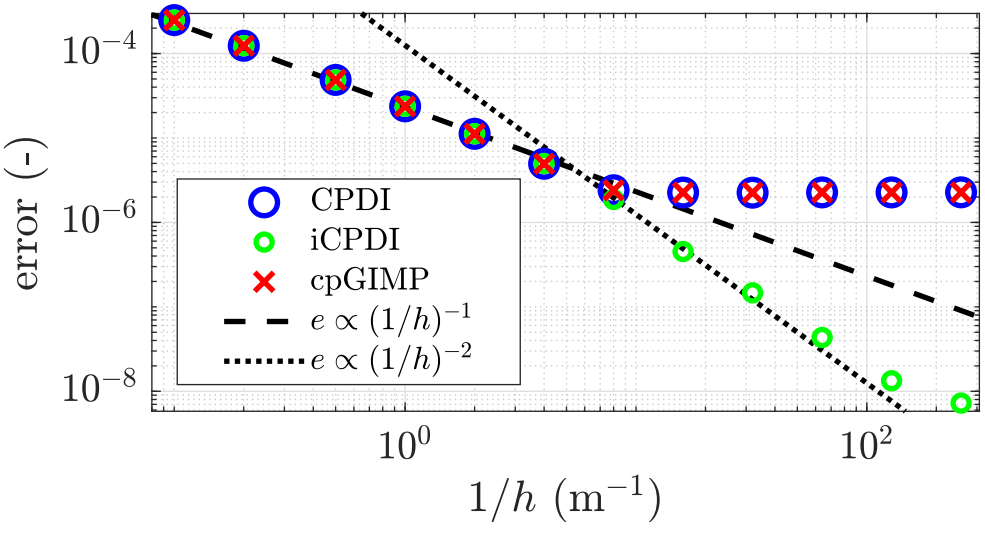

Figure 9. Linear convergence of the error: a limit is reached at error $\approx 2 \cdot 10^{-6}$ for the explicit solver, whereas the quasi-static solution still converges. This was already demonstrated in Bardenhagen and Kober (2004) as an error saturation due to the explicit scheme, i.e., the equilibrium is never resolved.

The convergence toward a quasi-static solution is shown in Fig. 9. As mentioned by Coombs et al. (2020), it is linear for both cpGIMP and CPDI2q, but contrary to Coombs and Augarde (2020); Coombs et al. (2020) who reported a full convergence, it stops at error $\approx 2 \cdot 10^{-6}$ for the explicit implementation. This was already outlined by Bardenhagen and Kober (2004) as a sat-

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

uration of the error caused by resolving the dynamic stress wave propagation, which is inherent to any explicit scheme. Hence, a static solution could never be achieved because, unlike quasi-static implicit methods, the elastic waves propagate indefinitely and the static equilibrium is never resolved. This is consistent when compared to the iCPDI solution we implemented (green circles), whose behaviour is still converging below the limit error $\approx 2 \cdot 10^{-6}$ reached by the explicit solver. In addition, the convergence becomes quadratic below this limit. It confirms that the error saturation is due to the explicit format and not to our numerical implementation.

4.2 Large deformation: the elastic cantilever beam problem

The cantilever beam problem Sinaie et al. (2017); Sadeghirad et al. (2011) is the second benchmark which demonstrates the robustness of the MPM solver. Two MPM variants are implemented, namely, i) the contiguous GIMP (cpGIMP) which relies on the stretch part of the deformation gradient (see Charlton et al. (2017)) to update the particle domain, and ii) the convected particle domain interpolation (CPDI Leavy et al. (2019); Sadeghirad et al. (2011)). In addition, two constitutive elastic models are selected, i.e., neo-Hookean Guilkey and Wiess (2003) or linear elastic York et al. (1999) solids. For consistency, we use the same physical quantities as in Sadeghirad et al. (2011), i.e., an elastic modulus $E = 10^6$ Pa, a Poisson's ratio $\nu = 0.3$, a density $\rho = 1050$ kg/m^3, the gravity $g = 10.0$ m/s and a real-time simulation $t = 3$ s with no damping forces introduced.

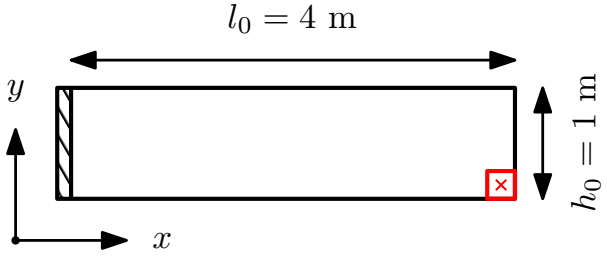

Figure 10. Initial geometry for the cantilever beam problem; the free end material point appears in red where a red cross marks its centre.

The beam geometry is depicted in Fig. 10 and is discretized by 64 bi-linear four-noded quadrilaterals, each of them initially populated by 3^2 material points (e.g., $n_p = 576$) with a time step determined by the CFL condition, e.g., $\Delta t = 10^{-3}$ s. The large deformation is initiated by suddenly applying the gravity at the beginning of the simulation, i.e., $t = 0$ s.

As indicated in Sadeghirad et al. (2011), the cpGIMP simulation failed when using the diagonal components of the deformation gradient to update the material point domain. However, as expected, the cpGIMP simulation succeeded when using the diagonal terms of the stretch part of the deformation tensor, as proposed by Charlton et al. (2017). The numerical solutions, obtained by the latter cpGIMP and CPDI, to the vertical deflection Δu of the material point at the bottom free end of the beam (e.g., the red cross in Fig. 10) are shown in Fig. 11. Some comparative results reported by Sadeghirad et al. (2011) are depicted by black markers (squares for the FEM solution and circles for the CPDI solution), whereas the results of our MPM solver are depicted by lines.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

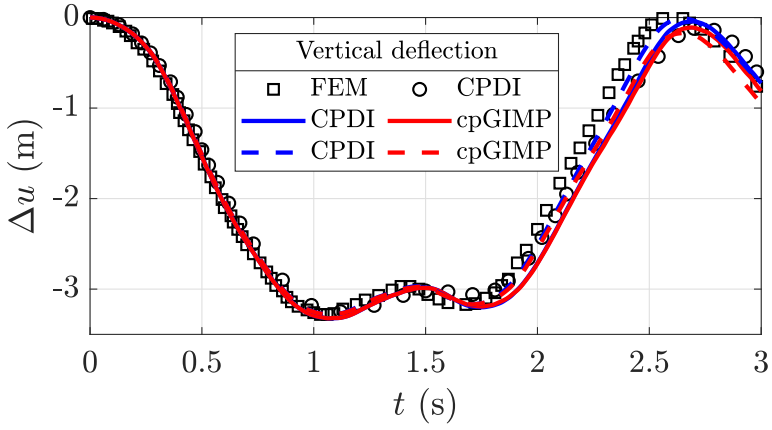

Figure 11. Vertical deflection response for the cantilever beam problem. The black markers denote the solutions of Sadeghirad et al. (2011) (circles for CPDI and squares for FEM). The line colour indicates the MPM variant (blue for CPDI and red for cpGIMP), solid lines refer to a linear elastic solid, whereas dashed lines refer to a neo-Hookean solid. The vertical deflection Δu corresponds to the vertical displacement of the bottom material point at the free end of the beam (the red cross in Fig. 10).

The local minimal and the minimal and maximal values (in timing and magnitude) are in agreement with the FEM solution of Sadeghirad et al. (2011). Moreover, the elastic response is in agreement with the CPDI results reported by Sadeghirad et al. (2011) but differs in timing with respect to the FEM solution. This confirms our numerical implementation of CPDI when compared to the one proposed by Sadeghirad et al. (2011). In addition, the elastic response does not substantially differ from a linear elastic solid to a neo-Hookean one. It demonstrates the incremental implementation of the MPM solver to be relevant in capturing large deformations for the cantilever beam problem.

4.3 Elasto-plasticity: the column collapse

We compare our MPM solver with a non-associated plasticity based on a Drucker-Prager model with tension cutoff (Huang et al. (2015)) to the experimental results of an elasto-plastic collapse of a material (e.g., an aluminium-bar assemblage, see Bui et al. (2008)). The numerical implementation is detailed in Huang et al. (2015), and we therefore suggest the interested reader to directly refer to their contribution since we do not describe the elasto-plastic model in this manuscript.

Our elasto-plastic MPM solver closely follows the implementation of Huang et al. (2015), except our solver relies on an MUSL procedure, whereas Huang et al. (2015) selected the USF procedure. The elasto-plastic problem is solved by i) an elastic trial, ii) corrected by a return mapping when the material yields either in shear or in tension or both, assuming a non-associated (the dilation angle $\psi = 0$) perfectly plastic behaviour of the material. Since the MPM variant in Huang et al. (2015) was not clearly stated, we use the uGIMP variant. The reason is the collapse results in extreme deformations, for which a domain update based on the deformation gradient systematically resulted in a failure during the simulation. We conducted preliminary investigations and concluded that the uGIMP variant was the most reliable MPM variant for such a problem.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions
Open Access
EGU

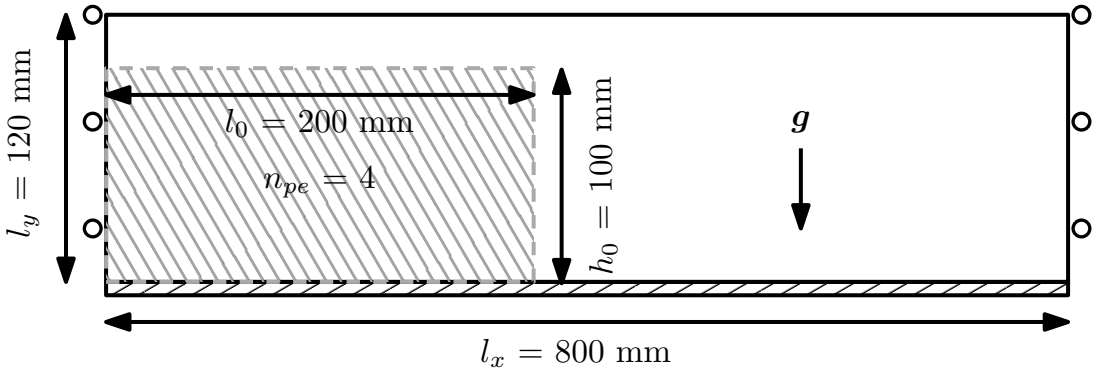

Figure 12. Initial geometry for the elasto-plastic collapse (Huang et al. (2015)). Roller boundaries are imposed on the left and right boundaries of the domain while a no-slip condition is enforced at the bottom of the domain. The aluminium-bar assemblage has dimensions of $l_0 \times h_0$ and is discretized by $n_{pe} = 4$ material points per element, and Table 2 summarizes the material properties.

The initial geometry and boundary conditions used for this problem are depicted in Fig. 12, while the parameters used are summarized in Table 2 and represent the problem described in Bui et al. (2008). The aluminium assemblage is discretized by $n_p = 28'800$ material points within a mesh made of 320 $\times$ 48 quadrilateral elements, resulting in a uniform spacing of $h = 1.25$ mm. The time step is given by the CFL condition, e.g., $\Delta t = 7.02 \cdot 10^{-5}$ s for a total simulation time of 1.25 s.

Table 2. Parameters used for the elasto-plastic collapse simulation. The values of parameters are those found in Huang et al. (2015), obtained using a shear box test by Bui et al. (2008).

Parameter	Symbol	Value	Unit
Density	ρ	2650	kg m^{-3}
Poisson's ratio	ν	0.3	-
Bulk modulus	K	0.7	MPa
Cohesion	c	0	Pa
Internal friction angle	ϕ	19.8	$^\circ$
Dilation angle	ψ	0	$^\circ$
Gravity	g	-9.81	m s^{-2}

Figure 13 shows the numerical solution compared with the experimental results of Bui et al. (2008). We observe a good agreement between the numerical simulation and the experiments, considering either the final surface (blue square dotted line) or the failure surface (blue circle dotted line). Similarly, the repose angle in the numerical simulation is approximately 13° from the horizontal, which is also in agreement with the experimental data reported by Bui et al. (2008) (e.g., they reported a final angle of 14°).

These numerical results demonstrate the solver to be in agreement with both previous experimental (Bui et al. (2008)) and numerical results (Huang et al. (2015)) and confirms its ability to solve elasto-plastic problems such as granular collapses

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions
Open Access
EGU

using an appropriate constitutive model. However, it also demonstrates the inability of the MPM variants based on a domain update (GIMPM or CPDI) to resolve extremely large plastic deformations when relying on the normal components of the deformation gradient or its stretch part to update the material point domain (interested readers are referred to the contribution of Coombs et al. (2020) regarding the suitability of different domain update variants). Consecutively, we performed an additional simulation using a domain update based on the determinant of the deformation gradient. No significant differences were observed with the experimental results, and the simulation succeeded.

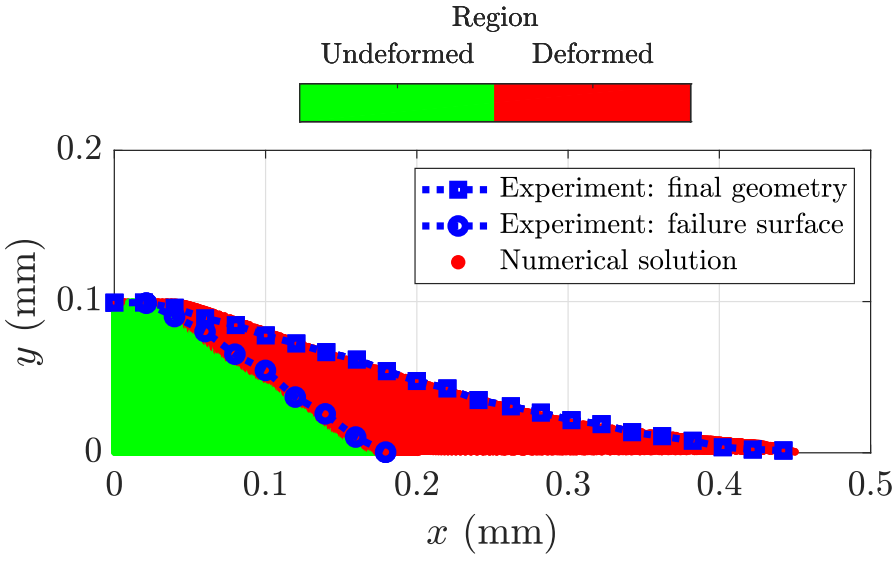

Figure 13. Final geometry of the collapse: in the intact (undeformed) region, the material points are coloured in green, whereas in deformed regions, they are coloured in red and indicate plastic deformations of the initial mass. The transition between the deformed and undeformed region marks the failure surface of the material. Experimental results are depicted by the blue dotted lines.

4.4 Computational efficiency: loop-based code versus vectorized code

We evaluate the computational efficiency of the MATLAB-based MPM solver, using the MATLAB version R2018a on an Intel Core i7-4790, with a benchmark based on the collapse of an aluminium-bar assemblage.

We vary the number of elements of the background mesh, which results in a variety of different mesh spacings h. The number of elements along the x-direction is $n_{el,x} = [20, 40, 80, 160, 320, 640]$, and the resulting number of elements in the y-direction is $n_{el,y} = [2, 5, 11, 23, 47, 95]$. The number of material points per element is kept constant, i.e., $n_{pe} = 3^2$, and the total number of material points is $n_p = [128, 465, 1830, 7260, 28'920, 115'440]$.

We calculate the average number of iterations per second (it/s) which corresponds to the number of calculation cycles during 1 second for an increasing total number of elements. We use this metric since it provides a clearer insight regarding the efficiency of the numerical solver, i.e., the more iterations there are, the more efficient the solver. Figure 14 shows an overall speed-up ratio of 20 reached by the vectorized solver compared to the iterative implementation. Such a gain is appreciable since

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

it results in a decrease of the runtime, i.e., approximately 40 hours for the iterative version, whereas the vectorized solution is achieved in 3 hours ($n_p = 115'440$).

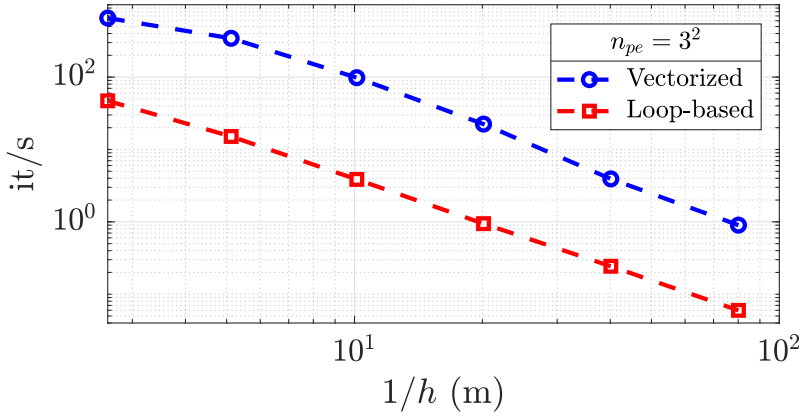

Figure 14. Number of iterations per second, i.e., the number of calculation cycles achieved in one second with respect to the mesh spacing h under the MATLAB version R2018a.

In addition, we also monitor the average time spent on the different functions called during a single execution cycle $\bar{t}_c$ (Fig. 15). When comparing time spent on the functions `p2Nsolve.m` and `mapN2p.m`, we report a speed up of 24 and 22, respectively. This difference is expected since these functions originally necessitate an extensive use of two nested for-loops to calculate i) the material point contribution or ii) the interpolation of updated nodal solutions.

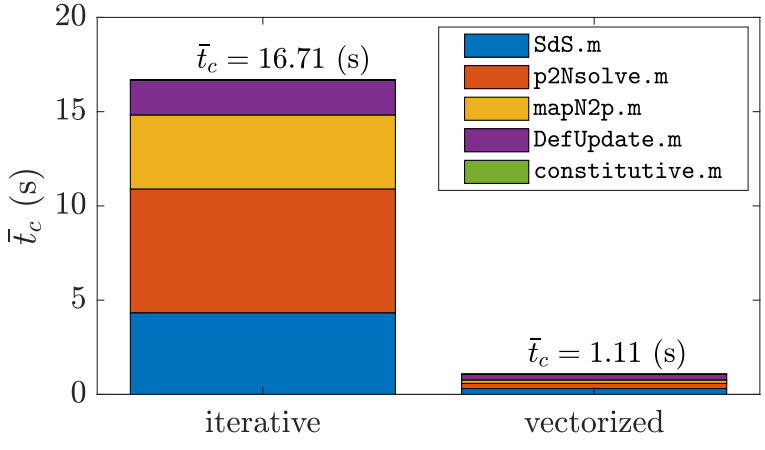

Figure 15. Average time $\bar{t}_c$ spent on the functions called during one calculation cycle for the iterative and the vectorized solver for $n_p = 115'440$.

We conclude with a direct comparison of a vectorized CPDI2q implementation of the collision of two elastic disks and the computational efficiency reported by Sinaie et al. (2017) of a Julia-based implementation. However, we note a difference

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

between the actual implementation and the one used by Sinaie et al. (2017); the latter is based on a USL variant with a cutoff algorithm, whereas our implementation relies on the MUSL (or double mapping) procedure, which necessitates a double mapping procedure. The initial geometry and parameters are the same as those used in Sinaie et al. (2017).

Our CPDI2q implementation ,in MATLAB R2018a, is, ad minima, 2.8 times faster than the Julia implementation proposed by Sinaie et al. (2017) for similar hardware (see Table 3). Sinaie et al. (2017) completed the analysis with an Intel Core i7-6700 (4 cores with a base frequency of 3.40 GHz up to a turbo frequency of 4.00 GHz) with 16 GB RAM, whereas we used an Intel Core i7-4790 with similar specifications (see section 2). However, the performance ratio between MATLAB and Julia seems to decrease as the mesh resolution increases.

Table 3. Efficiency comparison of the Julia implementation of Sinaie et al. (2017), and the MATLAB-based implementation for the two elastic disk impact problems.

mesh	n_{pe}	n_p	Its/s	
			Julia	MATLAB
20×20	2^2	416	132.80	224.45
20×20	4^2	1'624	33.37	81.07
40×40	2^2	1'624	26.45	77.61
80×80	4^2	25'784	1.82	4.57

4.5 Elasto-plastic slumping

We present an application of the MPM solver to the case of landslide mechanics. Considering a CPDI2q variant coupled to an elasto-plastic constitutive model based on a Mohr-Coulomb (M-C) non-associated plasticity Simpson (2017), we i) analyse the geometrical features of the slump and ii) compare the results (the geometry and the failure surface) to the numerical simulation of Huang et al. (2015), which is based on a Drucker-Prager model with tension cutoff (D-P).

Since an explicit scheme is used, we introduce a local damping in the equations of motion to resolve the equilibrium during a loading phase of 8 seconds, during which the gravity g is linearly ramped up to 9.81 m s^{-2}. The elasto-plastic behaviour is activated once this loading procedure is terminated, and the simulation proceeds during 7 additional seconds for a total simulation time of 15 seconds. The local damping coefficient is set to 0.1 since the latter damps the oscillations while preserving the dynamics Wang et al. (2016b).

The geometry of the problem is depicted in Fig. 16, the soil material is discretized by 110×35 elements with $n_{pe} = 3^2$, resulting in $n_p = 21'840$ material points; a mesh spacing $h = 1$ m and rollers are imposed at the left and right domain limits while a no-slip condition is enforced at the base of the material. The parameters used for the solution are shown in Table 4.

The numerical solution to the elasto-plastic problem is shown in Fig. 17. An intense shear zone, highlighted by the second invariant of the plastic strain ϵ_{II}, develops at the toe of the slope as soon as the material yields and it propagates backwards to the top of the material. It results in a rotational slump. The geometry and the failure surface reported by Huang et al. (2015) are highlighted by the continuous and the dotted lines respectively.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

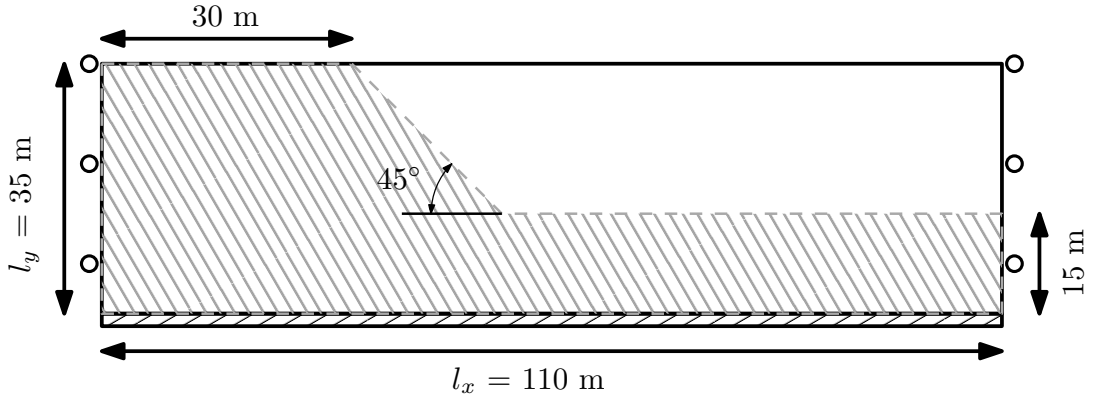

Figure 16. Initial geometry for the slump problem from Huang et al. (2015). Roller boundary conditions are imposed on the left and right of the domain while a no-slip condition is enforced at the base of the material.

Table 4. Parameters used for the elasto-plastic slump. The values of parameters are those found in Huang et al. (2015).

Parameter	Symbol	Value	Unit
Density	ρ	2100	kg m^{-3}
Poisson's ratio	ν	0.3	-
Elastic modulus	E	70	MPa
Cohesion	c	10	kPa
Internal friction angle	ϕ	20	°
Dilation angle	ψ	0	°

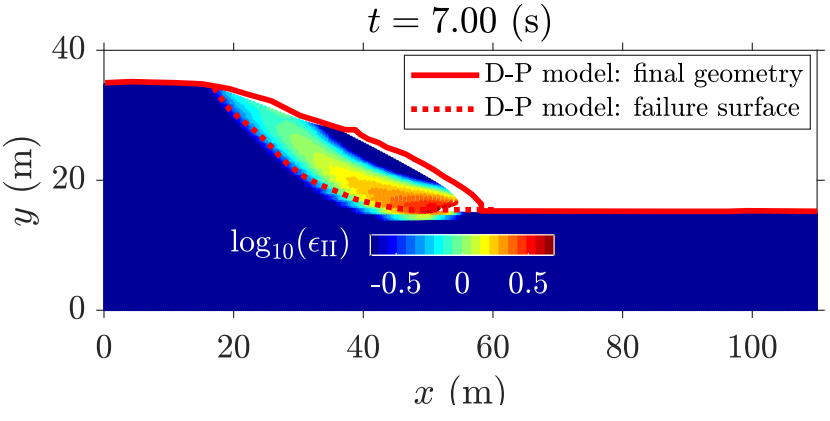

Figure 17. MPM solution to the elasto-plastic slump. The red lines indicate the solution obtained by Huang et al. (2015) and the coloured points indicate the second plastic strain invariant obtained with our CPDI2q solver. An intense shear zone progressively develops backwards from the toe of the slope, resulting in a circular failure mode.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

The maximal horizontal extent of the slump is smaller for the MPM solver with an M-C model, but the failure surface is in good agreement with the solution reported by Huang et al. (2015). They did not mention any use of a local damping in their implementation, and this can certainly explain the difference in the maximal extent. The local damping forces implemented in our solver should result in a smaller horizontal extent (because of the extra dissipation term introduced) but these should not affect the geometry of the shear band. Despite the horizontal extent, our results appear coherent with those reported by Huang et al. (2015).

5 Discussion

In this contribution, an efficient MPM solver is proposed that considers two variants (e.g., the cpGIMP and the CPDI/CPDI2q variant) under either an explicit or implicit

The efficiency derives from the combined use of the connectivity array `p2N` with the built-in function `accumarray()` to i) accumulate material point contributions to their associated nodes or ii) interpolate the updated nodal solutions to the associated material points in a vectorized manner. The efficiency is demonstrated by the speed-up values obtained for the evaluation of functions `p2nsolve.m` and `mapN2p.m`, which are 24 and 22 times faster, respectively, than an iterative approach that would require multiple nested for-loops. For the cantilever beam, the MATLAB-based solver is, ad minima, twice faster than a Julia implementation.

Regarding the elastic compaction of a one element-wide column, we report a good agreement of the numerical solver with previous explicit MPM implementations, such as Bardenhagen and Kober (2004). The same flaw of an explicit scheme is also experienced by the solver, i.e., a saturation of the error due to the specific used of an explicit scheme that resolves the wave propagation, thus preventing any static equilibrium to be reached. This confirms that our implementation is consistent with previous MPM implementations, especially for the implicit formulation where a quadratic convergence is resolved.

Regarding the cantilever beam problem, we report a good agreement of our solver with the results obtained by Sadeghirad et al. (2011). Furthermore, we report the vertical deflection of the beam to be very close in both magnitude and timing (for the CPDI2 variant) to the FEM solution. The beam almost recovered all the vertical deflection it experienced during the gravitational loading.

Since we compared both cpGIMPM and CPDI2 implementations for the cantilever beam problem, we observed a difference regarding the computational efficiency; the cpGIMP (e.g., it/s = 252.40) implementation is 3.5 times faster than the CPDI2 (e.g., it/s = 75.78) implementation for this case. This is mainly due to the extra cost of calculation of the basis functions and their derivatives. Additional computational resources are required to calculate these quantities, i.e., the basis function and their derivatives weights.

The elasto-plastic collapse demonstrates the abilities of the solver, considering both the number of iterations per seconds and its accuracy with respect to previous experimental and numerical results. However, this case indicates that extreme deformations are fatal for both cpGIMP and CPDI2q unless a domain update based on the determinant of the deformation gradient is chosen

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions

for the cpGIMP variant. Nevertheless, a splitting algorithm was proposed in Gracia et al. (2019); Homel et al. (2016), and it could be implemented to mitigate the material point domain stretch.

In addition, the computational efficiency reached by the solver is higher than expected and is even higher in the elastic case with respect to what was reported by Sinaie et al. (2017). This confirms the efficiency of MATLAB for solid mechanics problems, provided a reasonable amount of time is spent on the vectorization of the MPM algorithm.

The elasto-plastic slump also demonstrates the solver to be efficient in capturing complex dynamics in the field of geomechanics. Moreover, the CDPI2q solution showed that the algorithm proposed by Simpson (2017) returns stresses when the material yields and is well suited to the slumping mechanics.

However, as mentioned by Simpson (2017), such return mapping is valid only under the assumption of a non-associated plasticity with no volumetric plastic strain.

6 Conclusions

Our numerical investigations revealed that a domain-based approach in MPM fails when extremely large plastic deformations are involved, i.e., the elasto-plastic collapse. This can be avoided when a domain update based on the determinant of the deformation gradient is chosen.

We also have demonstrated the capability of MATLAB as an efficient language in regard to a material point method (MPM) implementation in an explicit or implicit formulation when bottleneck operations (e.g., calculations of the shape function or material point contributions) are vectorized. The MATLAB performances surpass those reached by an implementation in Julia, provided that built-in functions such as `accumarray( )` are used. However, the numerical efficiency naturally decreases with the level of complexity of the chosen MPM variant (sMPM, GIMPM or CPDI/CPDI2q).

The vectorization tasks we performed provide a fast and efficient MATLAB-based solver; however, the algorithmic structure could be transposed to a more efficient language, such as the C-CUDA language, that is known to efficiently take advantage of vectorized operations.

As a final word, a future implementation of a poro-elasto-plastic mechanical solver could be applied to complex geomechanical problems such as landslide dynamics while benefiting from a faster numerical implementation in C-CUDA, thus resolving high three-dimensional resolutions in an affordable amount a time.

Code availability. The fMPMM-solver developed in this study is licensed under the GPLv3 free software licence. The latest version of the code is available for download from Bitbucket at: https://bitbucket.org/ewyser/fmpmm-solver/src/master/ (last access: 29 May 2020) and https://github.com/ewyser/fMPMM-solver (last access: 29 May 2020). An fMPMM-solver archive is available from a permanent DOI repository (Zenodo) at: https://doi.org/10.5281/zenodo.3865422 (Wyser et al. (2020)). The fMPMM-solver software includes the reproducible codes used for this study.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions
Open Access
EGU

Appendix A: Acronyms

Table A1. Acronyms used throughout the manuscript

PIC	**P**article-**i**n-**C**ell
FLIP	**FL**uid **I**mplicit **P**article
FEM	**F**inite **E**lement **M**ethod
sMPM	**s**tandard **M**aterial **P**oint **M**ethod
GIMPM	**G**eneralized **M**aterial **P**oint **M**ethod
uGIMPM	**u**ndeformed **G**eneralized **M**aterial **P**oint **M**ethod
cpGIMPM	**c**ontiguous **p**article **G**eneralized **M**aterial **P**oint **M**ethod
CPDI	**C**onvected **P**article **D**omain **I**nterpolation
CPDI2q	**C**onvected **P**article **D**omain **I**nterpolation **2**nd order **q**uadrilateral

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Geoscientific Model Development Discussions
Open Access
EGU

Author contributions. EW wrote the manuscript and developed, together with YP, the numerical solver. MJ and YP supervised the early stages of the study and provided guidance. All authors have reviewed and approved the final version of the paper.

Competing interests. The authors declare that they have no conflicts of interest.

Acknowledgements. The authors gratefully thank Johan Gaume for his comments that contributed to improve the overall quality of the manuscript.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

References

Bandara, S. and Soga, K.: Coupling of soil deformation and pore fluid flow using material point method, Computers and Geotechnics, 63, 199–214, 2015.

Bandara, S., Ferrari, A., and Laloui, L.: Modelling landslides in unsaturated slopes subjected to rainfall infiltration using material point method, International Journal for Numerical and Analytical Methods in Geomechanics, 40, 1358–1380, 2016.

Bardenhagen, S., Barckbill, J., and Sulsky, D.: The material-point method for granular materials, Computer Methods in Applied Mechanics and Engineering, 187, 529–541, https://doi.org/10.1016/s0045-7825(99)00338-2, 2000.

Bardenhagen, S. G. and Kober, E. M.: The Generalized Interpolation Material Point Method, Computer Modeling in Engineering and Science, 5, 447–495, 2004.

Beuth, L., Benz, T., and Vermeer, P. A.: Large deformation analysis using a quasi-static material point method, Journal of Theoretical and Applied Mechanics, 38, 45–60, 2008.

Bui, H., Fukagawa, R., Sako, K., and Ohno, S.: Lagrangian meshfree particles method (SPH) for large deformation and failure flows of geomaterial using elastic-plastic soil constitutive model, International Journal for Numerical and Analytical Methods in Geomechanics, 32, 1537–1570, 2008.

Charlton, T. J., Coombs, W. M., and Augarde, C. E.: iGIMP: An implicit generalised interpolation material point method for large deformations, Computers and Structures, 190, 108–125, 2017.

Coombs, W. M. and Augarde, C. E.: AMPLE: A Material Point Learning Environment, Advances in Engineering Software, 139, 102 748, 2020.

Coombs, W. M., Charlton, T. J., Cortis, M., and Augarde, C. E.: Overcoming volumetric locking in material point methods, Computer Methods in Applied Mechanics and Engineering, 333, 1–21, 2018.

Coombs, W. M., Augarde, C. E., Brennan, A. J., Brown, M. J., Charlton, T. J., Knappett, J. A., Ghaffari Motlagh, Y., and Wang, L.: On Lagrangian Mechanics and the implicit material point method for large deformation elasto-plasticity, Computer Methods in Applied Mechanics and Engineering, 358, 112 622, 2020.

Dabrowski, M., Krotkiewski, M., and Schmid, D.: MILAMIN: MATLAB-based finite element method solver for large problems, Geochemistry, Geophysics, Geosystems, 9, 2008.

Dunatunga, A. and Kamrin, K.: Continuum modeling of projectile impact and penetration in dry granular media, Journal of the Mechanics and Physics of Solids, 100, 45–60, 2017.

Dunatunga, S. and Kamrin, K.: Continuum modelling and simulation of granualr flows through their many phases, Journal of Fluid Mechanics, 779, 483–513, 2015.

Fern, J., Rohe, A., Soga, K., and Alonso, E. E.: The Material Point Method for Geotechnical Engineering: A Practical Guide, CRC Press, 2019.

Gaume, J., Gast, T., Teran, J., van Herwijnen, A., and Jiang, C.: Dynamic anticrack propagation in snow, Nature Communications, 2018.

Gracia, F., Villard, P., and Richefeu, V.: Comparison of two numerical approaches (DEM and MPM) applied to unsteady flow, Computational Particle Mechanics, 6, 591–609, 2019.

Guilkey, J. E. and Wiess, J. A.: Implicit time integration for the material point method: Quantitative and algorithmic comparisons with the finite element method, International Journal for Numerical Methods in Engineering, 2003.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Homel, M. A., Brannon, R. M., and Guilkey, J.: Controlling the onset of numerical fracture in parallelized implementations of the material point method (MPM) with convective particle domain interpolation (CPDI) domain scaling, International Journal for Numerical Methods in Engineering, 107, 31–48, 2016.

Huang, P., Li, S. L., Guo, H., and Hao, Z. M.: Large deformation failure analysis of the soil slope based on the material point method, Computational Geosciences, 19, 951–963, 2015.

Iaconeta, I., Larese, A., Rossi, R., and Guo, Z.: Comparison of a Material Point Method and a Galerkin Meshfree Method for the Simulation of Cohesive-Frictional Materials, Materials, 10, 1150, 2017.

Leavy, R. B., Guilkey, J. E., Phung, B. R., Spear, A. D., and Brannon, R. M.: A convected-particle tetrahedron interpolation technique in the material-point method for the mesoscale modeling of ceramics, Computational Mechanics, 64, 563–583, 2019.

Nairn, J. A.: Material point method calculations with explicit cracks, Computer Modeling in Engineering and Sciences, 4, 649–663, 2003.

Sadeghirad, A., Brannon, R. M., and Burghardt, J.: A convected particle domain interpolation tehcnique to extend applicability of the material point method for problems involving massive deformations, International Journal for Numerical Methods in Engineering, 86, 1435–1456, 2011.

Sadeghirad, A., Brannon, R. M., and Guilkey, J. E.: Second-order convected particle domain interpolation (CPDI2) with enrichment for weak discontinuities at material interfaces, Internation Journal for Numerical Methods in Engineering, 95, 928–952, 2013.

Simpson, G.: Practical Finite Element Modelling in Earth Science using MatLab, Wiley, 2017.

Sinaie, S., Nguyen, V. P., Nguyen, C. T., and Bordas, S.: Programming the material point method in Julia, Advances in Engineering Software, 105, 17–29, 2017.

Steffen, M., Kirby, R. M., and Berzins, M.: Analysis and reduction of quadrature errors in the material point method (MPM), International Journal for Numerical Methods in Engineering, 76, 922–948, 2008a.

Steffen, M., Wallstedt, P. C., Guilkey, J. E., Kirby, R. M., and Berzins, M.: Examination and Analysis of Implementation Choices within the Material Point Method (MPM), Computer Modeling in Engineering and Science, 31, 107–127, 2008b.

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., and Selle, A.: A material point method for snow simulation, ACM Transactions on Graphics, 32, 2013.

Sulsky, D., Chen, Z., and Schreyer, H. L.: A particle method for history-dependent materials, Computer Methods in Applied Mechanics and Engineering, 118, 179–196, 1994.

Sulsky, D., Zhou, S.-J., and Schreyer, H. L.: Application of a particle-in-cell method to solid mechanics, Computer Physics Communications, 87, 236–252, 1995.

Vardon, P. J., Wang, B., and Hicks, M. A.: Slope Failure Simulations with MPM, Procedia Engineering, 175, 258–264, 2017.

Wang, B., Hicks, M. A., and Vardon, P. J.: Slope failure analysis using the random material point method, Géotechnique Letters, pp. 113–118, 2016a.

Wang, B., Vardon, P. J., Hicks, M. A., and Chen, Z.: Development of an implicit material point method for geotechnical applications, Computers and Geotechnics, 71, 159–167, 2016b.

Wang, L., Coombs, W. M., Augarde, C. E., Cortis, M., Charlton, T. J., Brown, M. J., Knappett, J., Brennan, R.M., D. C., Richards, D., and Blake, A.: On the use of domain-based material point methods for problems involving large distortion, Computer Methods in Applied Mechanics and Engineering, 255, 1003–1025, 2019.

Wieckowski, Z.: The material point method in large strain engineering problems, Computer Methods in Applied Mechanis and Engineering, 193, 4417–4438, 2004.

https://doi.org/10.5194/gmd-2020-183
Preprint. Discussion started: 20 July 2020
© Author(s) 2020. CC BY 4.0 License.

Wyser, E., Jaboyedoff, M., and Podladchikov, Y.: fMPMM-solver, https://doi.org/http://doi.org/10.5281/zenodo.3865422, 2020.

York, A. R., Sulsky, D., and Schreyer, H. L.: The material point method for simulation of thin membranes, International Journal for Numerical Methods in Engineering, 44, 1429–1456, https://doi.org/10.1002/(sici)1097-0207(19990410)44:10<1429::aid-nme536>3.0.co;2-4, 1999.

Zhang, X., Chen, Z., and Liu, Y.: The Material Point Method: A Continuum-Based Particle Method For Extreme Loading Cases, Elsevier, 2017.