# User manual of the parallel 3D implicit MPM program – pMPM

Lei Wang
sanshi.wang@gmail.com

12th September 2019

## Contents



## 1 Introduction

The parallel 3D implicit MPM program is developed to solve large deformation problem with complex geometry. This program is written in C++ with integrating some scientific computing libraries and tools for different purposes:

- `boost` is used for most of vectors or matrices, e.g. coordinates of mesh nodes and connectivities of elements. With `boost`, many operators for vectors and matrices are available, e.g. getting rows of matrix, computation of determinant.
- `MOAB` or Mesh Oriented datABase is used to manage the computational mesh. At the beginning of simulation, the mesh is saved into a `MOAB` structure. With that structure, the operation on mesh is convenient. For example, the neighbourhood of tetrahedral mesh can be built efficiently.
- `PETSc` is the solver for the system of nonlinear equations, after discretization of the partial differential equations.

The current parallelism is with the MPI over multiple processes.

This manual is organised as follows. Section 2 introduces a simple example to demonstrate how to run sequentially this program. Section 3 includes an example to show how to run this program in parallel. Section 4 introduces the code structure and how to develop new functions in this program.

Notably, this user manual assumed that you have the programming environment set up properly in advance. If you need to set up that, please refer to Appendix A.

## 2 Getting started with a simple example

This section introduces how to perform a numerical analysis with the proposed pMPM through a simple example. The procedure of performing a numerical analysis and where they can be done when using pMPM is shown in Figure 1. Each step is demonstrated in this example for simulation of the simple stretch. As shown in Figure 2 (a), the problem domain is a cuboid, with edge length 2 m. It is stretched along one direction in the simulation, see Figure 2 (b).

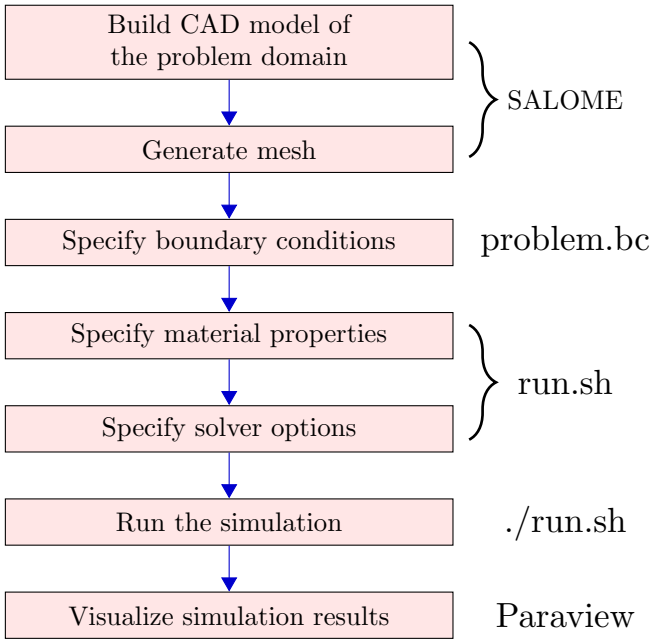


Figure 1: The procedures of performing a numerical analysis and where each of them is done when using pMPM on Linux. SALOME and Paraview are software, 'problem.bc' and 'run.sh' are text files.

The CAD model of a cuboid is built and tetrahedral mesh of it is generated in SALOME, as shown in Figure 3. The linear mesh is firstly generated and then converted to quadratic one. The nodes with boundary conditions are grouped

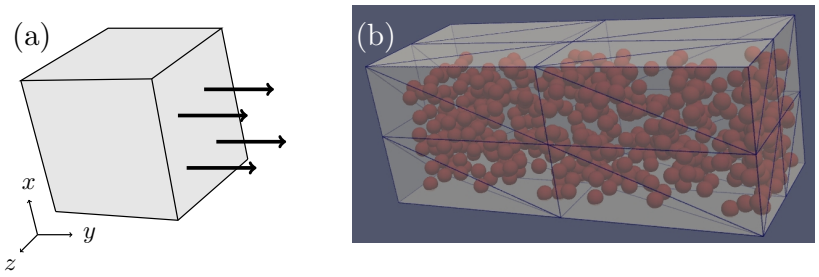


Figure 2: Simple stretch: (a) geometry and (b) deformed configuration with 6.8m displacement.

and there groups are named, by 'x0', 'y0', 'z0' and 'y1', highlighted in Figure 3. This quadratic mesh with groups of nodes are explored to 'box.unv'.

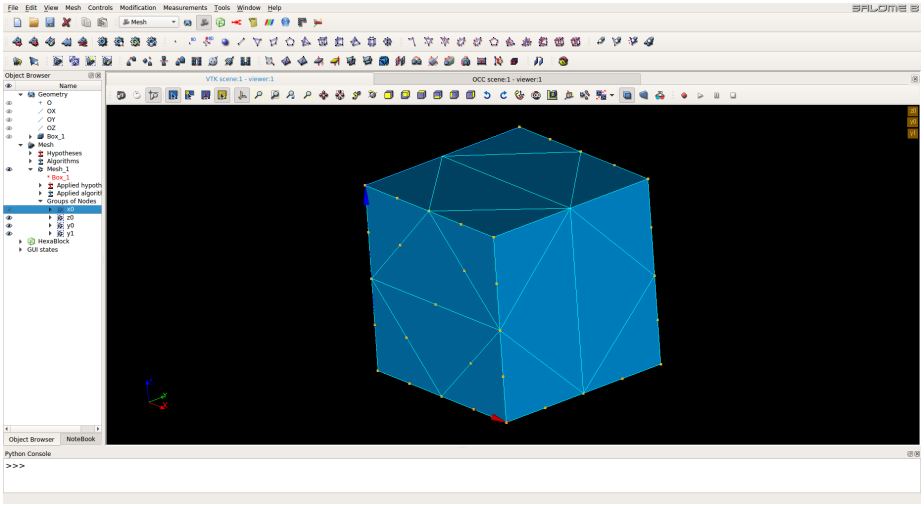

Figure 3: CAD is built and quadratic mesh is generated in SALOME with nodes grouped for boundary conditions.

Roller boundary conditions were applied on three surfaces, $x = 0, y = 0, z = 0$, and an incremental displacement of $\Delta u = 0.4\,\text{m}$ per load step was applied on the surface, $y = 2\,\text{m}$, as shown in Figure 2. These are specified in 'box.bc', as shown in File 2.1. In 'box.bc', the groups name in 'box.unv' are used for linking to where the boundary condition is applied. As shown in File 2.1, roller boundary condition is applied on nodes in groups 'x0', 'y0' and 'z0', and displacement boundary condition is specified for 'y1'. The moving mesh strategy is that the mesh is simply stretched along the $y$-axis direction such that the boundary of the physical domain aligns with the mesh. Notably, the names of these files except the extensions should be the same.

File 2.1: box.bc

```
# This file contains Boundary Condition information assigned to mesh groups of a unv mesh file

*DirchletBC begin
  x0 XDISP 0
  y0 YDISP 0
  z0 ZDISP 0
  y1 YDISP 4e-1
*DirechletBC end
```

The material uses a finite strain elasto-plastic associated flow constitutive model with a von Mises yield function. This is specified in the 'run.sh' in File 2.2. As seen in Line 15 of File 2.2, Young's modulus $E = 10^3$ Pa is specified by '-E', $\nu = 0.0$ by '-nu', and $f_c = 400$ Pa by '-fc'. The explanation of other user options in 'UserOpt' is listed in Table 2. For a complete list of available options, please refer to Appendix B.

File 2.2: run.sh

```
#!/bin/bash

# How to run this script?
#   ./run.sh 4 > run.log &
# will run the probelm with 4 processes and write the output to run.log

if [ $# -eq 0 ]
then
    nP=1
else
    nP=$1  # get the number of processes from input argument
fi

ExcuPath=..
InputPath=input
OutputPath=results_files
Input=box

[ -d $OutputPath ] && rm -rf $OutputPath/*

UserOpt='-writeMeshMp 1 -EXP 5 -GNL 1 -MNL 1 -E 1e3 -nu 0 -fc 4e2 -withFbpMesh 1 -withFbpSNES 1 -mppe 11 -autoDt 0 -steps 2 -debugoutput 0
```

```
    '
PCOpt='-pc_type lu  -pc_factor_mat_solver_type
    superlu_dist'
KSPOpt='-ksp_type gmres  -ksp_atol 1e-6 -
    ksp_rtol 1e-6  -ksp_monitor -
    ksp_monitor_true_residual -
    ksp_converged_reason'
SNESOpt='-snes_type newtonls -snes_monitor -
    snes_converged_reason -snes_max_it 40 -
    snes_max_funcs 40 -snes_atol 1e-6 -snes_rtol
    1e-6 -snes-stol 1e-6 -snes_linesearch_type
    basic'

time -p mpiexec -np $nP $ExcuPath/pMPM -
    Cmeshfile $InputPath/${Input}.unv -savefile $
    {Input} $UserOpt $PCOpt $KSPOpt $SNESOpt |
    tee $OutputPath/${Input}.log
```

Table 1: Meaning of user options in File 2.2

| option | meaning of the value here |
|---|---|
| writeMeshMp | save *.vtk for mesh and material points per step |
| EXP | indicate this example for directing corresponding moving mesh strategy and output |
| GNL | use finite strain |
| MNL | indicator of von Mises material model |
| E | Young's modulus |
| nu | Poission's ratio |
| fc | yield strength |
| withFbpMesh | input patch mesh and then computational mesh automatically is generated in code |
| withFbpSNES | use $\bar{F}$-patch method |
| mppe | number of material points per element |
| autoDt | not use automatic step size strategy |
| steps | number of load steps |
| debugoutput | not output debugging information |

The simulation can be run by './run.sh' in the Linux terminal. There will be some output on the terminal to show the model information and solving procedure. Also, there are many output files are saved in the directory 'results_files', including

**'box_mesh_$i$.vtk', $i$=1,2,$\cdots$:** the computational mesh at the beginning of the $i$-th load step, with nodal incremental displacement from previous load step,

**'box_mp_$i$.vtk', $i$=1,2,$\cdots$:** the material points at the beginning of the $i$-th load step, with total displacement, material point volume, logarithm

strain, Kirchhoff stress and Cauchy stress at material points,

**'residual.txt':** the L2 norm of residual or out-of-balance force for each step and iteration,

**'time_processes_$p$.txt':** simulation time for locating material points and build system of equations on the process $p$ in each step. Notably, this is a real simulation time, or real CPU time, without including waiting time.

**'time_assembly.txt':** time for computing Jacobian or tangential global stiffness matrix and out-of-balance force vector in each iteration,

**'ss_utotal_tau22.txt':** for step, incremental displacement, total displacement from boundary conditions and the normal component of Kirchhoff stress $\tau_{22}$ along the stretch direction at the first material point. Notably, this file is example specific output. Different values of EXP in Line 15 of File 2.2 will lead to save different interested values to output files.

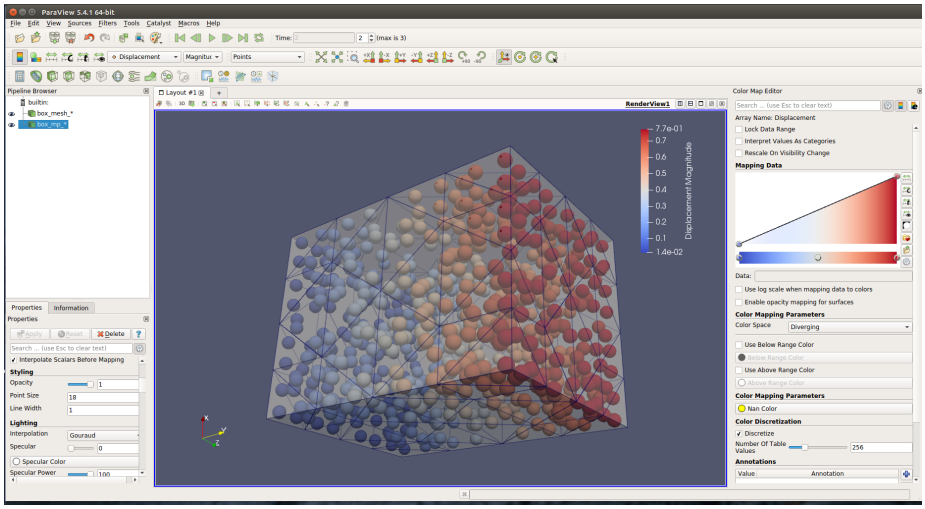


Figure 4: The computational mesh and material points with total displacement at the beginning of the third load step are visualized in Paraview.

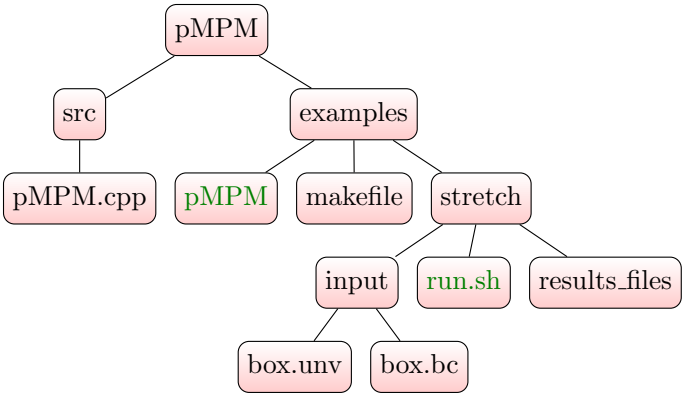


Figure 5: Files structure involving a minimal example – 'stretch'. `pMPM` is the executable, built from the source code `pMPM.cpp` by `make all` in the directory 'examples'. Running the simulation of the simple stretch of a box by `./run.sh` reads the model in the directory 'input', and saves all outputs in the directory 'results_files'.

# 3 How to run pMPM in parallel

This section introduces an examples to demonstrate how to run a simulation with pMPM in parallel. The example is double-notched plate. As shown in Figure 6, the double-notched plate is stretched. The physical domain is firstly discretized to the patch mesh (linear mesh) in (b), which is saved to 'plate.dat'. The sketch of this physical domain and discretization can be done automatically by running the Python script in SALOME by

**Tip 3.1: Automatically generate mesh through SALOME**

File $\longrightarrow$ Load Script... $\longrightarrow$ Choose 'plate.py'
File $\longrightarrow$ Connect

After loading this model into the SALOME, select the 'Mesh' module, the patch mesh, which is a linear mesh, can be visualized. Exporting this mesh 'Mesh 1' into the file 'plate.dat' in the directory 'input'. When running the simulation, pMPM reads 'plate.dat' and use METIS to perform the decomposition as shown in 6 (b).

In SALOME, this linear patch mesh will be converted to the quadratic mesh, for including nodes at the middle of edges. This quadratic mesh will exported into 'plate.unv' as the mesh file. The computational linear mesh will be generated inside of pMPM to divide each 10-node quadratic tetrahedron to eight linear tetrahedral element. The corresponding procedure in SALOME is summarized in Tip 3.2.

**Tip 3.2: Export mesh file**

Switch two 'Mesh' module on the second row of menu ⟶ right click on 'Mesh 1' ⟶ 'Convertto/from quadratic' ⟶ Covert to quadratic ⟶ Apply and close.
Select all groups under 'Groups of Nodes' ⟶ right click on them ⟶ 'Update' to update the contents of these groups for including additional mid-edge nodes.
right click on 'Mesh 1' ⟶ Export to 'plate.unv'.

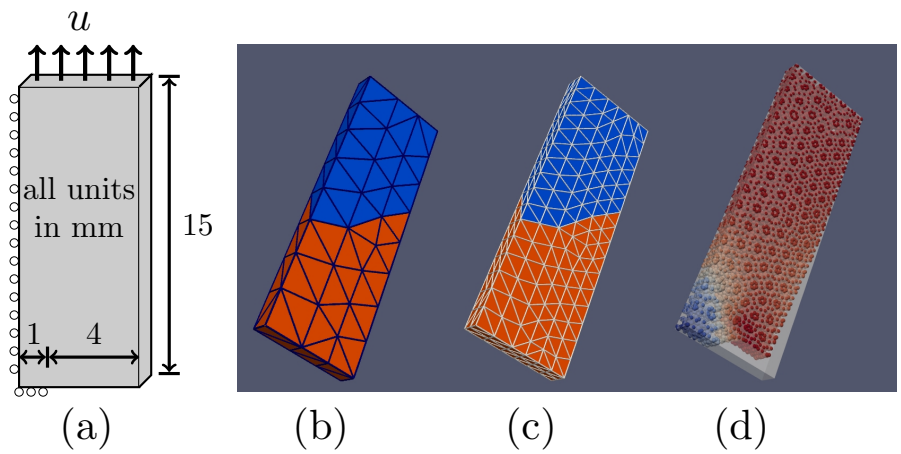


Figure 6: Double-notched plate: (a) geometry and boundary conditions, the roller boundary condition on both back and front surfaces is not shown; (b) the patch mesh with colors indicating on two processes; (c) the computational mesh, generated from the patch mesh; and (d) the material points in the deformed configuration with $u = 1\,\text{mm}$.

The associated boundary conditions are specified in 'plate.bc' (File 3.1). The material parameters and solver options are specified in 'run.sh' (File 3.2). Running this bash script will generate simulation results in directory 'results_files'. With the output '*.vtk' files in directory 'results_files', the computational mesh and material points fields can be visualised in Paraview, e.g. see 6 (d). With the 'EXP', pMPM also outputs the reaction force at the end, on which the displacement boundary condition is applied. This is called 'ss_utotal_fy_fnorm_rank$i$', where $0 \leqslant i < np$, the id of process. These contents of these values from different processes should be the same. This data can be plotted in Matlab to generate Figure 7, by running the enclosed 'plot_plate_fr.m' in the directory 'matlab'.

**Tip 3.3: Run simulation with $np$ processes in parallel**

./run.sh $np$ > run.log &

File 3.1: plate.bc

```
*DirchletBC begin
  x0 XDISP 0
  y0 YDISP 0
  z  ZDISP 0
  y1 YDISP 0.01
*DirechletBC end
```

File 3.2: run_plate.sh

```
#!/bin/bash

# How to run this script?
#   ./run.sh 4 > run.log &
# will run the probelm with 4 processes and
   write the output to run.log

if [ $# -eq 0 ]
then
    nP=1
else
    nP=$1  # get the number of processes from
       input argument
fi

ExcuPath=..
InputPath=input
OutputPath=results_files
Input=plate


[ -d $OutputPath ] && rm -rf $OutputPath/*


UserOpt='-writeMeshMp 1 -EXP 8 -GNL 1 -MNL 1 -E
   206.9 -nu 0.29 -fc 0.551135192126215 -
   withFbarPatch 1 -withFbarPatchSNES 1 -mppe 11
    -autoDt 0 -steps 101 -debugoutput 0'
PCOpt='-pc_type lu  -pc_factor_mat_solver_type
   superlu_dist'
KSPOpt='-ksp_type gmres  -ksp_atol 1e-6 -
   ksp_rtol 1e-6  -ksp_monitor -
   ksp_monitor_true_residual -
   ksp_converged_reason'
SNESOpt='-snes_type newtonls -snes_monitor -
   snes_converged_reason -snes_max_it 40 -
```

```
    snes_max_funcs 40 -snes_atol 1e-6 -snes_rtol
    1e-6 -snes-stol 1e-6 -snes_linesearch_type
    basic'

time -p mpiexec -np $nP $ExcuPath/pMPM -
    Cmeshfile $InputPath/${Input}.unv -savefile $
    {Input} $UserOpt $PCOpt $KSPOpt $SNESOpt |
    tee $OutputPath/${Input}.log
```

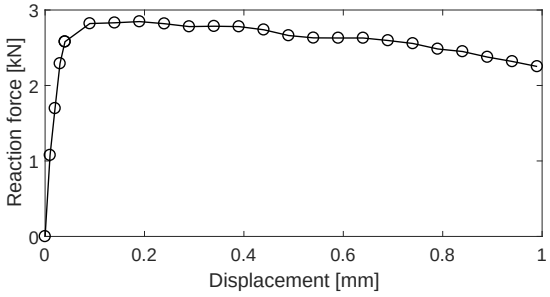


Figure 7: Double-notched plate: magnitude of reaction force against the displacement.

# 4 Code structure and new development

This section introduces the code structure and some tips on new development. With this information, the developer is able to understand where are the gateway for developing new functions and how to implement them.

The code structure for programming of the pMPM is shown in Figure 8. The main code of the program is `pMPM.cpp`. Many user-defined functions are included in `functions.cpp`. If some functions are inserted in `functions.cpp`, the corresponding declaration of these functions should be added into the `functions.hpp`. The `NRFunctions.cpp` includes all functions for the Newton Raphson iteration, e.g. computation of Jacobian (tangential stiffness matrix) and functions (out-of-balance-force). The `unvReader.cpp` includes functions to read unv mesh file into the program and save that to a MOAB structure. New material models can be inserted into `materialModels.cpp`.

The `pMPM.cpp` can be classified to several parts as follows

**Tip 4.1: Procedures in `pMPM.cpp`**

- set up problem to read all input files into variables in the pMPM,
- for each load step,

– output mesh and material point into files,
– call SNESSolve to solve the system of nonlinear equations,
– update material point field,
– apply moving mesh strategy.

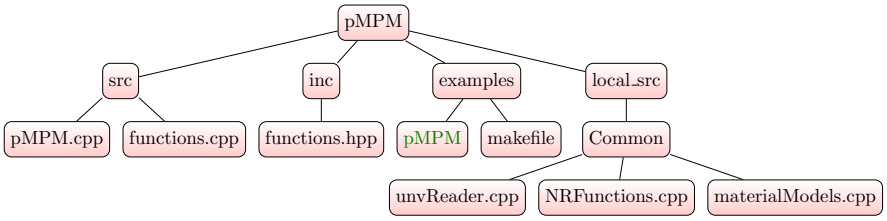


Figure 8: Files structure for programming in the pMPM. The main function is `pMPM.cpp` with some user defined functions in `functions.cpp`. The `NRFunctions.cpp` includes computation of Jacobian and functions.

# Appendix A Installation of the program

This section introduces how to install the program into your local computer, including how to download and configure your programming environment.

All of documents of this program can be downloaded from Github with the address https://github.com/sanshi2000/pMPM. It can be directly downloaded by clicking on 'Clone or download' as a ZIP file or using git to clone the repository. A useful tutorial for the later method can be found here.

After downloading the code into your computer, please follow the necessary steps in `pMPM/tools/Ubuntu_configuration/README` for configuration.

# Appendix B User options

This section introduces available user options and the meaning of different values. The first value for each option is the default value, which is used if the option is not used.

Table 2: Meaning of user options in File 2.2

| option | values | meaning |
|---|---|---|
| writeMeshMp | 0 | not output mesh and material points files |
| | 1 | save *.vtk for mesh and material points |
| writeInc | 1 | save mesh and material points to *.vtk per step |
| | $n$ | save mesh and material points to *.vtk per $n$ step |
| EXP | 1–12 | indicate an example for specifying corresponding moving mesh strategy and output, see `pMPM/examples/README` for details |
| GNL | 0 | use infinitesimal strain |
| | 1 | use finite strain |
| MNL | 1–3 | indicator of material model |
| E | $E$ | Young's modulus |
| nu | $\nu$ | Poission's ratio |
| fc | $f_c$ | yield strength |
| withFbpMesh | 0 | input linear mesh for standard MPM |
| or withFbarPatch | 1 | input patch mesh and then computational mesh automatically is generated in code |
| withFbpSNES | 0 | not use $F$-patch formulation, i.e. use standard MPM |
| or withFbarPatchSNES | 1 | use $\bar{F}$-patch method |
| mppe | 1,4, or 11 | number of material points per element |
| autoDt | 0 | not use automatic step size, i.e. used fixed step size |
| | 1 | use automatic step size strategy |
| steps | $s$ | number of maximum load steps |
| debugoutput | 0 | not output debugging information |
| | 1 | output debugging information |