Material Point Method Simulations using an Approximate Full Mass Matrix Inverse

John A. Nairn[a,*], Chad C. Hammerquist[b]

[a] *Wood Science and Engineering, Oregon State University, Corvallis, OR 97330, USA*
[b] *FracGeo, Woodlands, TX 77380, USA*

Abstract

All material point method (MPM) codes approximate the full mass matrix with a lumped mass matrix. Because this approach causes dissipation, most MPM simulations rely on so-called FLIP methods to limit dissipation. Recent work to deal with noise caused by those FLIP methods derived the XPIC method (for extended particle in cell method) that filters null space noise from particle velocities using a projection operator. This paper shows that the XPIC projection operator is equivalent to doing grid calculations using an asymptotic expansion of the full mass matrix inverse. From that insight, we derived FMPM(k) for full mass matrix MPM of order k (where the mass matrix inverse is expanded to k terms). Compared to prior MPM algorithms, FMPM changes the methods used to update particle velocities, positions, stresses, and strains. Several examples show that FMPM is more stable and accurate, has less dissipation than XPIC, and with high enough k has less dissipation than FLIP methods. One challenge in MPM is imposing velocity conditions on the grid and those challenges are amplified in FMPM. This paper describes a new moving-wall approach that improves grid boundary conditions and is beneficial to both FMPM and prior MPM. Finally, the use of FMPM for multimaterial mode MPM, with explicit cracks, and with affine extrapolation options is each discussed.

Keywords: Material Point Method, Full Mass Matrix, Computational Mechanics

1. Introduction

The weak, variational form used in finite element analysis [1, 2] and in explicit, material point method (MPM) simulations [3, 4] results in a discretized momentum equation on the grid of:

$$\tilde{\mathbf{m}}\boldsymbol{a} = \boldsymbol{f} + \boldsymbol{b}$$

where $\tilde{\mathbf{m}}$ is the full mass matrix, and $\boldsymbol{a}$, $\boldsymbol{f}$, and $\boldsymbol{b}$ are vectors of acceleration, force, and body force on each node. Although $\tilde{\mathbf{m}}$ is a symmetric, banded matrix, its $n \times n$ size (where n is number of nodes in the grid), is typically too large to store and invert during simulations with many time steps. The usual practice is to replace it with a lumped mass matrix [1–4], $\mathbf{m}$, defined by:

$$\mathbf{m} = \text{diag}(\boldsymbol{m}) \qquad \text{where} \qquad m_i = \sum_j \tilde{\mathbf{m}}_{ij}$$

The momentum equation then has the trivial solution:

$$\boldsymbol{a} = \mathbf{m}^{-1}(\boldsymbol{f} + \boldsymbol{b}) \qquad \Longrightarrow \qquad a_i = \frac{f_i + b_i}{m_i}$$

Despite its common use, lumped mass matrices cause detrimental energy dissipation (although some dissipation might be beneficial [1]).

*Corresponding author: john.nairn@oregonstate.edu, Tel: +1-541-737-4265, Fax: +1-541-737-3385

Our recent work on enhancing MPM stability led to the extended particle-in-cell method termed XPIC [5]. In brief, XPIC is implemented by filtering particle velocities using a projection operator, $\mathbf{P}_X$. This operator projects current particle velocities onto the non-null-space velocities associated with MPM extrapolation matrices (*i.e.*, it removes null-space noise) [5]. Here we show that XPIC filtering is mathematically equivalent to performing grid calculations using an approximate full mass matrix inverse. XPIC(k) is an MPM option that uses a k-term expansion of $\mathbf{P}_X$ [5]. This paper defines FMPM(k) as a new MPM option that uses a k-term expansion of the full mass matrix inverse whenever beneficial. The key changes in FMPM were to revise update methods used for particle positions, velocities, stresses, and strains. The results and discussion section shows that FMPM is stable and accurate, always has less dissipation than XPIC, and, with enough terms k, has less dissipation than prior MPM options.

2. FMPM Algorithm

2.1. MPM Extrapolations

Objects are modeled in MPM by two views — a Lagrangian view (particles or material points) and an Eulerian view (grid) [3, 4, 6]. Each time step maps information from the particles to the grid, updates grid values, and then maps updated information back to the particles, Particle quantities are denoted here with upper case vectors (*e.g.*, $\boldsymbol{V}$ for particle velocities) of length N where N is number of particles, while grid quantities are denoted with lower case vectors (*e.g.*, $\boldsymbol{v}$ for grid velocities) of length n where n is number of nodes. Subscripted vectors (*e.g.*, $\boldsymbol{V}_p$ or $\boldsymbol{v}_i$) indicate a quantity on a specific particle or node.

Linear mappings from particles to grid and from grid to particles are written as:

$$\boldsymbol{V} = \mathbf{S}\boldsymbol{v} \qquad \text{and} \qquad \boldsymbol{v} = \mathbf{S}^{+}\boldsymbol{V}$$

Here $\mathbf{S}$ is an $N \times n$ matrix that interpolates grid values to particle locations. In generalized interpolation MPM (or GIMP), the $\mathbf{S}$ elements convolute grid shape functions, $N_i(\boldsymbol{x})$, with particle shape functions, $\chi_p(\boldsymbol{x})$ [6]:

$$S_{pi} = \frac{\int \chi_p(\boldsymbol{x}) N_i(\boldsymbol{x}) d\boldsymbol{x}}{\int \chi_p(\boldsymbol{x}) d\boldsymbol{x}} = \frac{1}{V_p}\int_{V_p} N_i(\boldsymbol{x}) d\boldsymbol{x} \quad \left[\approx \frac{1}{V_p}\int_{V_p}\left(\sum_{c=1}^{n_c} M_c(\boldsymbol{x}) N_i(\boldsymbol{x}_c)\right) d\boldsymbol{x}\right]$$

The $N_i(\boldsymbol{x})$ functions are typically linear or quadratic spline [7] shape functions. The second form assumes that $\chi_p(\boldsymbol{x}) = 1$ within the particle domain and zero elsewhere such that the denominator is particle volume, V_p, and the numerator integrates grid shape function over the particle domain. Full, or "Finite" GIMP integrates over current particle domain, while uGIMP (for "undeformed" GIMP) integrates over initial particle domain translated to the current particle position [6]. The bracketed equation is the convected particle domain integration (or CPDI) method that approximates full GIMP by expanding $N_i(\boldsymbol{x})$ within the particle domain using linear, isoparametric, finite element shape functions on the deformed particle domain ($M_c(\boldsymbol{x})$ from 1 to n_c corners in particle domain) [8, 9]. Here uGIMP and CPDI imply shape functions using linear $N_i(\boldsymbol{x})$ while B2GIMP and B2CPDI imply shape functions calculated using quadratic spline $N_i(\boldsymbol{x})$ [7].

The ideal mapping from particles to the grid would invert $\mathbf{S}$, but that is not possible because $\mathbf{S}$ is not a square matrix. In general, MPM simulations have more particles than active grid nodes ($N > n$). The next ideal option would set $\mathbf{S}^{+}$ to the Moore-Penrose inverse, but that calculation is computationally too intensive. Instead, MPM uses a reverse mapping (and here the Moore-Penrose nomenclature of superscript "+" is adopted for that reverse mapping) derived by least squares minimization of mass-weighted, velocity extrapolation error [3]:

$$\boldsymbol{v} = \tilde{\mathbf{m}}^{-1}\mathbf{S}^{T}\mathbf{M}\boldsymbol{V} \qquad \text{with} \qquad \tilde{\mathbf{m}} = \mathbf{S}^{T}\mathbf{M}\mathbf{S} \qquad \left(i.e,\ (\tilde{\mathbf{m}})_{ij} = \sum_p M_p S_{ip}^{T} S_{pj}\right)$$

Here $\tilde{\mathbf{m}}$ is the $n \times n$, symmetric full mass matrix and $\mathbf{M}$ is an $N \times N$ diagonal matrix with particle mass M_p on the p^{th} diagonal. The resulting reverse mapping of particle velocity to the grid becomes:

$$\boldsymbol{v} = \tilde{\mathbf{S}}^{+}\boldsymbol{V} \quad \Longrightarrow \quad \tilde{\mathbf{S}}^{+} = \tilde{\mathbf{m}}^{-1}\mathbf{S}^{T}\mathbf{M}$$

Even this reverse mapping is impractical because it needs to invert the full mass matrix. The standard MPM reverse mapping replaces the full mass matrix, $\tilde{\mathbf{m}}$, with a lumped mass matrix [3, 4], $\mathbf{m}$, resulting in:

$$\mathbf{S}^+ = \mathbf{m}^{-1}\mathbf{S}^T\mathbf{M} \qquad \text{or} \qquad S^+_{ip} = \frac{M_p S_{pi}}{m_i} \qquad \text{and} \qquad \mathbf{m} = \text{diag}(\mathbf{S}^T\mathbf{M})$$

Note that a tilde on $\tilde{\mathbf{m}}$ and $\tilde{\mathbf{S}}^+$ denotes full mass matrix and its associated reverse mapping while $\mathbf{m}$ and $\mathbf{S}^+$ denote lumped mass matrix and its associated reverse mapping.

Besides quantity extrapolations, MPM needs to extrapolate quantity gradients. For example, to update stresses and strains on particles, MPM extrapolates velocity gradient from the grid using:

$$\nabla \boldsymbol{V}^T = \mathbf{G}\boldsymbol{v} \qquad \text{or} \qquad \nabla \boldsymbol{V}_p = \sum_i \boldsymbol{v}_i^T \boldsymbol{G}_{pi}$$

where $\boldsymbol{G}_{pi}$ are GIMP gradient shape functions found by convolution with $\nabla N_i(\boldsymbol{x})$ [6]. The GIMP (using $\chi_p(\boldsymbol{x}) = 1$) and CPDI (in brackets) forms for the gradient are:

$$\boldsymbol{G}_{pi} = \frac{1}{V_p}\int_{V_p} \nabla N_i(\boldsymbol{x}) d\boldsymbol{x} \quad \left[\approx \frac{1}{V_p}\int_{V_p}\left(\sum_{c=1}^{n_c} \nabla M_c(\boldsymbol{x}) N_i(\boldsymbol{x}_c)\right) d\boldsymbol{x}\right]$$

The elements of $\mathbf{G}$ and $\boldsymbol{v}^T$ are column vectors while $\boldsymbol{v}$ are row vectors. Thus $\nabla \boldsymbol{V}_p$ is tensor found by extrapolating outer products of $\boldsymbol{v}_i$ and $\boldsymbol{G}_{pi}$. When written in analogy with other linear mappings, the elements of $\nabla \boldsymbol{V}^T$ are transposes of individual particle velocity gradients. Note that the shape function and shape function gradient matrices are large and sparse. The non-zero elements are calculated whenever needed rather than storing them in memory.

2.2. Approximate Full Mass Matrix Inversion and Equivalence to XPIC

The full mass matrix can be rewritten using lumped mass $\mathbf{S}^+$ as:

$$\tilde{\mathbf{m}} = \mathbf{m}\mathbf{S}^+\mathbf{S} = \mathbf{m}\big(\mathbf{I}_n - (\mathbf{I}_n - \mathbf{S}^+\mathbf{S})\big) = \mathbf{m}\big(\mathbf{I}_n - \mathbf{A}\big)$$

where $\mathbf{I}_n$ is an $n \times n$ identity matrix and $\mathbf{A} = \mathbf{I}_n - \mathbf{S}^+\mathbf{S}$. Because $\mathbf{A}$ should be small (when a good round-trip mapping, $\mathbf{S}^+\mathbf{S}$, approximates $\mathbf{I}_n$), the full mass matrix inverse can be expanded in a Taylor series:

$$\tilde{\mathbf{m}}^{-1} = \big(\mathbf{I}_n - \mathbf{A}\big)^{-1}\mathbf{m}^{-1} = (\mathbf{I}_n + \mathbf{A} + \mathbf{A}^2 + \mathbf{A}^3 + \cdots)\mathbf{m}^{-1}$$

We define $\tilde{\mathbf{m}}_k^{-1}$ as the full mass matrix inverse expanded to k terms (last term is $\mathbf{A}^{k-1}$) and FMPM(k) as a new approach to MPM that replaces $\mathbf{m}^{-1}$ with $\tilde{\mathbf{m}}_k^{-1}$ whenever beneficial. Summing expansions of each $\mathbf{A}^j$ term gives:

$$\tilde{\mathbf{m}}_k^{-1} = \left(k\mathbf{I}_n + \sum_{\ell=1}^{k-1}\sum_{j=\ell}^{k-1}(-1)^\ell \binom{j}{\ell} \big(\mathbf{S}^+\mathbf{S}\big)^\ell\right)\mathbf{m}^{-1} = \left(\sum_{\ell=1}^{k}(-1)^{\ell+1}\binom{k}{\ell}\big(\mathbf{S}^+\mathbf{S}\big)^{\ell-1}\right)\mathbf{m}^{-1} \tag{1}$$

where elimination of one sum used

$$\binom{k}{\ell} = \sum_{j=\ell}^{k}\binom{j-1}{\ell-1}$$

which is easily proved by reverse induction.

The XPIC(k) derivation in Ref. [5] resulted in an expansion to the non-null-space projection operator, $\mathbf{P}_X$. If $\mathbf{S}^+$ was the actual Moore-Penrose inverse, the projection operator would be:

$$\mathbf{P}_{\text{MP}} = \mathbf{S}\mathbf{S}^+ = \mathbf{I}_N - (\mathbf{I}_N - \mathbf{S}\mathbf{S}^+)$$

where $\mathbf{I}_N$ is the $N \times N$ identity matrix. When $\mathbf{S}^+$ is lumped-mass MPM reverse mapping matrix, Ref. [5] showed that

$$\mathbf{P}_X = \mathbf{I}_N - (\mathbf{I}_N - \mathbf{S}\mathbf{S}^+)^k$$

Table 1: MPM

Task	Calculations	$\tilde{\mathbf{m}}_k^{-1}$ Cost
1. Grid Mass and Momenta	$\boldsymbol{p} = \mathbf{S}^T\mathbf{M}\boldsymbol{V},\ \ \boldsymbol{m} = \mathbf{S}^T\boldsymbol{M}$	0
2. Grid Forces	$\boldsymbol{f} = -\mathbf{G}^T\mathbf{V}^{(0)}\boldsymbol{\tau} + \mathbf{S}^T\mathbf{M}\boldsymbol{B}$	0
3. Update grid values	$\boldsymbol{p}^+ = \boldsymbol{p} + \boldsymbol{f}\Delta t,\ \ \boldsymbol{v}^+ = \tilde{\mathbf{m}}_k^{-1}\boldsymbol{p}^+$	$N(k-1)C_x$
4. Particle update	$\boldsymbol{V}^{(n+1)} = \mathbf{S}\boldsymbol{v}^+$ $\boldsymbol{X}^{(n+1)} = \boldsymbol{X}^{(n)} + \frac{\boldsymbol{V}^{(n+1)}+\boldsymbol{V}^{(n)}}{2}\Delta t$	0
5. Particle Stress and Strain	$\nabla\boldsymbol{V}^T = \mathbf{G}\boldsymbol{v}^+,\ \ d\mathbf{F}_p = \exp\left(\nabla\boldsymbol{V}_p\Delta t\right)$	0

converges to orthogonal null space removal as $k \to \infty$ [5]. Expanding $(\mathbf{I}_N - \mathbf{S}\mathbf{S}^+)^k$, the XPIC($k$) projection applied to particle velocity becomes [5]:

$$\mathbf{P}_X\boldsymbol{V} = \mathbf{S}\left(\sum_{\ell=1}^{k}(-1)^{\ell+1}\binom{k}{\ell}(\mathbf{S}^+\mathbf{S})^{\ell-1}\right)\mathbf{S}^+\boldsymbol{V} = \mathbf{S}\tilde{\mathbf{m}}_k^{-1}\boldsymbol{p}$$

where $\boldsymbol{p} = \mathbf{S}^T\mathbf{M}\boldsymbol{V}$ are grid momenta. In other words, the XPIC(k) projection to remove null space noise on the particles is equivalent to first converting grid momenta to grid velocities using an approximate full mass matrix inverse, $\boldsymbol{v} = \tilde{\mathbf{m}}_k^{-1}\boldsymbol{p}$, and then mapping those grid velocities to the particles using $\mathbf{S}$. The calculations needed to implement XPIC(k) are thus identical to those needed for full mass matrix methods. The joint algorithm is implemented using a recursion method [5]:

$$\mathbf{P}_X\boldsymbol{V} = \mathbf{S}\sum_{\ell=1}^{k}(-1)^{\ell+1}\boldsymbol{v}_\ell^* \quad \text{and} \quad \tilde{\mathbf{m}}_k^{-1}\boldsymbol{p} = \sum_{\ell=1}^{k}(-1)^{\ell+1}\boldsymbol{v}_\ell^*$$

where

$$\boldsymbol{v}_\ell^* = \binom{k}{\ell}(\mathbf{S}^+\mathbf{S})^{\ell-1}\mathbf{m}^{-1}\boldsymbol{p} = \frac{k+1-\ell}{\ell}\mathbf{S}^+\mathbf{S}\boldsymbol{v}_{\ell-1}^* \quad \text{and} \quad \boldsymbol{v}_1^* = k\mathbf{m}^{-1}\boldsymbol{p}$$

2.3. Core FMPM Time Step Tasks

This section examines each core MPM task (see Table 1). It does not derive MPM (that is available in many prior papers), but rather expresses each task using mapping matrices defined above and then modifies them when appropriate for FMPM. Although the tasks in Table 1 are needed in all FMPM calculations, most simulations will include additional tasks such as boundary conditions. Boundary conditions are discussed in detail; additional features such as contact, cracks, and affine methods are mentioned briefly.

Tasks 1 and 2: Grid Extrapolations

The first two tasks extrapolate particle momenta ($\boldsymbol{P} = \mathbf{M}\boldsymbol{V}$), mass ($\boldsymbol{M}$), Kirchhoff stress ($\boldsymbol{\tau}$), and specific body forces ($\boldsymbol{B}$) to grid momenta ($\boldsymbol{p}$), lumped mass ($\boldsymbol{m}$), and force ($\boldsymbol{f}$). The lumped mass matrix is $\mathbf{m} = \text{diag}(\boldsymbol{m})$. The stress extrapolation uses $\mathbf{V}^{(0)}$, which is a diagonal matrix with undeformed, initial particle volume $V_p^{(0)}$ on the p^{th} diagonal. The GIMP derivation naturally leads to stress extrapolation as volume-weighted Cauchy stress or $V_p\boldsymbol{\sigma}_p$ [6]. By using $V_p = JV_0$ and $\boldsymbol{\sigma}_p = \tau_p/J$, where J is determinant of the deformation gradient, $V_p\boldsymbol{\sigma}_p$ can be replaced by $V_0\boldsymbol{\tau}_p$ thereby avoiding calculation of deformed volume on every time step. These standard MPM tasks are unchanged in FMPM. Despite the lumped mass matrix calculation, these steps do not use that matrix. The diagonal elements of $\mathbf{m}$ (stored in $\boldsymbol{m}$), however, are needed later when finding $\tilde{\mathbf{m}}_k^{-1}$.

Task 3: Update Grid Momenta and Calculate Grid Velocities

Updating nodal momenta to $\boldsymbol{p}^+$ is a trivial task and identical to standard MPM. FMPM differs when those momenta are converted to grid velocities. Rather than using a lumped mass matrix, it finds updated nodal velocities using $\boldsymbol{v}^+ = \tilde{\mathbf{m}}_k^{-1}\boldsymbol{p}^+$. This full-mass matrix calculation is FMPM's main computational cost. Because $\tilde{\mathbf{m}}_k^{-1}$ calculations used in FMPM(k) are equivalent to XPIC calculations, the extra cost is the same

as for XPIC — $N(k-1)C_x$ where C_x is time needed for one of the $k-1$ loops in XPIC calculations [5]. Note that linear scaling in N is more favorable than n^3 scaling to invert the full mass matrix.[†]

Task 4: Particle Updates

FMPM changes updates for particle velocity and position. We start with generic, explicit updates:

$$\begin{aligned} \boldsymbol{V}^{(n+1)} &= \boldsymbol{V}^{(n)} + \mathbb{A}^{(n)}\Delta t \\ \boldsymbol{X}^{(n+1)} &= \boldsymbol{X}^{(n)} + \mathbb{V}^{(n)}\Delta t + \frac{1}{2}\mathbb{A}^{(n)}(\Delta t)^2 \end{aligned}$$

where $\mathbb{A}^{(n)}$ and $\mathbb{V}^{(n)}$ are effective particle acceleration and initial velocity for the n^{th} time step. Every explicit MPM particle update method can be reduced to $\mathbb{A}^{(n)}$ and $\mathbb{V}^{(n)}$ using:

$$\mathbb{A}^{(n)}\Delta t = \boldsymbol{V}^{(n+1)} - \boldsymbol{V}^{(n)} \quad \text{and} \quad \mathbb{V}^{(n)}\Delta t = \boldsymbol{X}^{(n+1)} - \boldsymbol{X}^{(n)} - \frac{\boldsymbol{V}^{(n+1)} - \boldsymbol{V}^{(n)}}{2}\Delta t$$

Because $\mathbb{V}^{(n)}$ and $\mathbb{A}^{(n)}$ are assumed constant during one time step, the average Lagrangian (or material) velocities for the particles can be found from either velocity or position updates as:

$$\frac{\Delta \boldsymbol{X}}{\Delta t} = \mathbb{V}^{(n)} + \frac{1}{2}\mathbb{A}^{(n)}\Delta t \quad \text{and} \quad \langle \boldsymbol{V} \rangle = \boldsymbol{V}^{(n)} + \frac{1}{2}\mathbb{A}^{(n)}\Delta t$$

These two Lagrangian velocities differ by $\mathbb{V}^{(n)} - \boldsymbol{V}^{(n)}$. This velocity inconsistency can be eliminated by choosing $\mathbb{V}^{(n)} = \boldsymbol{V}^{(n)}$, but this option is not used in prior MPM methods. For example, a standard FLIP update method[‡] is [3]:

$$\boldsymbol{V}^{(n+1)} = \boldsymbol{V}^{(n)} + \mathbf{S}\mathbf{m}^{-1}\boldsymbol{f}\Delta t \quad \text{and} \quad \boldsymbol{X}^{(n+1)} = \boldsymbol{X}^{(n)} + \mathbf{S}\mathbf{m}^{-1}\boldsymbol{p}\Delta t + \frac{1}{2}\mathbf{S}\mathbf{m}^{-1}\boldsymbol{f}\Delta t$$

which correspond to effective acceleration and velocity of:

$$\mathbb{A}^{(n)}\Delta t = \mathbf{S}\mathbf{m}^{-1}\boldsymbol{f}\Delta t \quad \text{and} \quad \mathbb{V}^{(n)} = \mathbf{S}\mathbf{m}^{-1}\boldsymbol{p}$$

Because $\mathbb{V}^{(n)} \neq \boldsymbol{V}^{(n)}$, this update has velocity inconsistencies. Changing FLIP updates to use $\mathbb{V}^{(n)} = \boldsymbol{V}^{(n)}$ would change only the position update, but that approach gives very poor results (likely due to potential noise in particle velocities). Brackbill [11] maintains that $\mathbb{V}^{(n)} = \mathbf{S}\mathbf{m}^{-1}\boldsymbol{p}$ is required in FLIP such that velocity for position update and acceleration for velocity update are extrapolated to the particles by the same method.

The FMPM update proposed here in Table 1 corresponds to effective acceleration and velocity of

$$\mathbb{A}^{(n)}\Delta t = \mathbf{S}\tilde{\mathbf{m}}_k^{-1}\boldsymbol{p}^+ - \boldsymbol{V}^{(n)} \qquad \text{and} \qquad \mathbb{V}^{(n)} = \boldsymbol{V}^{(n)}$$

Note that this update does choose $\mathbb{V}^{(n)} = \boldsymbol{V}^{(n)}$, resulting in consistent Lagrangian velocities. This option is stable in FMPM, because the full mass matrix calculations filter out noise in particle velocities [5]. Because FMPM simply redefines effective acceleration, its particle updates incur no extra computational cost. The updates do depend on $\boldsymbol{v}^+ = \tilde{\mathbf{m}}_k^{-1}\boldsymbol{p}^+$, but that extra cost was included above for task 3.

Prior to recognizing that XPIC(k) calculations are equivalent to approximating the full mass matrix, the recommended XPIC(k) updates (rewritten in full mass matrix notation) were [5]:

$$\boldsymbol{V}^{(n+1)} = \mathbf{S}\left(\tilde{\mathbf{m}}_k^{-1}\boldsymbol{p} + \mathbf{m}^{-1}\boldsymbol{f}\Delta t\right) \quad \text{and} \quad \boldsymbol{X}^{(n+1)} = \boldsymbol{X}^{(n)} + \left[\mathbf{S}\mathbf{m}^{-1}\boldsymbol{p} + \frac{\boldsymbol{V}^{(n+1)} - \boldsymbol{V}^{(n)}}{2}\right]\Delta t$$

[†]Scaling to invert the banded, symmetric mass matrix could potentially improve to $b^2 n$, where b is bandwidth, but bandwidth would depend on node numbering [10] and scaling would still be less favorable than linear in N.

[‡]Many codes use first order position update, $\boldsymbol{X}^{(n+1)} = \boldsymbol{X}^{(n)} + \mathbf{S}\mathbf{m}^{-1}\boldsymbol{p}^+\Delta t$, but here second order is used because it leads to simpler form for $\mathbb{V}^{(n)}$.

The implied effective terms are:

$$\mathbb{A}^{(n)}\Delta t = \mathbf{S}\left(\tilde{\mathbf{m}}_k^{-1}\boldsymbol{p} + \mathbf{m}^{-1}\boldsymbol{f}\Delta t\right) - \boldsymbol{V}^{(n)} \quad \text{and} \quad \mathbb{V}^{(n)} = \mathbf{S}\mathbf{m}^{-1}\boldsymbol{p}$$

This process reduces noise, but it retains a Lagrangian velocity inconsistency. The two differences between FMPM and XPIC updates are that FMPM finds acceleration using $\tilde{\mathbf{m}}_k^{-1}\boldsymbol{f}$ instead of $\mathbf{m}^{-1}\boldsymbol{f}$ and that $\mathbb{V}^{(n)}$ is changed to $\boldsymbol{V}^{(n)}$ to eliminate velocity inconsistencies. The FMPM and XPIC velocity updates can also be compared using the XPIC projection operator:

$$\boldsymbol{V}^{(n+1)} = \begin{cases} \mathbf{P}_X\boldsymbol{V} + \mathbf{S}\mathbf{m}^{-1}\boldsymbol{f}\Delta t & \text{XPIC} \\ \mathbf{P}_X\left(\boldsymbol{V} + \mathbf{S}\mathbf{m}^{-1}\boldsymbol{f}\Delta t\right) & \text{FMPM} \end{cases}$$

In other words, XPIC removes null space noise from $\boldsymbol{V}$ using $\mathbf{P}_X\boldsymbol{V}$ and then updates those velocities by FLIP methods while FMPM updates by FLIP first and then removes null space noise from that result.

Defining particle-in-cell or PIC methods as methods that replace particle velocities with updated, lumped-mass-matrix velocities extrapolated from the grid, FMPM(1), XPIC(1) [5], and a prior MPM PIC update method [12] are three different PIC methods. They all have the same effective acceleration, $\mathbb{A}^{(n)}\Delta t = \mathbf{S}\mathbf{m}^{-1}\boldsymbol{p}^{+} - \boldsymbol{V}^{(n)}$, but they differ in effective velocities (which affects only their position updates):

$$\mathbb{V}^{(n)} = \begin{cases} \boldsymbol{V}^{(n)} & \text{FMPM(1)} \\ \mathbf{S}\mathbf{m}^{-1}\boldsymbol{p} & \text{XPIC(1)} \\ \frac{1}{2}(\boldsymbol{V}^{(n+1)} + \boldsymbol{V}^{(n)}) & \text{MPM PIC} \end{cases} \tag{2}$$

Here MPM PIC is using a first-order FLIP position update ($\Delta\boldsymbol{X} = \mathbf{S}\mathbf{m}^{-1}\boldsymbol{p}^{+}\Delta t = \boldsymbol{V}^{(n+1)}\Delta t$). Orthogonal cutting simulations show that XPIC(1) updates are a significant improvement over MPM PIC updates [13, 14]. An example below quantifies those differences and shows that FMPM(1) provides further improvement.

Task 5: Stress and Strain Update

The final task is to update particle stresses and strains using the constitutive law for any implemented material type. This update first maps velocity gradient from the grid to the particles using $\mathbf{G}$. The particle velocity gradient, $\nabla\boldsymbol{v}_p$, is used to evaluate an incremental deformation gradient, $d\mathbf{F}_p = \exp(\nabla\boldsymbol{v}_p\Delta t)$. The matrix exponential can be efficiently expanded to multiple terms using the Cayley-Hamilton theorem [9, 15]. A change in FMPM is that $\nabla\boldsymbol{v}_p$ is found by extrapolating grid velocities found with a full mass matrix rather than a lumped mass matrix.

Table 1 inserts this update immediately after the particle update suggesting the name USL for update stresses (and strains) last. Beside USL, several other stress updates are used. The update can be done immediately after grid extrapolations (USF for update stresses first [16]), or can be done in both places using half the time step for each like a midpoint rule (USAVG [17]). Another variation is to re-extrapolate updated particle momenta to the grid prior to updates done after the particle updates [18] (converting USL and USAVG methods to USL+ or USAVG+ methods). In FMPM, each stress update must be preceded by grid velocity calculations using $\boldsymbol{v} = \tilde{\mathbf{m}}_k^{-1}\boldsymbol{p}$. Because the USL method can use the grid velocities calculated in the preceding grid-values update, it incurs no additional computational cost. In contrast, USF, USAVG and USL+ need one extra full mass matrix calculation for an additional $N(k-1)C_x$ cost. USAVG+ needs two full mass matrix calculations for an additional $2N(k-1)C_x$ cost. In brief, when using FMPM, USL is the fastest method. An example below further shows that USL is as accurate or more accurate than other methods.

Computational Costs

Summing the third column in Table 1, the total, extra computational cost for FMPM(k) by the USL method is $N(k-1)C_x$. As explained in the previous section, this cost increases to $2N(k-1)C_x$ for USF, USAVG and USL+ and to $3N(k-1)C_x$ for USAVG+. Because most tasks in conventional MPM loop over particles, its cost is approximately linear in the number of particles or NC_0 where C_0 is the cost per particle. The relative time for FMPM(k) compared to conventional MPM is therefore

$$\text{Relative Time} = 1 + A(k-1)F$$

where $F = C_x/C_0$ is the fraction of a time step spent within one loop of full mass matrix tasks. $A = 1$ for USL update method, but increases to 2 for USF, USAVG and USL+ or 3 for USAVG+. The extra time is proportional to $(k-1)$, but the actual slow down depends on F. For example, the k to double computational time is $k_2 = (1+F)/F$. Note that full mass matrix calculations only involve extrapolations between particles and the grid; C_x is therefore independent of complicating features in the simulation. In contrast, C_0 is smallest for trivial, model simulations, but increases when simulations add more physics such as large deformations, plasticity, damage mechanics, contact, cracks, coupled conduction, phase transitions, and more. When simulation complexity causes C_0 to increase, F will decrease. The examples in this paper had $0.33 < F < 0.49$.

2.4. Boundary Conditions

MPM simulations can add particle- or grid-based boundary conditions. First, the standard MPM derivation invokes divergence theorem resulting in a surface integral that physically represents traction boundary conditions [3]. When discretized into traction forces on the grid, $\boldsymbol{f}^{(T)}$, the traction force on node i is given by:

$$\boldsymbol{f}_i^{(T)} = \int_{S_T} N_i(\boldsymbol{x})\boldsymbol{T}(\boldsymbol{x})\, dS = \sum_{\partial p} \boldsymbol{T}_{p,s} \int_{A_s} N_i(\boldsymbol{x})\, dS$$

where S_T is surface of the deformed object subjected to traction load $\boldsymbol{T}(\boldsymbol{x})$ and $N_i(\boldsymbol{x})$ is *grid* shape function for node i. The second form implements this calculation by applying traction loads to surfaces of individual particles on the object's surface. In this form, $\partial p = \{p, s : \text{particle } p \text{ has constant traction } \boldsymbol{T}_{p,s} \text{ on surface } s\}$ and A_s is the deformed area for surface s of particle p. This integral can be evaluated several ways including CPDI style to account for surface deformation [9, 19]. Because traction forces are implemented by adding $\boldsymbol{f}^{(T)}$ to $\boldsymbol{f}$ during the grid forces task, no changes are needed in FMPM.

A second boundary condition style is to impose velocities on grid nodes. Zero-velocity nodes model a barrier while non-zero velocity nodes can pull or push objects. Non-zero velocity conditions need to translate through the grid when modeling displacements larger than a grid cell. Velocity boundary conditions are frequently the source of problems in MPM simulations. FMPM does not resolve these problems and full mass matrix approximations near edges may even exacerbate some effects. Nevertheless, we were able to implement grid boundary conditions in FMPM. The remainder of this section describes grid-based velocity condition options.

The task for velocity boundary conditions can be stated as: calculate an updated $\boldsymbol{p}^+$ without regard to grid-based boundary conditions, then correct grid velocity $\boldsymbol{v}^+$ and find acceleration $\boldsymbol{a}$ to such that final velocity updates match the conditions. When using a lumped mass matrix, the task is straightforward. Imagine one condition b setting the velocity on node i to $v_i^{(b)}$ in direction $\hat{\boldsymbol{n}}^{(b)}$. First, the uncorrected, lumped-mass velocity is found from $\boldsymbol{v}^{+UL} = \mathbf{m}^{-1}\boldsymbol{p}^+$ and then final velocity and acceleration are adjusted such that:

$$\boldsymbol{v}_i^{+L} \cdot \hat{\boldsymbol{n}}^{(b)} = v_i^{(b)} = \left(\boldsymbol{v}_i^{+UL} + \boldsymbol{a}_i^{(b)}\Delta t\right) \cdot \hat{\boldsymbol{n}}^{(b)} \quad \text{or} \quad \boldsymbol{a}_i^{(b)}\Delta t = \left(v_i^{(b)} - \boldsymbol{v}_i^{+UL} \cdot \hat{\boldsymbol{n}}^{(b)}\right)\hat{\boldsymbol{n}}^{(b)}$$

Multiple boundary conditions have one such equation for each controlled node and direction and those equations are uncoupled. The calculations for multiple conditions change each $\boldsymbol{v}_i^{+UL}$ to have $v_i^{(b)}$ in the boundary condition direction and set the final nodal accelerations to:

$$\boldsymbol{a} = \mathbf{m}^{-1}\boldsymbol{f} + \sum_b \boldsymbol{a}^{(b)}$$

Note that $\boldsymbol{f}^{(BC)} = \sum_b \mathbf{m}\boldsymbol{a}^{(b)}$ are reaction forces at boundary condition nodes.

In FMPM, a controlled velocity on one node should cause reaction forces on nearby nodes according to off-diagonal elements of the full mass matrix. But, each boundary condition still provides only one equation, which is not enough information to determine reaction force distribution among all nodes. Instead, we approached FMPM velocity conditions by analogy to the lumped mass-matrix methods. First, the uncorrected, full-mass matrix velocity is found from $\boldsymbol{v}^{+U} = \tilde{\mathbf{m}}_k^{-1}\boldsymbol{p}^+$ that expands to

$$\boldsymbol{v}^{+U} = \left(\sum_{\ell=1}^{k} (-1)^{\ell+1} \binom{k}{\ell} \left(\mathbf{S}^+\mathbf{S}\right)^{\ell-1}\right) \boldsymbol{v}^{+UL} \tag{3}$$

This initial step suggests three options:

Velocity Method: Change $\boldsymbol{v}_i^{+U}$ to have $v_i^{(b)}$ in the boundary condition direction:

$$\boldsymbol{v}^{+} = \boldsymbol{v}^{+U} + \sum_b \Delta \boldsymbol{v}^{(b)} \quad \text{where} \quad \Delta \boldsymbol{v}_j^{(b)} = \left(v_i^{(b)} - \boldsymbol{v}_i^{+U} \cdot \hat{\boldsymbol{n}}^{(b)} \right) \hat{\boldsymbol{n}}^{(b)} \delta_{ij}$$

Note that this calculation does not find final nodal accelerations. Unlike FLIP, however, the FMPM update does not use nodal accelerations and therefore they need not be calculated. A potential drawback of this approach is that the calculation affects only nodes with boundary conditions. An expectation that boundary conditions should influence nearby nodes as well is not met.

Lumped Method: Change $\boldsymbol{v}^{+UL}$ to $\boldsymbol{v}^{+L}$ in Eq. (3) (*i.e.*, velocity corrected using lumped mass matrix boundary condition methods) and then set $\boldsymbol{v}^{+} = \boldsymbol{v}^{+U}$. By this approach, the boundary conditions will affect accelerations and velocities on nodes with boundary conditions as well as some nearby nodes. But the nearby nodes might include nodes with other boundary conditions. As a result, the final grid velocities will only approximately satisfy boundary conditions.

Combined Method: This approach combines the above two methods. First, use $\boldsymbol{v}^{+L}$ in Eq. (3) and then change the resulting $\boldsymbol{v}_i^{+U}$ to have $v_i^{(b)}$ in the boundary condition direction. This approach spreads the effect of boundary conditions to nearby nodes and exactly satisfies the boundary conditions.

Section 3 considers each option, but, as described in the next section, best results with grid-based boundary conditions need to apply them to several nodes near a boundary.

2.5. Blurred Wall Velocity Conditions with Velocity Gradients

Problems with grid velocity boundary conditions are inherent to MPM's two discretizations — particles and a grid. Even when particle domains *appear* to have sharp edges, when those particles are extrapolated to the grid, such edges on the grid are blurred. Figure 1A shows a 1D grid with a particle domain on the left that ends at the central node. The dashed line plots mass in particle basis while the solid lines plot grid mass interpolated from lumped mass matrix. The grid mass is blurred by ± 1 cell or ± 1.5 cells for linear or quadratic spline shape functions. Expecting a grid boundary condition on the central node to effectively impose edge velocity on the particle domain is unreasonable. For non-zero velocity, expecting that periodically jumping the controlled node from one node to the next as displacement exceeds cell length would smoothly control edge velocity is also unreasonable. Each such jump acts like an impact that induces vibrations and degrades the simulation. This section describes "wall" boundary conditions for gripping and moving blurred edges. This "wall" option is beneficial to both prior MPM and FMPM.

The new boundary condition style is a virtual moving wall (see Fig. 1B). The zone left of the wall is filled with particles (such that the wall is at the object's edge) while the zone to the right is empty. Instead of applying velocity to one node near the particle-domain edge, this new style applies a velocity boundary condition to any node inside the particle domain that is less than or equal to a distance d from the wall and also applies a condition to any node outside the wall with non-zero extrapolated mass. The parameter d can be tuned for optimal results. To properly set velocity on multiple nodes, the wall is assigned to a velocity, $v^{(b)}$, and a gradient, $\nabla v^{(b)}$, such that the velocity on each controlled node is set to

$$v_i = v^{(b)} + \nabla v^{(b)} (x_i - x_{wall})$$

where x_i is position of node i and x_{wall} is current wall position. If the gradient is not known, it can be set to zero, but if a zero gradient extends too far into a domain, that approach would cause artifacts near the edge. A useful option to ameliorate edge artifacts is to estimate a gradient consistent with the simulation by using velocities extrapolated from the particles to nodes 1 and 2 in Fig. 1B (*i.e.*, the last uncontrolled node and the first controlled node within the particle domain). A least-square fit of these extrapolated velocities through $v^{(b)}$ at the wall leads to

$$\nabla v^{(b)} = \frac{\Delta v_1 \Delta x_1 + \Delta v_2 \Delta x_2}{(\Delta x_1)^2 + (\Delta x_2)^2} \quad \text{where} \quad \Delta v_i = v_i - v^{(b)} \quad \text{and} \quad \Delta x_i = x_i - x_{wall} \tag{4}$$

where v_i is velocity extrapolated from the particles to node i.

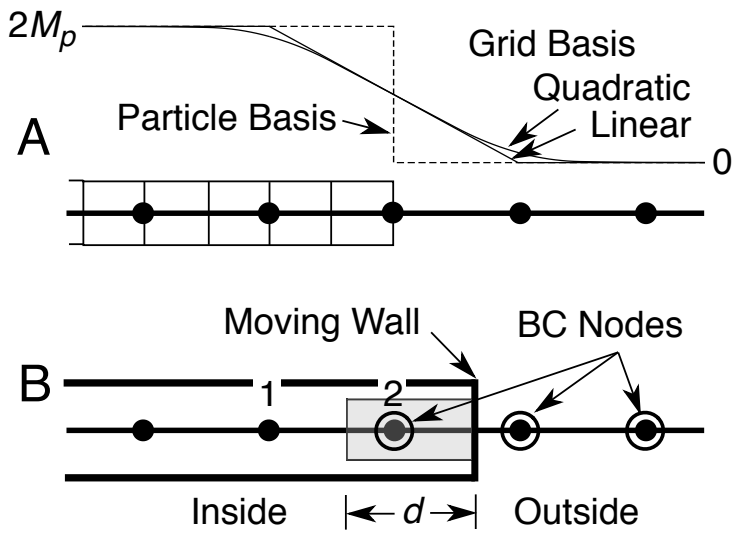

Figure 1: A. The edge of a particle domain on a 1D grid. The dashed line shows particle-basis mass while the solid lines interpolate lumped mass on the nodes using linear or quadratic spline shape functions. B. A moving wall on the right edge of a particle domain should set velocity boundary conditions on all circled nodes. d is the wall depth that determines which nodes inside the particle domain get velocity boundary conditions. Nodes 1 and 2 are used to estimate velocity gradient on the wall from extrapolated particle values.

The above description is 1D. This approach applies directly to 2D or 3D simulations where a row (in 2D) or plane (in 3D) of nodes can model a wall moving orthogonal to the x, y, or z axis. In principle, this approach could model arbitrarily-shaped walls moving in any direction, but we have not tested (or implemented) that option. A moving boundary approach is similar to methods implemented by Yang [20] that imagines a force field near a "wall" and adds forces to nodes proportional to some function of distance from the node to the wall. The Yang [20] approach was designed for contact calculations while the moving wall here is for velocity boundary conditions. We tried scaling the boundary condition reaction force by functions of distance but those attempts led to premature failure at the boundary conditions. Instead, full boundary condition changes were applied to all nodes identified as needed for modeling a moving wall. Section 3 has an example to guide selection of d for optimal results.

2.6. Additional Simulation Options (Briefly)

MPM simulations handle contact problems by using a multimaterial mode where each material extrapolates to its own velocity field [21–27]. Nodes that see only a single material are handled by standard MPM methods, but nodes with more than one material have to implement contact mechanics. All current MPM contact methods rely on lumped-mass matrix velocities and it is unlikely those methods can be adapted to full mass matrix calculations.

Preliminary results show that FMPM works reasonably with current contact methods provided it is corrected by methods recently derived to eliminate artifacts seen in XPIC simulations of shock impact [27]. In brief, multimaterial mode extrapolates momenta from particles of material type α to the grid using a modified shape function matrix:

$$\boldsymbol{p}^{\alpha 0} = \mathbf{S}^{\alpha T}\mathbf{M}\boldsymbol{V} \quad \text{where} \quad S^{\alpha}_{pi} = \begin{cases} S_{pi} & p \in \alpha \\ 0 & p \notin \alpha \end{cases}$$

Here $\boldsymbol{p}^{\alpha 0}$ is momentum that ignores contact interactions. The first task in MPM contact is to change this momentum to a contact-adjusted momentum of $\boldsymbol{p}^{\alpha} = \boldsymbol{p}^{\alpha 0} + \Delta\boldsymbol{p}^{\alpha}$ where $\Delta\boldsymbol{p}^{\alpha}$ is change in momentum calculated by detecting contact and implementing contact laws. A revised XPIC(k) particle update to minimize artifacts in shock impact used $\Delta\boldsymbol{p}^{\alpha}$ to redefine effective acceleration as:

$$\mathrm{A}^{(n)}\Delta t = S^{\alpha}\left((\tilde{\mathbf{m}}^{\alpha}_{k})^{-1}\boldsymbol{p}^{\alpha} - (\tilde{\mathbf{m}}^{\alpha}_{k-1})^{-1}\Delta\boldsymbol{p}^{\alpha}\right) + \mathbf{S}^{\alpha}(\mathbf{m}^{\alpha})^{-1}\boldsymbol{f}^{\alpha}\Delta t - \boldsymbol{V}^{(n)} \tag{5}$$

where $(\tilde{\mathbf{m}}_k^\alpha)^{-1}$ is the k-term expansion to inverse of a full mass matrix for material α only defined by $\tilde{\mathbf{m}}^\alpha = \mathbf{S}^{\alpha T}\mathbf{M}\mathbf{S}^\alpha$, $\mathbf{m}^\alpha$ is the lumped mass matrix for material α, and $\boldsymbol{f}^\alpha$ is nodal force for material α including contact forces. Note that Eq. (5) recast the effective acceleration from Ref. [27] using the full mass matrix notation defined here. By analogy to standard updates, we propose this update can be converted to FMPM by replacing $\mathbf{m}^\alpha$ with $\tilde{\mathbf{m}}_k^\alpha$ to get:

$$\mathbb{A}^{(n)}\Delta t = S^\alpha\left((\tilde{\mathbf{m}}_k^\alpha)^{-1}\boldsymbol{p}^{\alpha+} - (\tilde{\mathbf{m}}_{k-1}^\alpha)^{-1}\Delta\boldsymbol{p}^\alpha\right) - \boldsymbol{V}^{(n)}$$

While this approach gives good results in initial trials, additional corrections might further improve FMPM (and XPIC) in very-high velocity contact simulations. This subject may be addressed in a future publication.

MPM can model explicit cracks by using the CRAMP algorithm [17, 28–30]. Like multimaterial methods, the CRAMP method extrapolates multiple velocity fields to the grid where the different fields now correspond to particles on different sides of the crack. Most of CRAMP works in FMPM without modification. The only exception is modeling crack contact and that should be corrected by the XPIC revisions described above for contact in multimaterial MPM.

A recent development in MPM is to track velocity gradient on the particles, use that gradient when extrapolating to the grid, and update it during particle updates. The method has been termed APIC for an "Affine" method that updates particles using PIC methods [31, 32]. The APIC approach can be extended to full mass matrix methods by defining an "Affine" full mass matrix, $\tilde{\mathbf{m}}_A$, and extrapolating to the grid using:

$$\boldsymbol{v} = [\tilde{\mathbf{S}}]^+ \cdot \hat{\boldsymbol{V}} \quad \text{where} \quad [\tilde{\mathbf{S}}]^+ = (\tilde{\mathbf{m}}_A^{-1}\mathbf{S}^T\mathbf{M}, \tilde{\mathbf{m}}_A^{-1}\mathbf{S}^{\Lambda T}\mathbf{M}\mathbf{D}^{-1}) \quad \text{and} \quad \hat{\boldsymbol{V}} = (\boldsymbol{V}, \mathbf{B}^T)$$

Here $(\mathbf{S}^\Lambda)_{ip}^T = S_{pi}(\boldsymbol{x}_i - \boldsymbol{X}_p)^T$, $\mathbf{B}$ is vector of $\mathbf{B}_p$ particle tensors introduced in APIC to track velocity gradient, and $\mathbf{D}^{-1}$ is diagonal matrix with $\mathbf{D}_p^{-1}$ on the p^{th} diagonal that is used to find Eulerian velocity gradient using $\nabla\boldsymbol{V}_p = \mathbf{B}_p\mathbf{D}_p^{-1}$ [31, 32]. Using this extrapolation method and a corresponding extrapolation from grid to the particles, a revised full mass matrix becomes:

$$\tilde{\mathbf{m}}_A = \mathbf{m}[\mathbf{S}]^+ \cdot [\mathbf{S}] \quad \text{with} \quad [\mathbf{S}] = (\mathbf{S}, \mathbf{S}^\Lambda)$$

and $[\mathbf{S}]^+$ replaces full mass matrix in $[\tilde{\mathbf{S}}]^+$ with lumped mass matrix. Prior APIC methods lumped $\tilde{\mathbf{m}}_A$ (which reduces to $\mathbf{m}$ in standard MPM) and always updated the velocity gradient term by PIC methods (hence the APIC name). These choices mean that APIC has significant dissipation (albeit much less dissipation than non-affine PIC methods [33]). The path to affine MPM methods with less dissipation is to use the revised mass matrix in new full mass matrix calculations. This approach will be in a future publication.

3. Results and Discussion

Although FMPM is presented as a revision of MPM, when compared to MPM with XPIC(k), the changes are minor, but significant. Compared to XPIC MPM, the only two changes are 1) FMPM finds grid velocities using approximate full mass matrix inverse ($\boldsymbol{v}^+ = \tilde{\mathbf{m}}_k^{-1}\boldsymbol{p}^+$) prior to updating particle stresses and strains and 2) FMPM redefines $\mathbb{V}^{(n)}$ and $\mathbb{A}^{(n)}$ for particle updates. The revised grid-based boundary condition methods are generic MPM features that appear important for FMPM simulations, but can benefit FLIP and XPIC simulations as well.

A concern about any MPM scheme that deviates from FLIP MPM is dissipation. This section starts with a simple example that characterizes dissipation of FMPM compared to FLIP and XPIC and demonstrates that FMPM of sufficient order k and proper time step has excellent performance with very low dissipation. This example also suggests that the USL method is both the most efficient and the most justifiable method. Three additional examples consider traction boundary conditions, evaluate moving wall velocity condition methods, and demonstrate improved results in shock impact.

3.1. Energy Dissipation and Convergence

An effective problem for evaluating FMPM convergence and dissipation is to collide two 50×20 mm blocks in 2D, plane-strain simulations using CPDI shape functions, the USL update method, and four particles per

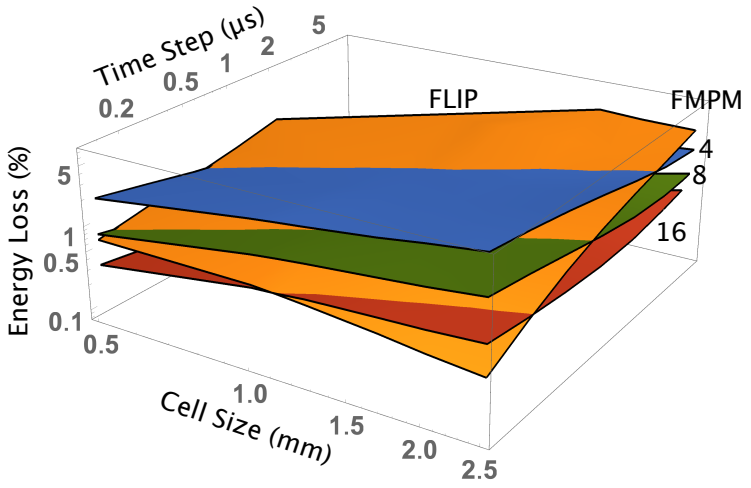

Figure 2: Energy loss (in percent) after elastic impact between two blocks as a function of time step and cell size. The planes show results for FLIP, FMPM(4), FMPM(8), and FMPM(16).

cell. The two blocks were modeled using a neo-Hookean, hyperelastic material with strain energy and Cauchy stress given by [2, 34]:

$$W = \frac{\lambda}{2}\left(\frac{J^2-1}{2} - \ln J\right) + \frac{G}{2}(I_1 - 3 - 2\ln J) \quad \text{and} \quad \boldsymbol{\sigma} = \frac{\lambda}{2}\left(J - \frac{1}{J}\right)\mathbf{I} + \frac{G}{J}\left(\mathbf{F}\mathbf{F}^T - \mathbf{I}\right) \tag{6}$$

where λ and G are Lamé and shear moduli, J is determinant of deformation gradient $\mathbf{F}$, and I_1 is first invariant of $\mathbf{F}\mathbf{F}^T$. The simulations used $\lambda = 7.29766$ MPa and $G = 3.7594$ MPa (which correspond to small-strain tensile modulus $E = 10$ MPa and Poisson's ratio $\nu = 0.33$). The blocks started with velocities $V_x^{(init)} = \pm 8.165$ m/sec (10% of the material's wave speed) and were moving toward each other along the x axis. In single-material MPM, the objects start interacting when particles from each block extrapolate to common nodes between the blocks, which for CPDI shape functions occurs when edges of the blocks are separated by less than two grid cells. The blocks were initially separated by 4 grid cells. This simulation is a good test for energy conservation because the abrupt impact represents an extreme case with energy being contained in very high frequencies. Any method that over filters high-frequency velocities will suffer from unacceptable dissipation.

This elastic impact should conserve energy. We ran simulations until soon after the blocks were fully separated and calculated energy dissipation by subtracting final total energy (sum of kinetic and strain energy) from initial energy (all kinetic energy). Figure 2 shows spatial and temporal convergence by plotting percent energy lost as function of cell size and time step in a 3D plot. The plot only shows FMPM(k) results that achieve less dissipation than FLIP. FMPM(4) dissipates less energy than FLIP for large time steps, but more than FLIP for small time steps. FMPM(8) and FMPM(16) reduce dissipation further. FMPM(16) dissipates less energy than FLIP except for large cells with very small time steps.

Figure 3 clarifies convergence by cross plotting energy loss as a function of cell size at constant time step or as a function of time step at constant cell size. The time step is plotted as CFL or Courant-Friedrichs-Lewy factor that is ratio of the time step to the time required for a stress wave to cross a single cell [35]. The spatial convergence shows FMPM converges well with slope close to 1. As expected FMPM(1) is highly dissipative (loses 20% to 90% of the energy). This simple dynamic problem clearly identifies FMPM(1) as having too much dissipation. Moving to just FMPM(2) reduced dissipation about an order of magnitude. For problems with much less dynamic response, FMPM(2) alone may be enough for stable MPM with low dissipation. For more dynamic problems, however, higher order FMPM(k) is likely needed. By FMPM(4), dissipation is comparable to FLIP. For FMPM(8) and higher, the dissipation is always less than FLIP for this time step.

Figure 3B shows that FMPM does not converge with reduced time step — dissipation increases as the time step decreases. When time step is reduced at constant cell size, a given simulation uses more time steps.

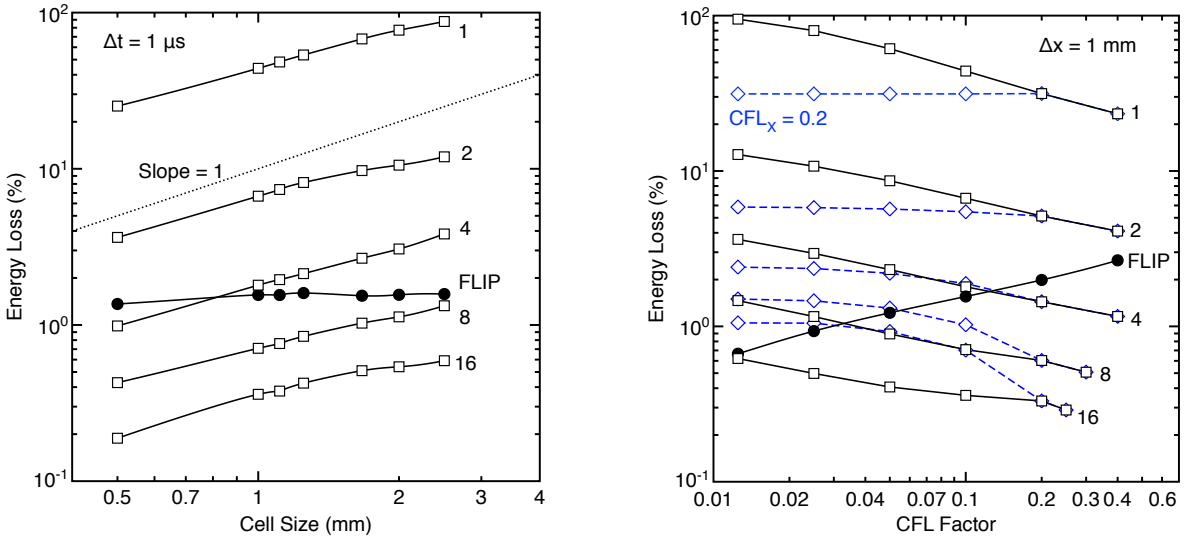

Figure 3: Energy loss (in percent) after elastic impact between two blocks for FLIP and FMPM(k) for k from 1 to 16. A. Spatial convergence as a function of cells size. B. Temporal convergence plotted as a function of CFL factor. The dashed curves used CFL factor for the simulation, but $\text{CFL}_\text{X} = 0.2$ for full mass matrix calculations.

Because each time step replaces particle velocities with extrapolated velocities, any errors in the mass matrix inverse may accumulate as the number of time steps increases. To keep dissipation lower than FLIP, the CFL factor for a given FMPM order should be above some minimum value. The higher the order FMPM(k), the more it can tolerate a low CFL factor. For the problem in Fig. 3, the CFL can be as low as 0.1, 0.02, or 0.01 for FMPM(k) of order 4, 8, or 16, respectively.

If too many FMPM extrapolations cause dissipation, perhaps doing them less often would help? To implement this option, we tried periodic FMPM. In brief, a simulation selects two CFL factors — one CFL for the simulation time step and a second CFL_X to choose the frequency of using FMPM calculations. Time steps not using FMPM use standard FLIP methods. Because we claim FMPM is doing the proper calculations by approximating the full mass matrix, it is inconsistent to mix it with "improper" FLIP time steps. But, the XPIC interpretation of FMPM is that it is filtering out null-space noise [5]. Viewed as a noise-reduction method, applying that filtering periodically could make sense. The dashed lines in Fig. 3B repeat the two-block simulations as a function of CFL with fixed $\text{CFL}_\text{X} = 0.2$ for FMPM calculations. The results are mixed. For low-order k, FMPM dissipated more energy than FLIP and thus the periodic method was able to reduce dissipation. But, when FMPM(k) dissipated less than FLIP, interspersing with FLIP caused more dissipation. The recommended approach is to use proper order of FMPM(k) rather than to combine it periodically with FLIP time steps.

For computational efficiency, FMPM should use the lowest order possible. To assess benefits of higher orders, Fig. 4 plots energy error as a function of FMPM order k for 1 mm cells and CFL=0.2. These results also considered MPM shape function effects. Both CPDI and uGIMP by FMPM for $k \geq 3$ dissipate less than when using them with FLIP. B2CPDI and B2GIMP shape functions reduced the dissipation in FLIP and now required $k \geq 4$ to dissipate less than FLIP. The "XPIC(CPDI)" curve in Fig. 4 shows the energy dissipation of prior XPIC noise reduction methods [5]. The XPIC dissipation decreases at higher order k, but is always more dissipative than both FLIP and FMPM. Note that order k corresponds to number of grid-to-particle extrapolations used in FMPM calculations. In theory, higher k provides more accurate mass-matrix inversion, but eventually improvements with increasing k are overwhelmed by round-off error associated with the extra extrapolations. In fact, very high k eventually devolves into unstable calculations. All calculations in Fig. 4 remained stable to $k = 16$ except for uGIMP, which was stable to $k = 12$ but unstable when $k = 16$. We did not try any higher k values.

Finally, from the slope of relative calculation time as a function of FMPM order, the cost scaling for FMPM in these simulations was $F = 0.49$. The effect of number of processors on $F = C_x/C_0$ depends on a code's parallel scaling for C_x and C_0. Our calculations found F to be independent of the number of

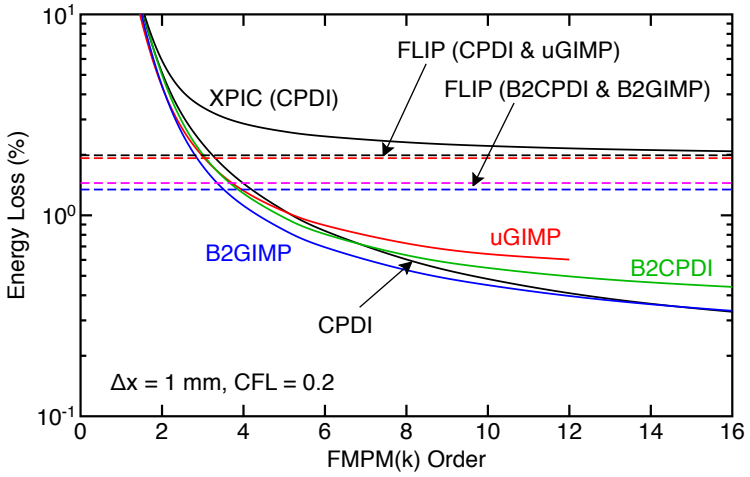

Figure 4: Energy loss after elastic impact between two blocks as a function of FMPM order for various shape functions (CPDI, uGIMP, B2CPDI, and B2GIMP). The horizontal lines show the energy loss for FLIP simulations with the various shape functions. The "XPIC (CPDI)" curve is for XPIC as a function order using CPDI shape functions.

processors. In other words, parallel scaling for full mass matrix calculations is about the same as scaling for conventional MPM. From energy loss results in Fig. 4 for the two-block problem, the benefits of using higher than FMPM(4) or FMPM(5) probably do not justify the extra cost. With $F = 0.49$, such simulations would be about 2.5 to 3 times slower. Note that block impact is a trivial, model problem. It modeled an elastic material, had no boundary conditions, and no additional MPM options. The cost scaling F will decrease as simulation complexity increases (see shock example below). The minimum order needed for other problems will also differ. The minimum order should be lower for problems close to quasi static and higher for problems in shock physics.

3.2. Particle Stress and Strain Update Method

Figure 5 plots spatial convergence for a constant time step by FLIP or FMPM(8) using all update methods defined above (USL, USF, USAVG, USL+, and USAVG+). A common justification for using FLIP is that it is a low dissipation method. Indeed, the energy loss by USAVG+ is close to zero for any grid size. But how can a method get exact energy conservation using a grid that cannot represent velocity variations in the exact solution? Such a method may retain total energy, but it must be partitioning that energy incorrectly into velocities frequencies it can represent. This non-physical transfer is caused by FLIP's use of a lumped mass matrix. Another reason to question FLIP's energy conservation is its inconsistent spatial convergence properties. None of the update methods appear to converge. USF gains energy at high resolution (*i.e.*, negative energy loss). USL and USL+ dissipate energy and are not converging well. USAVG and USAVG+ are between USF and USL methods and even USAVG+ starts to gain energy at high resolution.

In contrast, the FMPM(8) convergence is consistent. The differences between the update methods are small and all methods are converging toward zero energy loss. Because all update methods are similar, the most efficient method, namely USL, is the preferred method. Another advantage of USL is that the particle velocity, position, stress, and strain are all updated using the same grid velocity. In contrast, USF and USL+ update velocity and position with grid velocity found after the momentum update but update stresses and strains using a different grid velocity. The USF method has the worst convergence and is heading toward energy gain at the highest resolution. Similarly, USAVG and USAVG+ are likely worse than USL because they include 50% USF. The solid, thick line in Fig. 5 is cubic interpolation of the USL results; it extrapolates close to zero energy loss at zero cell size.

3.3. Traction Boundary Conditions

To verify traction boundary conditions in FMPM and compare to the same conditions in standard MPM, we impacted a 198×10 mm bar on the left end with a triangular tensile traction pulse. The boundary

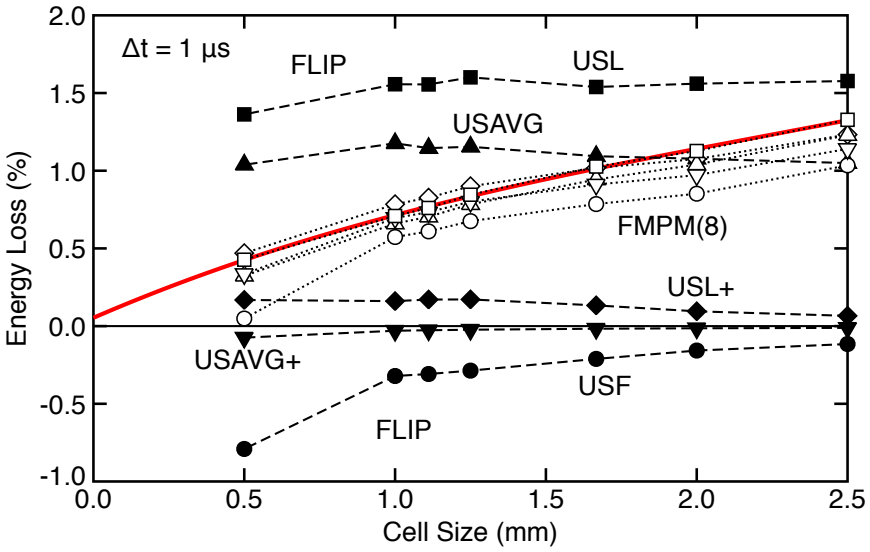

Figure 5: Spatial convergence for two-block impact using a time step $\Delta t = 1$ μs for USL (squares), USF (circles), USL+ (diamonds), USAVG (triangles), and USAVG+ (inverted triangle) particle stress and strain update methods. The filled simples with dashed lines used FLIP particle updates. The open symbols with dotted lines used FMPM(8) particle updates. The solid, thick line is a cubic interpolation of the USL methods by FMPM(8).

condition pulse and the stress within the bar prior to stress wave reaching right edge of the specimen were:

$$\boldsymbol{T}(x=0) = \sigma_{max}\mathrm{tri}(t,\Delta t) \qquad \text{and} \qquad \sigma_{\mathrm{xx}}(\mathrm{x},\mathrm{t}) = \sigma_{\mathrm{max}}\mathrm{tri}\,(\mathrm{t}-\mathrm{cx},\Delta\mathrm{t})$$

where $\mathrm{tri}(t,\Delta t)$ is a triangular function of full width Δt, peak = 1 at $t = \Delta t/2$, and zero outside the range $0 < t < \Delta t$, and c is the material's wave speed. These simulations used a small-strain, isotropic, linear-elastic material with $E = 10$ MPa, $\nu = 0$, and $\rho = 1000$ kg/m^3 giving $c = 100$ m/s. The traction pulse used $\sigma_{max} = 0.0005$ MPa (to insure small strain) and $\Delta t = 1$ ms. The MPM features were 2D, plane stress analysis, B2SPLINE shape functions, four material points per grid cell, CFL factor 0.2, and USL particle update method. The simulations were run for 1.5 ms such that final spatial variation in σ_{xx} should be a triangular function centered at $x = 100$ mm with half width of 50 mm. To assess convergence, the cell size was varied and we calculated RMS error in particle stresses using:

$$\text{RMS Stress Error} = \sqrt{\frac{1}{N}\sum_{p}\left(\sigma_{p,xx} - 0.0005\,\mathrm{tri}\left(\frac{150-x}{100},1\right)\right)^2}$$

The convergence results for traction boundary conditions are in Fig. 6A. Lumped mass-matrix FLIP shows approximately first-order convergence. Note that FLIP appears to have spatial convergence here (when it did not in Fig. 5) because CFL was held constant meaning the time step also decreased as cell size decreased. As expected, FMPM(1) has larger error because that method filters high frequency spatial frequencies needed to resolve a triangular pulse. But all FMPM with $k \geq 2$ have lower error than standard MPM and also have first-order convergence. Fig. 6B plots particle stresses for cell size 1 mm and several calculation methods compared to theoretical triangular function centered at $x = 100$ mm. FMPM(1) results have a "smoothed" pulse caused by too much velocity filtering. The inset near the peak shows that the FLIP has both amplitude and phase errors (the peak is rounded and offset about 2 mm from the expected peak at 100 mm). All FMPM($k \geq 2$) have more accurate peak location and reduced rounding as k increases. FMPM($k > 4$) provided only negligible improvements in pulse shape compared to FMPM(4). The inset near the edge of the pulse at $x = 50$ mm shows that FMPM($k \geq 2$) reduces the ringing seen in FLIP.

3.4. Velocity Boundary Conditions

To test velocity boundary condition options, we used the method of manufactured solutions (MMS) for a uniaxial strain problem with known, exact solution [36]. Imagine a rectangular 20×4 m bar with each particle

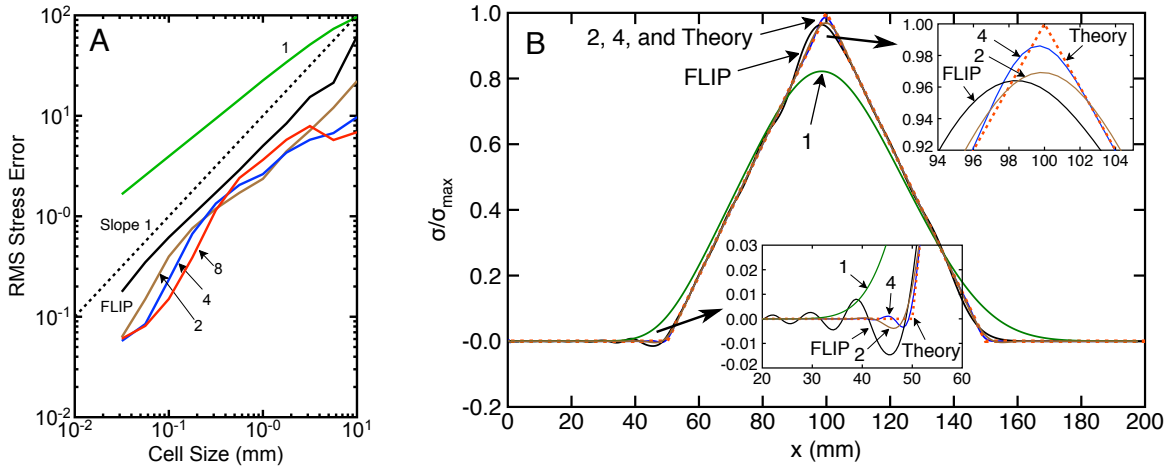

Figure 6: A. RMS error convergence for stresses in a thin bar impacted with a triangular traction pulse for FLIP and FMPM of various orders. B. Stress *vs.* position at the end of the simulation for FLIP, FMPM(1), FMPM(2), and FMPM(4) compared to "theory" or a dotted-line triangular pulse. The insets magnify the plot near the peak and near the leading edge.

subjected to deformation gradient with $F_{xx} = 1 + \dot{\varepsilon}t$, $F_{yy} = F_{zz} = 1$ and all other components zero, where $\dot{\varepsilon}$ is a constant strain rate. If $x = 0$ at the bar center, each particle starts with velocity $\boldsymbol{V}_p = (\dot{\varepsilon}X_p, 0, 0)$, and the material is Neohookean (see Eq. (6)), the exact solution is that particle velocities remain constant (*i.e.*, the problem has zero acceleration) and all particles have the same stress:

$$\sigma_{xx} = (\lambda + 2G)\frac{(2+\dot{\varepsilon}t)\,\dot{\varepsilon}t}{2\,(1+\dot{\varepsilon}t)} \quad \text{and} \quad \sigma_{zz} = \sigma_{yy} = \lambda\frac{(2+\dot{\varepsilon}t)\,\dot{\varepsilon}t}{2\,(1+\dot{\varepsilon}t)}$$

We attempted to reproduce this exact solution in MPM using 0.25 m cells, USL update, two material points per cell in each direction, and $\dot{\varepsilon} = 0.0025$ sec^{-1}. The simulation was controlled solely by the new wall velocity boundary conditions that set velocity and gradient on the x axis ends of the bar to have Eulerian velocity and gradient:

$$v(x,t) = \frac{\dot{\varepsilon}x}{1+\dot{\varepsilon}t} \quad \text{and} \quad \frac{\partial v(x,t)}{\partial x} = \frac{\dot{\varepsilon}}{1+\dot{\varepsilon}t}$$

while the sides should have fixed walls with zero velocities and gradients in the y and z directions. To focus on moving boundary conditions, we set G=0.375 Pa and $\lambda = 0$ (or E=0.75 Pa and $\nu = 0$) such that $\sigma_{yy} = \sigma_{zz} = 0$. We could therefore eliminate side boundary conditions and use only moving-wall boundary conditions on the two ends. The end velocities of ± 0.025 m/sec were 5.8% of the material's wave speed (with density $\rho = 1$ kg/m^3). The absence of acceleration, means that high loading rate does not induce inertial effects. We ran 3D, 2D plain strain (plane stress is identical when $\lambda = 0$), and 1D (in separate code) simulations and all gave similar results. The results plotted here are from 2D calculations.

Simulation results for prior MPM (labeled FLIP) and FMPM(k) for $k = 1$, 2, 4, and 8 using CPDI shape functions are in Fig. 7. All simulations elongated the specimen by 20% such that the moving walls crossed eight grid cells during the simulation. The plotted velocity error (in percent) was calculated as average of the RMS velocity error over the full simulation normalized to the end velocity:

$$\langle \text{Velocity Error} \rangle \;(\%) = \frac{100}{TV_{end}}\int_0^T \sqrt{\frac{(V_{p,x}(t) - \dot{\varepsilon}X_p^{(0)})^2 + V_{p,y}(t)^2 + V_{p,z}(t)^2}{N}}\,dt$$

where $\boldsymbol{V}_p(t) = (V_{p,x}(t), V_{p,y}(t), V_{p,z}(t))$ is particle velocity as a function of time, $X_p^{(0)}$ is the particle's initial x position, and T is total simulation time. Simulations using FMPM(2), FMPM(4), and FMPM(8) used the three different boundary condition methods discussed above — velocity method (dotted lines), lumped method (dashed lines), and combined method (solid lines). FLIP and FMPM(1) (thicker lines) used only the lumped method because adding velocity corrections would not change the results. Observations about velocity boundary conditions are:

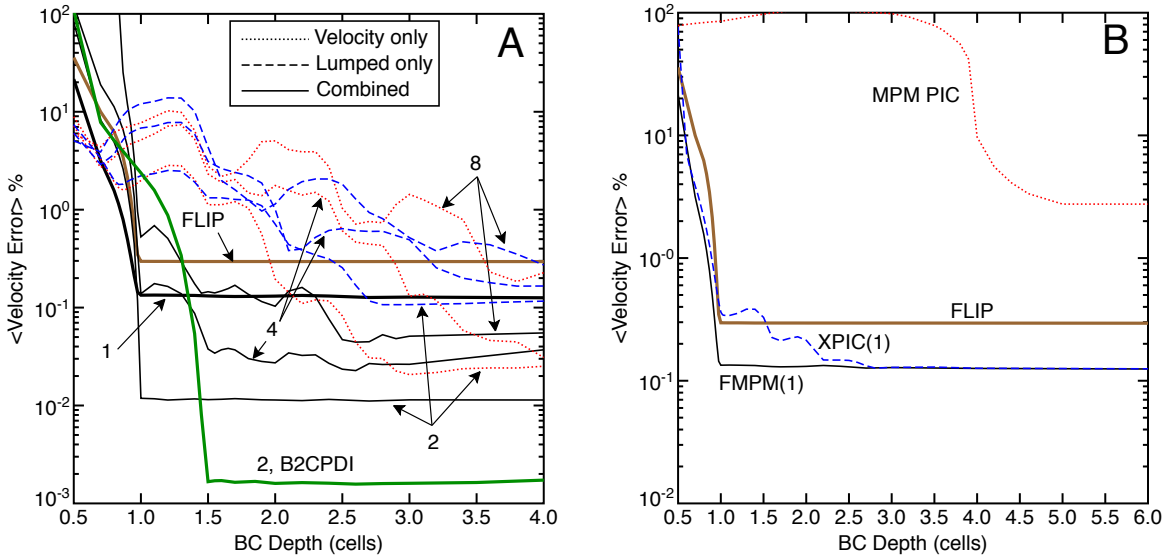

Figure 7: A. RMS velocity errors averaged over 20% elongation as a function of wall boundary condition depth for FLIP and FMPM(1,2,4,8). For FMPM(2,4,8), dotted lines used "Velocity Method", dashed lines used "Lumped Method," and solid lines combined those methods. The curve with lowest error used FMPM(2) and B2CPDI shape functions. B. RMS velocity errors averaged over 20% elongation as a function of wall boundary condition depth for three PIC methods — FMPM(1), XPIC(1), and MPM PIC. FLIP results are included for reference

1. The errors are high for all methods for wall depth $d < 1$ grid cell (errors $\geq 100\%$ mean the bar broke free from the boundary conditions). To robustly pull an object, velocity boundary conditions in MPM should set the velocity on at least one node *inside* the particle domain. The continuous dependence of results on d are a consequence of a moving wall that causes node-to-wall distance to change during the simulation. For simulations with a fixed wall on a node (*e.g.*, set material edge to zero velocity), the simulation errors would be a step function of d. Fixed-wall conditions similarly have higher errors for $d < 1$ and should always use $d \geq 1$ or fix the node on edge of particle domain *and* the first node inside the domain (in addition to any active nodes outside the domain).
2. The errors for FMPM(k) are eventually less than FLIP errors, but the higher the order, the higher the d needed to get low error. This behavior is likely because the full mass matrix spreads the conditions to more nodes implying that more nodes need to be controlled for accurate results. Neither the velocity nor the lumped method worked well. They required a high d before errors are comparable to FLIP. The combined method, however, provides significant improvements. Note that FMPM(2) errors are $30\times$ lower than FLIP errors.
3. The benefits of increasing depth diminish for $d \geq 1$ for FLIP, FMPM(1) and FMPM(2), for $d \geq 1.5$ for FMPM(4), and for $d \geq 2.5$ for FMPM(8) (using the combined method). The "2, B2CDI" curve repeats FMPM(2) simulations using quadratic spline shape functions on the grid and the errors are very low ($175\times$ lower than FLIP). Because quadratic splines blur edges further, they needed $d \geq 1.5$ to reach low errors. To avoid edge artifacts, d should be kept to a minimum and ideally to a small fraction of cells across the particle domain. The recommendations are to use $d = 1$ or $d = 1.5$ for linear or quadratic grid shape function, to increase d if needed for higher order FMPM(k), and to use the combined method.
4. FMPM(2) ran stably until CPDI shape functions numerically separated, which happens when x ends of particles starting with one edge at $x = 0$ are in disconnected cells in the grid ($> 300\%$ strain in this example). FMPM(2) with B2CPDI shape functions ran to 650% before numerical fracture.

Equation (2) defines three particle-in-cell methods. All three update particle velocity by replacing it with a value extrapolated from lumped-mass-matrix velocities on the grid. They differ in how they update particle position. Velocity errors using the three PIC methods are plotted in Fig. 7B. The "MPM PIC" method, which uses a first-order FLIP update for position, could not even grip the specimen for $d < 4$. It should

never be used. Orthogonal cutting simulations in Ref. [13] pointed out that PIC methods are equivalent to adding a damping term to acceleration. That damping must be included in both the velocity and position updates. In short, the "MPM PIC" method is using different accelerations for velocity and position updates. An update that uses consistent acceleration from Ref. [13] is plotted as XPIC(1) and has good results. The FMPM(1) proposed in this paper provides some additional improvement and therefore defines the optimal PIC approach to MPM.

3.5. Example Shock Wave Calculation

The last example modeled pressure and temperature shock waves induced by high-speed impact on one end of a bar. This problem is identical to the shock example in Ref. [27]. In brief, we modeled a 50×2.5 mm nickel bar confined by walls on top, bottom, and left and impacted on the right with impact speed of 1840 m/sec (40% of the material's bulk wave speed). The MPM conditions were 2D, plane strain analysis, 0.25×0.25 mm cells, four particles per cell, B2CPDI shape functions, USL update method, coupled heat conduction, and CFL = 0.2. The confined edges used the new wall boundary conditions with zero velocity, zero gradient, and $d = 1.5$. The shock impact was done by wall boundary conditions with $v_i^{(b)} = -1840$ m/sec, $d = 1.5$, and gradient determined from extrapolated velocities (see Eq. (4)). The nickel was modeled as a hyperelastic-plastic material based on Mie–Grüneisen equation of state [37] for pressure. The Mie–Grüneisen properties were $K = 188$ GPa (bulk modulus), $\gamma_0 = 2$, $S_1 = 1.44$, and $\rho_0 = 8.87$ g/cm^3 [37]. The plasticity properties had no effect in these confined shock conditions (and thus are not listed). To complete the shock physics, the modeling added a pressure q when in compression described as "artificial viscosity" [37, 38] and given by:

$$q = \rho \Delta x |D_{kk}| \big(A_1 C + A_2 \Delta x |D_{kk}| \big)$$

where ρ is current density, $|D_{kk}|$ is trace of the velocity gradient, C is current bulk wave speed, Δx is cell size in the MPM grid, and A_1 and A_2 are two parameters that control the amount of artificial viscosity.

Figure 8A shows simulation results for FLIP, XPIC(4), FMPM(1), FMPM(2), and FMPM(4) including artificial viscosity with parameters $A_1 = 0.4$ and $A_2 = 4.0$. The FLIP and XPIC(4) results are similar showing that XPIC does not over damp this shock example (compared to FLIP). But both FLIP and XPIC(4) have considerable ringing around the leading edge of the shock front, which is theoretically a square wave. FMPM(1) overdamps causing the plateau pressure to be too low and the leading front to lag the expected front (*i.e*, shock velocity is too low). But FMPM(2) and FMPM(4) give excellent results. The plateau pressure agrees with theoretical results for the modeled constitutive law [39] and the leading edge has the proper shock velocity. Even in this highly-dynamic problem, FMPM(2) appears acceptable and FMPM(4) is sufficiently high order. Simulations with FMPM(8) were not significantly different than FMPM(4) simulations. The oscillations on the right edge are boundary conditions artifacts at the impact site. They did not affect the shock wave results a few cells away from that edge.

Artificial viscosity is essential to control ringing at the leading edge of the pressure wave when modeling by FLIP or XPIC. Because FMPM greatly reduces that ringing, we tested whether FMPM can do shock simulations with little (or no) artificial viscosity. Figure 8B shows FLIP and FMPM(4) simulations with no artificial viscosity. The FLIP simulations have very large oscillations that compromise modeling results. The FMPM(4) results now show some ringing, but overall pressure wave was reasonably accurate. The plateau pressure, however, does not agree as well with theoretical predictions [39]. Besides pressure waves, we looked at temperature waves caused by compression-induced heating. Like pressure waves, temperature waves had ringing in FLIP and were much closer to a square wave in FMPM(4). Unlike pressure waves, however, all methods required artificial viscosity for temperature plateau to agree with predictions [39]. Although FMPM still needs artificial viscosity to model both pressure and temperature effects, the FMPM(4) results virtually eliminated ringing seen in FLIP and XPIC calculations. The ringing effects are likely a consequence of using lumped mass matrix to find particle acceleration.

Finally, this shock problem is a multi-physics simulation. It included non-linear constitutive law, plasticity, coupled conduction, and many wall boundary conditions. The computational cost factor for these simulations was calculated to be $F = 0.33$. Thus, FMPM(4) simulations were 2.0 times slower than FLIP calculations, but that cost resulted in much improved results.

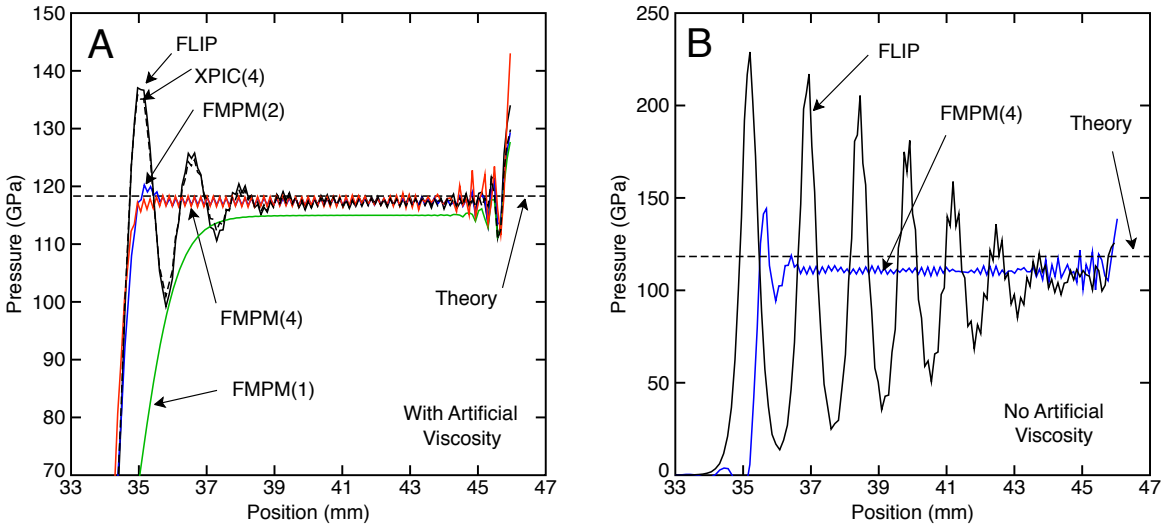

Figure 8: A. Pressure shock wave for simulations using FLIP, XPIC(4), FMPM(1), FMPM(2), and FMPM(4). The simulations used artificial viscosity. The horizontal, dashed line is the theoretical prediction for the impacted material. B. Pressure shock wave simulations using FLIP and FMPM(4) run without artificial viscosity.

4. Conclusions

A new series expansion to the full mass matrix inverse in MPM was derived. When compared to a prior projection operator used in XPIC(k) methods to remove null-space noise [5], the two derivations were found to be functionally equivalent. This new insight guided development of FMPM(k) that uses an approximate full mass matrix inverse to make relatively minor enhancements to prior XPIC(k) methods. The key changes were:

1. Redefine $\mathbb{A}^{(n)}$ to get acceleration from approximate full mass matrix inverse. In contrast, XPIC(k) found acceleration with a lumped mass matrix.
2. Redefine $\mathbb{V}^{(n)}$ to match particle velocity at the start of the time step. This approach eliminates velocity inconsistencies and differs from XPIC(k)'s use of a smoothed effective velocity.
3. When updating particle stresses and strains, find the particle velocity gradient by extrapolating grid velocities calculated with an approximate full mass matrix inverse. XPIC(k) based these updates on lumped mass-matrix velocities.
4. By updating particle stresses and strains after the particle update (USL method), the computational cost of FMPM(k) is identical to the cost of XPIC(k).

Despite the modest changes, FMPM improves significantly on XPIC. All simulations showed that FMPM dissipates less energy than XPIC of the same order and is more accurate. Compared to FLIP, all simulations showed that with sufficient order, FMPM dissipates less energy and provides more accurate results. In many simulations, FMPM(2) will be adequate for stable results without undesirable dissipation. In highly-dynamic problems, FMPM may require k from 3 to 5, but rarely will any problem need $k > 5$. We recommend avoiding very high orders ($k > 8$) because they would be computationally expensive and could have problems at grid-based, velocity boundary conditions. When grid-based, velocity boundary conditions, are used, all variants of MPM can benefit by new wall methods that account for velocity gradient and set boundary conditions on nodes within 1.0 cell (for linear shape functions) or 1.5 cells (for quadratic shape functions) inside the current wall position.

Acknowledgements

This work was made possible by the endowment for the Richardson Chair in Wood Science and Forest Products.

References

References

[1] O. C. Zienkiewicz, R. L. Taylor, J. Z. Zhu, The Finite Element Method: Its Basis & Fundamentals, Elsevier Butterworth-Heinemann, Oxford, UK, 2000.

[2] O. C. Zienkiewicz, R. L. Taylor, The Finite Element Methods for Solid and Structural Mechanics, Elsevier Butterworth-Heinemann, Oxford, UK, 2000.

[3] D. Sulsky, Z. Chen, H. L. Schreyer, A particle method for history-dependent materials, Comput. Methods Appl. Mech. Engrg. 118 (1994) 179–186.

[4] D. Sulsky, S.-J. Zhou, H. L. Schreyer, Application of a particle-in-cell method to solid mechanics, Comput. Phys. Commun. 87 (1995) 236–252.

[5] C. C. Hammerquist, J. A. Nairn, A new method for material point method particle updates that reduces noise and enhances stability, Computer Methods in Applied Mechanics and Engineering 318 (2017) 724–738.

[6] S. G. Bardenhagen, E. M. Kober, The generalized interpolation material point method, Computer Modeling in Engineering & Sciences 5 (2004) 477–496.

[7] M. Steffen, P. C. Wallstedt, J. E. Guilkey, R. Kirby, M. Berzins, Examination and analysis of implementation choices within the material point method (MPM), Computer Modeling in Engineering & Sciences 31 (2) (2008) 107–127.

[8] A. Sadeghirad, R. M. Brannon, J. Burghardt, A convected particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations, Int. J. Num. Meth. Engng. 86 (12) (2011) 1435–1456.

[9] J. A. Nairn, J. E. Guilkey, Axisymmetric form of the generalized interpolation material point method, Int. J. for Numerical Methods in Engineering 101 (2015) 127–147.

[10] N. E. Gibbs, J. W. G. Poole, P. K. Stockmeyer, An algotithm for reducing the bandwidth and profile of a sparse matrix, SIAM J. Numer. Anal 13 (1976) 236–250.

[11] J. U. Brackbill, H. M. Ruppel, FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions, Journal of Computational Physics 65 (2) (1986) 314 – 343.

[12] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, A. Selle, A material point method for snow simulation, ACM Trans. Graph. 32 (4) (2013) 102:1–102:10.

[13] J. A. Nairn, Numerical simulation of orthogonal cutting using the material point method, Engineering Fracture Mechanics 149 (2015) 262–275.

[14] J. A. Nairn, Numerical modeling of orthogonal cutting: Application to woodworking with a bench plane, Interface Focus 6 (3) (2016) 20150110.

[15] C. Moler, C. Van Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty five years later, SIAM Review 45 (1) (2003) 3–49.

[16] S. G. Bardenhagen, Energy conservation error in the material point method, J. Comp. Phys. 180 (2002) 383–403.

[17] J. A. Nairn, Material point method calculations with explicit cracks, Computer Modeling in Engineering and Sciences 4 (6) (2003) 649–664.

[18] S. Zhou, The numerical prediction of material failure based on the material point method, Ph.D. thesis, University of New Mexico (1998).

[19] A. Sadeghirad, R. Brannon, J. Guilkey, Second-order convected particle domain interpolation (CPDI2) with enrichment for weak discontinuities at material interfaces, International Journal for Numerical Methods in Engineering 95 (11) (2013) 928–952. doi:10.1002/nme.4526.
URL http://dx.doi.org/10.1002/nme.4526

[20] W.-C. Yang, Study of tsunami-induced fluid and debris load on bridges using the material point method, Ph.D. thesis, University of Washington (2016).

[21] S. G. Bardenhagen, J. U. Brackbill, D. Sulsky, The material point method for granular materials, Computer Methods in Applied Mechanics and Engineering 187 (2000) 529–541.

[22] S. G. Bardenhagen, J. E. Guilkey, K. M. Roessig, J. U. Brackbill, W. M. Witzel, J. C. Foster, An improved contact algorithm for the material point method and application to stress propagation in granular material, Computer Modeling in Engineering & Sciences 2 (2001) 509–522.

[23] X.-F. Pan, A.-G. Xu, G.-C. Zhang, P. Zhang, J.-S. Zhu, S. Ma, X. Zhang, Three-Dimensional Multi-mesh Material Point Method for Solving Collision Problems, Communications in Theoretical Physics 49 (2008) 1129–1138. arXiv:0708.3532, doi:10.1088/0253-6102/49/5/09.

[24] V. Lemiale, A. Hurmane, J. A. Nairn, Material point method simulation of equal channel angular pressing involving large plastic strain and contact through sharp corners, Computer Modeling in Eng. & Sci. 70 (1) (2010) 41–66.

[25] J. A. Nairn, Modeling of imperfect interfaces in the material point method using multimaterial methods, Computer Modeling in Engineering and Sciences 92 (3) (2013) 271–299.

[26] J. A. Nairn, S. G. Bardenhagen, G. S. Smith, Generalized contact and improved frictional heating in the material point method, Computational Particle Mechanics 5 (3) (2018) 285–296. doi:10.1007/s40571-017-0168-1.

[27] J. A. Nairn, C. C. Hammerquist, G. Smith, New material point method contact algorithms for improved accuracy, large-deformation problems, and proper null-space filtering, Computer Methods in Applied Mechanics and Engineering 362 362 (2020) 112859. doi:10.1016/j.cma.2020.112859.

[28] Y. Guo, J. A. Nairn, Three-dimensional dynamic fracture analysis in the material point method, Computer Modeling in Engineering & Sciences 16 (2006) 141–156.

[29] S. G. Bardenhagen, J. A. Nairn, H. Lu, Simulation of dynamic fracture with the material point method using a mixed j-integral and cohesive law approach, Int. J. Fracture 170 (2011) 49–66.

[30] Y. Guo, J. Nairn, Simulation of dynamic 3d crack propagation within the material point method, CMES - Computer Modeling in Engineering and Sciences 113 (4) (2017) 389–410.

[31] C. Jiang, C. Schroeder, A. Selle, J. Teran, A. Stomakhin, The affine particle-in-cell method, ACM Trans ACM Trans Graph 34 (4) (2015) 51:1–51:10.

[32] C. Jiang, C. Schroeder, J. Teran, An angular momentum conserving affine-particle-in-cell method, Journal of Computational Physics 338 (2017) 137–164.

[33] O. Ding, T. Shinar, C. Schroeder, Affine particle in cell method for MAC grids and fluid simulation, Journal Of Computational Physics 408 (2020) 109311.

[34] R. W. Ogden, Non-Linear Elastic Deformations, Ellis-Harwood, New York, 1984.

[35] R. Courant, K. Friedrichs, H. Lewy, On the partial difference equations of mathematical physics, IBM J. Res. Dev. 11 (2) (1967) 215–234. doi:10.1147/rd.112.0215.
URL http://dx.doi.org/10.1147/rd.112.0215

[36] K. Kamojjala, R. Brannon, A. Sadeghirad, J. Guilkey, Verification tests in solid mechanics, Engineering with Computers 31 (2) (2015) 193–213. doi:10.1007/s00366-013-0342-x.

[37] M. L. Wilkens, Computer Simulation of Dynamic Phenomena, Springer-Verlag, New York, 1999.

[38] J. Von Neumann, R. D. Richtmyer, A method for the numerical calculation of hydrodynamic shocks, J. Appl. Phys. 21 (1950) 232–237.

[39] C. A. Forest, Isoentrope energy, Hugoniot temperature, and the Mie-Gruneisen equation of state, AIP Conference Processinds 370 (1) (1996) 31–34.