PDLSM-FEM MANUAL

VERSION 1.0

Qibang Liu, X. J. Xin

qibangliu@ksu.edu

Aug. 2021

Kansas State University
Mechanical & Nuclear Engineering

Content

1 Introduction

PDLSM-FEM solver is an open-source parallel implementation of coupled peridynamics least squares minimization and finite element method (PDLSM-FEM) in 2D and 3D by MPI technique. This solver is written in a C++ environment on cross platforms. It includes implicit static, explicit dynamic, and implicit dynamic solvers for structure analysis under displacement or traction loading.

- **Pre-process**. PDLSM-FEM solver requires a mesh data file with the specified format. Users may use commercial finite element codes such as ANSYS, ABAQUS to generate the mesh data file.
- **Solver**. PDLSM-FEM solver stores the large sparse matrix in a compressed sparse row (CSR) format and solves the large linear systems of equations by the ParallelDirect Sparse Solver (PDSS) of Intel Math Kernel Library (MKL).
- **Post-process**. PDLSM-FEM solver stores the results in ASCII VTK format or BINARY VTK format, which can be visualized directly by Paraview, as shown in Fig. 1.

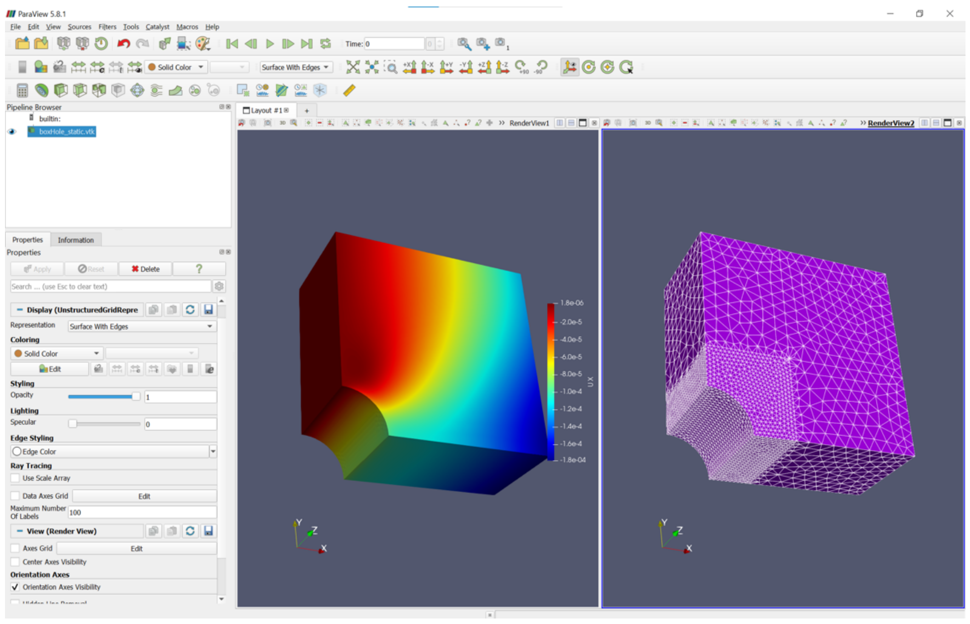


Fig. 1: Data visualization by paraview.

The PDLSM-FEM theory is briefly presented in Section 4, and more details can be found in the paper [1, 2] published by the authors.

2 Build and Run PDLSM-FEM

PDLSM-FEM is built as an executable from source code using a build environment generated by CMake. PDLSM-FEM is parallelized by message passing interface (MPI), thus you may install MPI libraries (such as OpenMPI, MPICH, MPICH2, Intel MPI, MS MPI) on your machine first. In PDLSM-FEM, the large sparse matrix is solved by parallel direct sparse solver (PDSS) of Intel Math Kernel Library (MKL). Thus, it is required to install MKL on your machine before build the source code. Overall, prerequisites are as below:

1. CMake
2. C++ compiler;
3. MPI library;
4. MKL.

For Windows platform, we use the MS MPI, on Linux platform, we use OpenMPI and on macOS, it uses MPICH. If you do not use the same libraries, you may need to modified the below code (including the NAMES and HINTS) in the file FindMPI.cmake under the folder 'cmake' of PDLSM-FEM, to make sure it finds the correct libraries.

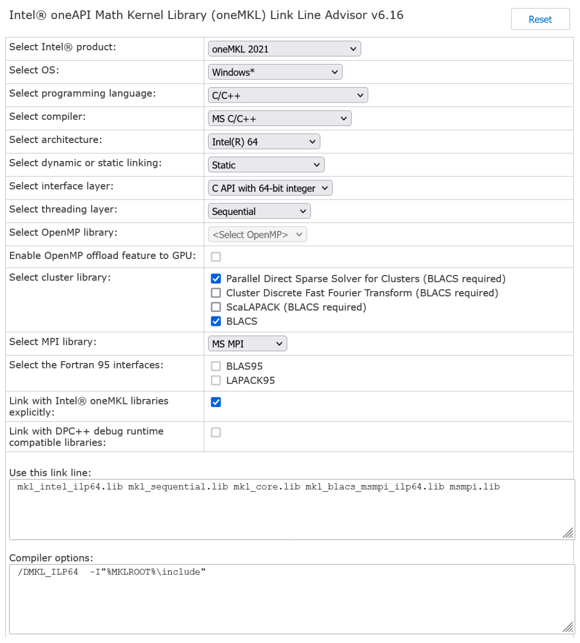
```
#===set link MPI libraries=====================
if(WIN32)
   find_library(MPI_LIBRARY NAMES msmpi.lib  HINTS $ENV{MPIROOT}/Lib/x64)
elseif(UNIX AND NOT APPLE)
   find_library(MPI_LIBRARY NAMES libmpi.so  HINTS $ENV{MPIROOT}/lib/release
   $ENV{MPIROOT}/lib)
elseif(APPLE)
   find_library(MPI_LIBRARY NAMES libmpi.dylib  HINTS $ENV{MPIROOT}/lib/release
    $ENV{MPIROOT}/lib)
endif()
```

For MKL, the required libraries depend on many parameters. Please use the MKL link advisor to check the required libraries and compiler options on your platform as illustrated in Fig. 2. As shown in the link line and compiler options of the Fig. 2b, the required libraries are

```
${MKLROOT}/lib/intel64/libmkl_intel_ilp64.a ${MKLROOT}/lib/intel64/
 libmkl_sequential.a ${MKLROOT}/lib/intel64/libmkl_core.a ${MKLROOT}/lib/
 intel64/libmkl_blacs_openmpi_ilp64.a
```

and compiler options are

```
-lpthread -lm -ldl -DMKL_ILP64  -m64
```



Intel® oneAPI Math Kernel Library (oneMKL) Link Line Advisor v6.16 Reset

Select Intel® product:	oneMKL 2021
Select OS:	Windows*
Select programming language:	C/C++
Select compiler:	MS C/C++
Select architecture:	Intel(R) 64
Select dynamic or static linking:	Static
Select interface layer:	C API with 64-bit integer
Select threading layer:	Sequential
Select OpenMP library:	<Select OpenMP>
Enable OpenMP offload feature to GPU:	☐
Select cluster library:	✓ Parallel Direct Sparse Solver for Clusters (BLACS required) ☐ Cluster Discrete Fast Fourier Transform (BLACS required) ☐ ScaLAPACK (BLACS required) ✓ BLACS
Select MPI library:	MS MPI
Select the Fortran 95 interfaces:	☐ BLAS95 ☐ LAPACK95
Link with Intel® oneMKL libraries explicitly:	✓
Link with DPC++ debug runtime compatible libraries:	☐

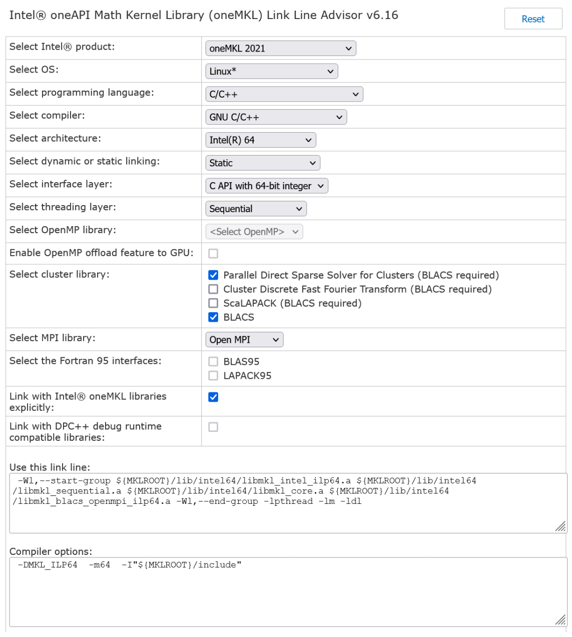
Use this link line:

```
 mkl_intel_ilp64.lib mkl_sequential.lib mkl_core.lib mkl_blacs_msmpi_ilp64.lib msmpi.lib
```

Compiler options:

```
 /DMKL_ILP64  -I"%MKLROOT%\include"
```

(a)



Intel® oneAPI Math Kernel Library (oneMKL) Link Line Advisor v6.16 Reset

Select Intel® product:	oneMKL 2021
Select OS:	Linux*
Select programming language:	C/C++
Select compiler:	GNU C/C++
Select architecture:	Intel(R) 64
Select dynamic or static linking:	Static
Select interface layer:	C API with 64-bit integer
Select threading layer:	Sequential
Select OpenMP library:	<Select OpenMP>
Enable OpenMP offload feature to GPU:	☐
Select cluster library:	✓ Parallel Direct Sparse Solver for Clusters (BLACS required) ☐ Cluster Discrete Fast Fourier Transform (BLACS required) ☐ ScaLAPACK (BLACS required) ✓ BLACS
Select MPI library:	Open MPI
Select the Fortran 95 interfaces:	☐ BLAS95 ☐ LAPACK95
Link with Intel® oneMKL libraries explicitly:	✓
Link with DPC++ debug runtime compatible libraries:	☐

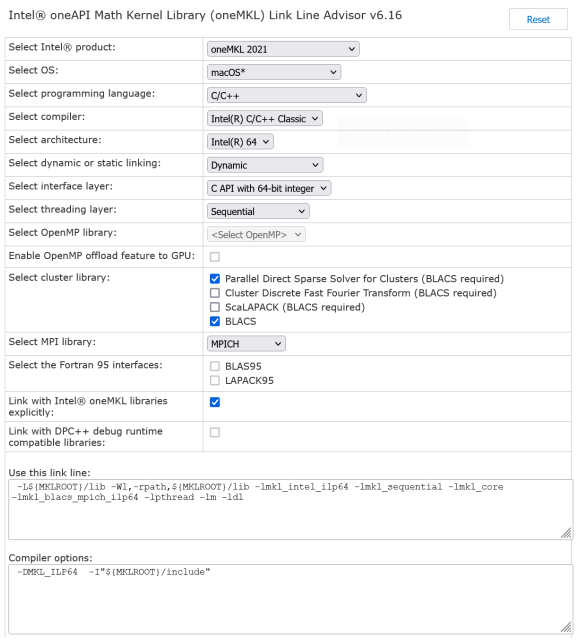
Use this link line:

```
 -Wl,--start-group ${MKLROOT}/lib/intel64/libmkl_intel_ilp64.a ${MKLROOT}/lib/intel64
/libmkl_sequential.a ${MKLROOT}/lib/intel64/libmkl_core.a ${MKLROOT}/lib/intel64
/libmkl_blacs_openmpi_ilp64.a -Wl,--end-group -lpthread -lm -ldl
```

Compiler options:

```
 -DMKL_ILP64  -m64  -I"${MKLROOT}/include"
```

(b)



Intel® oneAPI Math Kernel Library (oneMKL) Link Line Advisor v6.16 Reset

Select Intel® product:	oneMKL 2021
Select OS:	macOS*
Select programming language:	C/C++
Select compiler:	Intel(R) C/C++ Classic
Select architecture:	Intel(R) 64
Select dynamic or static linking:	Dynamic
Select interface layer:	C API with 64-bit integer
Select threading layer:	Sequential
Select OpenMP library:	<Select OpenMP>
Enable OpenMP offload feature to GPU:	☐
Select cluster library:	✓ Parallel Direct Sparse Solver for Clusters (BLACS required) ☐ Cluster Discrete Fast Fourier Transform (BLACS required) ☐ ScaLAPACK (BLACS required) ✓ BLACS
Select MPI library:	MPICH
Select the Fortran 95 interfaces:	☐ BLAS95 ☐ LAPACK95
Link with Intel® oneMKL libraries explicitly:	✓
Link with DPC++ debug runtime compatible libraries:	☐

Use this link line:

```
 -L${MKLROOT}/lib -Wl,-rpath,${MKLROOT}/lib -lmkl_intel_ilp64 -lmkl_sequential -lmkl_core
-lmkl_blacs_mpich_ilp64 -lpthread -lm -ldl
```

Compiler options:

```
 -DMKL_ILP64  -I"${MKLROOT}/include"
```

(c)

Fig. 2: Illustration of link advisor: (a) for windows; (b) for Linux; (c) for MacOS.

Note that if you use different parameters such as different MPI version, you have to use the MKL link advisor to find out the required the libraries and compiler options, and modified the below code in the file "FindMKL.cmake" for required libraries

```
#===set link MKL libraries=====================
if(WIN32)
    file(GLOB  MKL_LIBRARY
    $ENV{MKLROOT}/lib/intel64/mkl_intel_ilp64.lib
    $ENV{MKLROOT}/lib/intel64/mkl_core.lib
    $ENV{MKLROOT}/lib/intel64/mkl_sequential.lib
    $ENV{MKLROOT}/lib/intel64/mkl_blacs_msmpi_ilp64.lib)
elseif(UNIX AND NOT APPLE)
    file(GLOB  MKL_LIBRARY
    $ENV{MKLROOT}/lib/intel64/libmkl_intel_ilp64.so
    $ENV{MKLROOT}/lib/intel64/libmkl_core.so
    $ENV{MKLROOT}/lib/intel64/libmkl_blacs_openmpi_ilp64.so
    $ENV{MKLROOT}/lib/intel64/libmkl_sequential.so)
elseif(APPLE)
    file(GLOB  MKL_LIBRARY
    $ENV{MKLROOT}/lib/intel64/libmkl_intel_ilp64.dylib
    $ENV{MKLROOT}/lib/intel64/libmkl_sequential.dylib
    $ENV{MKLROOT}/lib/intel64/libmkl_core.dylib
    $ENV{MKLROOT}/lib/intel64/lmkl_blacs_mpich_ilp64.dylib)
endif()
```

and modified the compiler options at the end of the file of "CMakeList.txt" as:

```
#===set compiler options=====================
if(WIN32)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /DMKL_ILP64")
elseif(UNIX AND NOT APPLE)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -lpthread -lm -ldl -DMKL_ILP64 -m64
   ")
elseif(APPLE)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -lpthread -lm -ldl -DMKL_ILP64 -m64
   ")
endif()
```

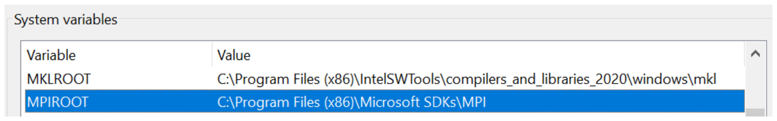
2.1 Build on Windows

In this subsection, we illustrate how to build PDLSM-FEM with visual studio and CMake GUI on Window. Building PDLSM-FEM with CMake is a three-step process. First, use CMake to generate a build environment in a new directory. For that purpose, you may use either the command-line utility cmake or the graphical utility cmake-GUI.

Before creating the build environment, it needs to set two environment variables named MKLROOT and MPIROOT, and their values are equal to the installed root directories of MKL and MPI as shown in Fig. 3. To ensure you set the correct value of MPI-

ROOT, please make sure under the path of MPIROOT, there are files 'MPIROOT/include/mpi.h' and 'MPIROOT/lib/release/impi.lib'. If you do not use the MS MPI, you may modify the below codes in file 'FindMPI.cmake' (includes the NAMES and the HINTS) to make sure it finds the correct library.

```
#===set link MPI libraries=====================
if(WIN32)
   find_library(MPI_LIBRARY NAMES msmpi.lib  HINTS $ENV{MPIROOT}/Lib/x64)
elseif(UNIX AND NOT APPLE)
   find_library(MPI_LIBRARY NAMES libmpi.so  HINTS $ENV{MPIROOT}/lib/release
   $ENV{MPIROOT}/lib)
elseif(APPLE)
   find_library(MPI_LIBRARY NAMES libmpi.dylib  HINTS $ENV{MPIROOT}/lib/release
    $ENV{MPIROOT}/lib)
endif()
```

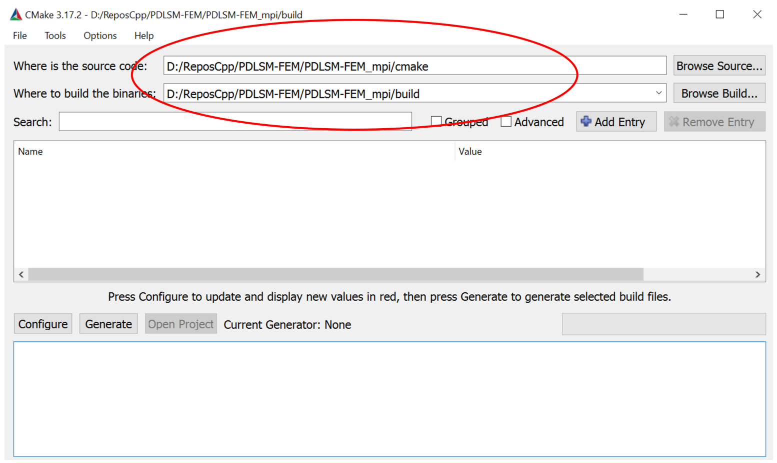


System variables

Variable	Value
MKLROOT	C:\Program Files (x86)\IntelSWTools\compilers_and_libraries_2020\windows\mkl
MPIROOT	C:\Program Files (x86)\Microsoft SDKs\MPI

Fig. 3: Environment variables.

Then go to the CMake-GUI, and input the location of the source file and where to build the binaries, as Fig. 4 shows. After that, click the configure button and select visual studio as compiler. After configuring is done, click the Generate button and Open Project button. Then build the source code in Visual studio.



CMake 3.17.2 - D:/ReposCpp/PDLSM-FEM/PDLSM-FEM_mpi/build

Where is the source code: D:/ReposCpp/PDLSM-FEM/PDLSM-FEM_mpi/cmake

Where to build the binaries: D:/ReposCpp/PDLSM-FEM/PDLSM-FEM_mpi/build

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

Fig. 4: Generate a build environment.

2.2 Build on Linux

In this subsection, we illustrate how to build PDLSM-FEM on a Linux platform using command-line utility cmake. You may also use cmake-GUI as shown in previous subsection.

First, the two environment variables MPIROOT and MKLROOT are required by adding two commands in the .bashrc file as below:

```
export MKLROOT=/homes/qibangliu/intel/oneapi/mkl/2021.3.0
export MPIROOT=/opt/software/software/OpenMPI/4.0.3-GCC-9.3.0
```

Note that the path value of the environment variables may be different based on the versions and installation paths of MKL and MPI. The principles to set environment variables are that:

(a) Under the path "MKLROOT/include", there is the "mkl.h file", and under the path "MKLROOT/lib/intel64", there are libraries as the MKL link line advisor show, such as "libmkl_intel_ilp64.so", "libmkl_core.so".

(b) Under the path "MPIROOT/include", there is the header file "mpi.h", and under the path "MPIROOT/lib", there is the library "libmpi.so"

Here, the paths "MPIROOT/include" and "MPIROOT/lib" are the "HINTS" defined in the file "FindMPI.cmake" as below

```
 find_path(MPI_INCLUDE_DIR mpi.h HINTS $ENV{MPIROOT}/include)
 find_library(MPI_LIBRARY NAMES libmpi.so  HINTS $ENV{MPIROOT}/lib $ENV{MPIROOT
   }/lib/release)
```

After modified the .bashrc file, you may type the commands to source it:

```
source ~/.bashrc
```

The values of MKLROOT and MPIROOT are the directories where the MKL and MPI libraries are installed. To check the paths are correct, you may type the below commands to check the header files and libraries files are there:

```
ls $MKLROOT/include
ls $MKLROOT/lib/intel64
ls $MPIROOT/include
ls $MPIROOT/lib
```

The next step is the compilation and linking of all objects, libraries, and executables. Here is a minimal example using the command line version of CMake to build PDLSM-FEM:

```
cd /homes/PDLSM-FEM_mpi #change to the PDLSM-FEM distribution directory
```

```
mkdir build #create and use a build directory
cd build #change to the build directory
cmake ../cmake #configuration reading CMake scripts from ../cmake
make #compilation
```

2.3 Build on macOS

Building PDLSM-FEM on a Mac platform is similar to that of on Linux platform. First, it requires to set two environment variables MPIROOT and MKLROOT, following the setting principle as described in subection 2.2. Note that we set the cmake files for MPICH library, if you use different libraries, you may have to modify the file 'Find-MPI.cmake' as described at the head of the section. Also, if the parameters, such as compiler, architecture, MPI version are different with Fig. 2c showing, you may have to modify the files 'FindMKL.cmake' and 'CMakeList.txt' as described at the head of the section. After that you will use CMake to build the codes as:

```
cd PDLSM-FEM_mpi #change to the PDLSM-FEM distribution directory
mkdir build #create and use a build directory
cd build #change to the build directory
cmake ../cmake #configuration reading CMake scripts from ../cmake
make #compilation
```

2.4 Run PDLSM-FEM

The executable PDLSM-FEM can be run in serial or in parallel using MPI with one of the following commands:

```
cd build #change to the build directory
PDLSM-FEM PDFEMcmd.in #For serial run on Linux
mpirun -n 4 PDLSM-FEM PDFEMcmd.in #For parallel run on Linux
PDLSM-FEM.exe PDFEMcmd.in #For serial run on Wnindows
mpiexec -np 4 PDLSM-FEM.exe PDFEMcmd.in #For parallel run on Windows
```

Here the "PDFEMcmd.in" is a command file to set the parameters for PDLSM-FEM solver, and the details of the command file are introduced in Section 3.1. Note that we only test that run PDLSM-FEM in serial on Windows, but not in parallel. If parallel running on windows does not work, we suggest installing Windows Subsystem for Linux on windows platform and then parallel run PDLSM-FEM in a Linux environment.

All the results data are written in the folder "Results" which is created automatically. If the folder is empty, please check the permission of the folder, and add write permission to it if necessary.

3 Input files

PDLSM-FEM requires two input files. One is the mesh file which contains the material properties, nodes coordinates, elements connectivities, boundary conditions, crack information, etc, and another one is the command file which is for configuring the solver.

3.1 Commands

The commands for configuring the PDLSM-FEM solver are introduced one by one in this section.

```
# read mesh data from mesh file "filename". the first command must be MSHFILE.
MSHFILE filename
#Set solver. Solver can be DYNAMIC, STATIC, or QUASI-STATIC.
SOLVER QUASI-STATIC
# set solving parameter.
#syntax: SETSOLVING (time incremental)  (total time steps)  (write  results
    frequency)  m  a;
SETSOLVING 1 72 2 3 0.33
```

For static solver, it will output three files of 'project_PD.vtk', 'project_FEM.vtk', and 'project_static.vtk', which include the results of PD domain, that of FEM domain and that of the whole domain, respectively. For dynamic and quasi-static solver, only the results of the whole domain are obtained.

The parameter m is to determine the horizon as $\delta_{(i)} = m\Delta_{(i)}$, and a is the non-local factor for the weight function:

$$\omega(|\boldsymbol{\xi}|) = e^{-\left(\frac{|\boldsymbol{\xi}|}{a\delta_{(i)}}\right)^2}. \tag{1}$$

The non-local factor may affect the accuracy in numerical implementation and may also affect the stability for daynamic solving. For quasi-static solver, the time incremental is set as 1 s.

```
# Newmark's method: ON or OFF, default is OFF;
#syntax: NEWMARK ON gamma beta
NEWMARK OFF
# lumped mass: ON or OFF; Only available for explicit dynamic solver; default
    is OFF:
LUMPMASS ON
```

PDLSM-FEM solver enables explicit and implicit solving (Newmark's method) for dynamic problems. The standard Newmark's method assumes that

$$\mathbf{U}_{t+\Delta t} = \mathbf{U}_t + \Delta t \dot{\mathbf{U}}_t + \Delta t^2 \left[\left(\frac{1}{2} - \beta \right) \ddot{\mathbf{U}}_t + \beta \ddot{\mathbf{U}}_{t+\Delta t} \right] \tag{2}$$

$$\dot{\mathbf{U}}_{t+\Delta t} = \dot{\mathbf{U}}_t + \Delta t \left[(1-\gamma) \ddot{\mathbf{U}}_t + \gamma \ddot{\mathbf{U}}_{t+\Delta t} \right] \tag{3}$$

where γ and β are constants optionally selected by analyst, and the default values are $\gamma = 0.5$ and $\beta = 0.25$. The keyword *NEWMARK* is for setting the implicit dynamic solver, and the value *OFF* means explicit dynamic solver.

The mass matrix $\mathbf{M}^{pd}$ from PD contribution is a lumped diagonal mass matrix, while $\mathbf{M}^{fem}$ from FEM contribution can be either lumped diagonal or consistent mass matrix. For explicit dynamic solutions, using lumped diagonal mass matrix $\mathbf{M}^{fem}$ may be more efficient since it is easier to solve for diagonal $\mathbf{M}^{-1}$. The keyword *LUMPMASS* is for setting the mass matrix format.

```
# Initialize essential boundary condition;
# syntax: Keyword, ID, value.
EBC 0  0
EBC 1  0
# set incremental for essential boundary condition:
# syntax: Keyword ID velocity.
VEBC 0 -1e-5
VEBC 1  1e-5
# Initialize natural BC;
# syntax: Keyword ID value.
NBC 0  1e6
# set incremental for natural boundary condition:
# syntax: Keyword ID velocity.
VNBC 0 4e6
```

The boundary conditions (BCs) are set through keywords: *EBC*, *VECB*, *NBC* and *VNBC* as shown above. The *ID* represents a set of nodes for essential BCs, or a set of boundary elements for natural BCs. The lists of the sets are stored in the mesh file (see Section 3.2).

```
# Failure criteria. syntax: FC flag
# Flags: 0-- no fialure allowed;
# Flags: 1--maximum circumferential tensile stress;
# Flags: 2-- maximum principal stress.
# syntax: FC 1 m_r d_cf, (m_r, d_cf are optional). Example:
FC 1 6.0 1.0
#syntax: FC 2 TopK. Example:
FC 2 4
```

PDLSM-FEM enables two failure criteria only for 2D problems. More failure criteria for 2D and 3D problems are still under development. The criterion of maximum circumferential tensile stress is only for the quasi-static solver. The parameter mr is to determine the integral path of J-integral and dcf is to determine crack growth amount at each step. If they are not explicitly set, the default values $m_r = 6.0$ and $d_{cf} = 1.0$ are used. More details about this failure criterion can be found in Section 4.6.1.

The criterion of maximum principal stress is for both quasi-static solver and dynamic solver but requires an appropriate *Topk* value. *Topk* represents the number of *Topk* bonds with the largest value that will be handled for failure. $Topk == 0$ means no limitation of bonds. More details about this failure criterion can be found in Section 4.6.2.

```
# Calculate total reaction force. Syntax: RF ID
RF 1
#Write results to vtk file in format BINARY or ASCII. Default is BINARY;
VTKFORMAT ASCII
VTKFORMAT BINARY
```

PDLSM-FEM enables calculation of reaction force through keyword *RF*, and *ID* represents a set of nodes and is corresponds to the EBC ID.

The results of displacement, stress, and local damage φ are written in vtk format files which can be visualized directly by Paraview, as shown in Fig. 1. The data can be written in ASCII or BINARY format. For binary format, the data is written in big-endian form, even though the machines is little-endian.

3.2 Mesh file

The mesh file consists of six basic parts:

1. Header. The header consists of a project name and a label which are character strings terminated by the end-of-line character "\n".
2. Problem type. PDLSM-FEM enables solving of 3D, 2D plane stress, and 2D plane strain problems.
3. Material properties. The material properties part consists of Young's modulus, Poisson's ratio, density, fracture toughness, and ultimate tensile stress.
4. Mesh data. The mesh data includes nodes information mesh connectivity.
5. Boundary conditions. The boundary conditions consists of natural boundary conditions (NBCs) and essential boundary conditions (EBC).
6. No-fail region and pre-exist crack. A no-fail region is introduced in PDLSM-FEM to avoid spurious cracking. The pre-exist cracks are represented by segments for 2D and triangle faces for 3D.

The describing the mesh file formats is shown in below.

- **Header**. The first line of the mesh file is the project name and the second line is the label to define the file, as shown in below:

```
plateCrack
==The plate with a central crack under displacement loading==
```

- **Problem type**:

```
Dimension   problem_type
```

  The Dimension can be equal to '3D' or '2D', and problem_type can be equal to '1' or '2', with '1' representing plane stress and '2' representing plane strain.

- **Material properties** has below format:

```
Youngs_modulus Poissons_ratio density  fracture_toughness
    ultimate_tensile_stress
```

- **Mesh DATA**. The mesh data has below format:

```
Ng Eg
NID_1     x_1   y_1   z_1
NID_2     x_2   y_2   z_2
NID_3     x_3   y_3   z_3
...
...
NID_Ng   x_Ng y_Ng z_Ng
EID_1     EType_1   I_1 J_1   K_1   L_1   # for 2D element
EID_2     EType_2   I_2 J_2   K_2   L_2
EID_3     EType_3   I_3 J_2   K_3   L_3
...
...
```

  here 'Ng' represents the total number of nodes and NIDs represent node ID and they must be subsequent numbers that start from 1. For 2D problem, the z-coordinate is equal to 0.

  'Eg' represents the total number of elements and 'EIDs' represent element ID and they also must be subsequent numbers that start from 1. 'E_type1' represents element type and can be equal to '1' or '2', with '1' representing PD element and '2' represents the finite element. 'I', 'J', 'K', 'L' are connectivities of 2D elements and for triangle elements, K=L. For 3D tetrahedron, the connectivity is: I J K K L L L L. For 3D brick element, the connectivity is: I J K L M N O P. The connectivity order must follow the definition in general FE method.

- **Boundary conditions**. The boundary condition has below format:

```
====================Essential BCs================
Nebc
Ne_1 dof_1 value_1  #EBC_0 ID=0
Ne_2 dof_2 value_2  #EBC_1 ID=1
Ne_3 dof_3 value_3  #EBC_2 ID=2
...
...
Ne_Nebc dof_Nebc value_Nebc #EBC_(Nebc-1) ID=Nebc-1
i_1  j_1  k_1  l_1  m_1  n_1 ... #list node ID for EBC_0
i_2  j_2  k_2  l_2  m_2  n_3 ... #list node ID for EBC_1
i_3  j_3  k_3  l_3  m_3  n_3 ... #list node ID for EBC_2
......
......
i_Nebc  j_Nebc  k_Nebc  l_Nebc ... #list node ID for EBC_(Nebc-1)
====================Natural BCs================
Nnbc
Nn_1   tn_1 #NBC_0 ID=0
Nn_2   tn_2 #NBC_1 ID=1
Nn_3   tn_3 #NBC_2 ID=2
...
...
Nn_Nnbc   tn_Nnbc  #NBC_(Nnbc-1) ID=Nnbc-1
p_1 q_1 r_1 s_1 #list of face element connectivity for NBC_0
p_2 q_2 r_2 s_2 #list of face element connectivity for NBC_0
p_3 q_3 r_3 s_3 #list of face element connectivity for NBC_0
...
...
p_(Nn_1) q_(Nn_1) r_(Nn_1) s_(Nn_1) #list of face element connectivity for
    NBC_0
... ... ... ... #list of face element connectivity for NBC_1
... ... ... ... #list of face element connectivity for NBC_1
... ... ... ...
... ... ... ...
... ... ... ... #list of face element connectivity for NBC_(Nnbc-1)
... ... ... ... #list of face element connectivity for NBC_(Nnbc-1)
```

here 'Nebc' represents the number of Essential BCs, 'Ne_i' represents the number of nodes in the set of 'EBC_i'. Dof can be 'UX', 'UY' or 'UZ'.

'Nnbc' represents the number of natural BCs, 'Nn_i' represents the number of boundary elements which is applied normal traction 'tn_i'. The connectivities of these boundary elements also must be in counter-clockwise order.

The values of 'EBC_i' with ID=i and 'NBC_i' with ID=i can be set by commands *EBC*, *VEBC*, *NBC* and *VNBC*, and EBC_i can be used to calculate reaction force by commands *RF*, as follows:

```
# syntax: Keyword, ID, value.
EBC 0  0e-5.
VEBC 0 -1e0
NBC 0  1e6
VNBC 0 4e6
# Calculate total reaction force. Syntax: RF ID
```

```
RF 0
```

.

- **No-fail region & pre-exist crack**. The No-fail region is a list of node ID, and the bonds between them and their family members are not allowed breakage. The pre-exist crack is represented by 3 points coordinates for 3D and 2 points coordinates for 2D.

  Note that, if the criterion maximum circumferential tensile stress is used, and two ends of a crack are inside of the domain, it needs to split the crack into two cracks. This part has below format:

```
====================NO FAIL region================
Nnf
i_1 i_2 i_3 ... ... i_Nnf # list of nodes ID
====================pre-exist crack================
Nc
x_1 y_1 z_1    xt_1 yt_1 zt_1 # for 2D
x_2 y_2 z_2    xt_2 yt_2 zt_2
......
x_Nc y_Nc z_Nc    xt_Nc yt_Nc zt_Nc
```

  here 'Nnf' is the number of no-fail nodes and 'Nc' is the number of cracks followed by two points coordinates representing a line crack for 2D . 'xt', 'yt', 'zt' represent coordinates of the crack tip. For 2D, zt=0. For 3D, the crack is represented by 3 points coordinates as:

```
x_1 y_1 z_1  x_2 y_2 z_3  x_3 y_3 z_3 # for 3D
```

4 PDLSM-FEM theory

Compared to FEM, PD is computationally expensive. PDLSM-fEM is new framework for coupling PD with FEM to take advantage of both methods. For completeness and brevity in presentation, the governing equations of coupled PDLSM-FEM is outlined succinctly in this section, with more lengthy details presented in [1, 2]. The numerical implementation of PDLSM-FEM within this solver includes implicit static or quasi-static, explicit dynamic, and implicit dynamic solving.

4.1 PDLSM

In this section, the peridynamics least square minimization (PDLSM) first developed in [3] is briefly reviewed. PDLSM employs the concept of PD interactions and LSM in conjunction with the Taylor series expansion (TSE). Relevant formulas essential for understanding PDLSM can be found in [3]. As described in [3], the variation between $f(\mathbf{x}')$ and $f(\mathbf{x})$ can be approximated based on TSE in a 3 dimensional space as

$$f(\mathbf{x}') = \sum_{n=0}^{N} B^n f(\mathbf{x}) + R(N, \mathbf{x}). \tag{4}$$

in which $R(N, \mathbf{x})$ is the remainder and B^n is defined as

$$B^n = \frac{1}{n!}\left(\xi_1 \frac{\partial}{\partial x_1} + \xi_2 \frac{\partial}{\partial x_2} + \xi_3 \frac{\partial}{\partial x_3}\right)^n, \tag{5}$$

and $\mathbf{x}' = \mathbf{x} + \boldsymbol{\xi}$ represents the neighbours of point $\mathbf{x}$ within its interaction domain H_x. The weighted error E_x from the approximation in Eq. (4) within the interaction domain can be evaluated based on least square minimization as

$$E_x = \int_{H_x} \omega(|\boldsymbol{\xi}|) R^2 dV_{x'} = \int_{H_x} \omega(|\boldsymbol{\xi}|) \left[f(\mathbf{x}') - \sum_{n=0}^{N} B^n f(\mathbf{x}) \right]^2 dV_{x'}. \tag{6}$$

This error can be minimized by requiring its first variation to vanish as $\delta E_x = 0$. For 2nd-order TSE ($N = 2$), this requirement leads to the derivatives of $f(\mathbf{x})$ in a integral form as

$$\begin{aligned} &\left[\frac{\partial}{\partial x_1} \quad \frac{\partial}{\partial x_2} \quad \frac{\partial}{\partial x_3} \quad \frac{\partial^2}{\partial x_1^2} \quad \frac{\partial^2}{\partial x_2^2} \quad \frac{\partial^2}{\partial x_3^2} \quad \frac{\partial^2}{\partial x_1 x_2} \quad \frac{\partial^2}{\partial x_2 x_3} \quad \frac{\partial^2}{\partial x_3 x_1} \right]^T f(\mathbf{x}) = \\ &\int_{H_x} \omega(|\boldsymbol{\xi}|) \begin{bmatrix} \mathbf{g} \\ \mathbf{d} \end{bmatrix} \left(f(\mathbf{x}') - f(\mathbf{x}) \right) dV'. \end{aligned} \tag{7}$$

The vector $\mathbf{g} = [g_1 \quad g_2 \quad g_3]^T$ and $\mathbf{d} = [d_1 \quad d_2 \quad d_3 \quad d_4 \quad d_5 \quad d_6]^T$ are defined as

$$\begin{bmatrix} \mathbf{g} \\ \mathbf{d} \end{bmatrix} = \mathbf{A}^{-1}\hat{\boldsymbol{\xi}}, \tag{8}$$

where

$$\hat{\boldsymbol{\xi}} = [\xi_1 \quad \xi_2 \quad \xi_3 \quad \xi_1^2 \quad \xi_2^2 \quad \xi_3^2 \quad \xi_1\xi_2 \quad \xi_2\xi_3 \quad \xi_1\xi_3]^T, \tag{9}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}, \tag{10}$$

$$\mathbf{A}_{11} = \int_{H_x} \omega(|\boldsymbol{\xi}|) \begin{bmatrix} \xi_1^2 & \xi_1\xi_2 & \xi_1\xi_3 \\ \xi_1\xi_2 & \xi_2^2 & \xi_2\xi_3 \\ \xi_1\xi_3 & \xi_2\xi_3 & \xi_3^2 \end{bmatrix} dV_{x'}, \tag{11}$$

$$\mathbf{A}_{12} = \int_{H_x} \omega(|\boldsymbol{\xi}|) \begin{bmatrix} \frac{\xi_1^3}{2} & \frac{\xi_1\xi_2^2}{2} & \frac{\xi_1\xi_3^2}{2} & \xi_1^2\xi_2 & \xi_1^2\xi_3 & \xi_1\xi_2\xi_3 \\ \frac{\xi_1^2\xi_2}{2} & \frac{\xi_2^3}{2} & \frac{\xi_2\xi_3^2}{2} & \xi_1\xi_2^2 & \xi_1\xi_2\xi_3 & \xi_2^2\xi_3 \\ \frac{\xi_1^2\xi_3}{2} & \frac{\xi_2^2\xi_3}{2} & \frac{\xi_3^3}{2} & \xi_1\xi_2\xi_3 & \xi_1\xi_3^2 & \xi_2\xi_3^2 \end{bmatrix} dV_{x'}, \tag{12}$$

$$\mathbf{A}_{21} = \int_{H_x} \omega(|\boldsymbol{\xi}|) \begin{bmatrix} \xi_1^3 & \xi_1^2\xi_2 & \xi_1^2\xi_3 \\ \xi_1\xi_2^2 & \xi_2^3 & \xi_2^2\xi_3 \\ \xi_1\xi_3^2 & \xi_2\xi_3^2 & \xi_3^3 \\ \xi_1^2\xi_2 & \xi_1\xi_2^2 & \xi_1\xi_2\xi_3 \\ \xi_1\xi_2\xi_3 & \xi_2^2\xi_3 & \xi_2\xi_3^2 \\ \xi_1^2\xi_3 & \xi_1\xi_2\xi_3 & \xi_1\xi_3^2 \end{bmatrix} dV_{x'}, \tag{13}$$

$$\mathbf{A}_{22} = \int_{H_x} \omega(|\boldsymbol{\xi}|) \begin{bmatrix} \frac{\xi_1^4}{2} & \frac{\xi_1^2\xi_2^2}{2} & \frac{\xi_1^2\xi_3^2}{2} & \xi_1^3\xi_2 & \xi_1^3\xi_3 & \xi_1^2\xi_2\xi_3 \\ \frac{\xi_1^2\xi_2^2}{2} & \frac{\xi_2^4}{2} & \frac{\xi_2^2\xi_3^2}{2} & \xi_1\xi_2^3 & \xi_1\xi_2^2\xi_3 & \xi_2^3\xi_3 \\ \frac{\xi_1^2\xi_3^2}{2} & \frac{\xi_2^2\xi_3^2}{2} & \frac{\xi_3^4}{2} & \xi_1\xi_2\xi_3^2 & \xi_1\xi_3^3 & \xi_2\xi_3^3 \\ \frac{\xi_1^3\xi_2}{2} & \frac{\xi_1\xi_2^3}{2} & \frac{\xi_1\xi_2\xi_3^2}{2} & \xi_1^2\xi_2^2 & \xi_1^2\xi_2\xi_3 & \xi_1\xi_2^2\xi_3 \\ \frac{\xi_1^2\xi_2\xi_3}{2} & \frac{\xi_2^3\xi_3}{2} & \frac{\xi_2\xi_3^3}{2} & \xi_1\xi_2^2\xi_3 & \xi_1\xi_2\xi_3^2 & \xi_2^2\xi_3^2 \\ \frac{\xi_1^3\xi_3}{2} & \frac{\xi_1\xi_2^2\xi_3}{2} & \frac{\xi_1\xi_3^3}{2} & \xi_1^2\xi_2\xi_3 & \xi_1^2\xi_3^2 & \xi_1\xi_2\xi_3^2 \end{bmatrix} dV_{x'}. \tag{14}$$

The differential operator of Eq. (7) can be used to derive the equation of motion in engineering problems by replacing the $f(\mathbf{x})$ as displacement components $u_i(\mathbf{x})$ $(i = 1,\ 2,\ 3)$, as shown in [4]. In PD, the equation of motion is given by

$$\rho\ddot{\mathbf{u}} = \mathbf{L}^{pd}(\mathbf{x}, t) + \mathbf{b}(\mathbf{x}, t), \tag{15}$$

where ρ is the mass density, $\ddot{\mathbf{u}}$ is the acceleration, $\mathbf{b}$ is the external force vector, and $\mathbf{L}$ is the internal force vector. The internal force at point $\mathbf{x}$ is evaluated by the integration of bond force between material points $\mathbf{x}$ and $\mathbf{x}'$ over the interaction domain H_x of the material point $\mathbf{x}$, as shown in Fig. 5. According to the classical continuum mechanics

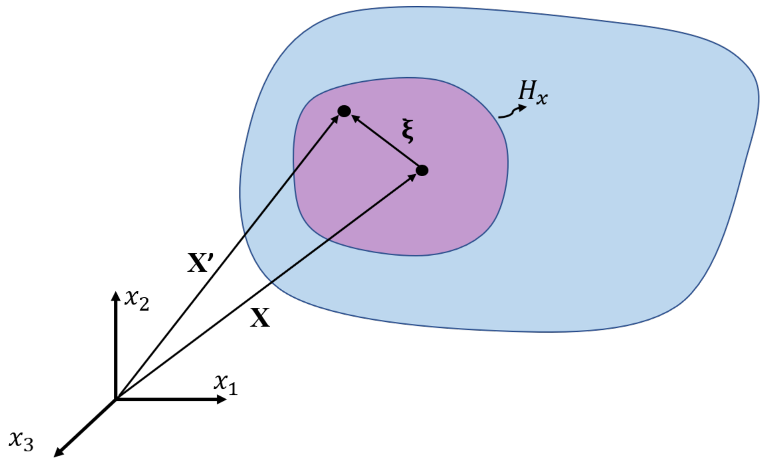


Fig. 5: Bond between points $\mathbf{x}'$ and $\mathbf{x}$ in the interaction domain H_x.

(CCM), the internal force vector can be expressed as

$$\mathbf{L}^{pd}(\mathbf{x},t) = \nabla \cdot \boldsymbol{\sigma}^{pd}(\mathbf{x},t), \tag{16}$$

in which $\boldsymbol{\sigma}^{pd}$ is Cauchy's stress. For a linear isotropic material, the stress tensor can be expressed as

$$\boldsymbol{\sigma}^{pd} = \lambda \mathbf{tr}(\nabla \mathbf{u}^{pd})\mathbf{I} + \mathcal{G}\left[\nabla \mathbf{u}^{pd} + (\nabla \mathbf{u}^{pd})^T\right], \tag{17}$$

with λ representing Lame's material constant and $\mathcal{G}$ representing shear modulus. Replacing the $f(\mathbf{x})$ of Eq. (7) by displacement components $u_i(\mathbf{x})$ leads to the non-local displacement gradient as

$$\nabla \mathbf{u}^{pd} = \int_{H_x} \omega(|\boldsymbol{\xi}|)\boldsymbol{\eta} \otimes \mathbf{g} dV_{x'}, \tag{18}$$

and the non-local internal force vector as

$$\mathbf{L}^{pd} = \int_{H_x} \omega(|\boldsymbol{\xi}|)\mathbf{G}\boldsymbol{\eta} dV_{x'}, \tag{19}$$

where $\boldsymbol{\xi} = \mathbf{x}' - \mathbf{x}$ and $\boldsymbol{\eta} = \mathbf{u}(\mathbf{x}') - \mathbf{u}(\mathbf{x})$ are the relative position and relative displacement of points $\mathbf{x}'$ and $\mathbf{x}$, respectively, and $\omega(|\boldsymbol{\xi}|)$ is the weight function. The matrix $\mathbf{G}$ in Eq. (19) is defined as

$$\mathbf{G} = \begin{bmatrix} (\lambda+\mathcal{G})d_1 + \mathcal{G}(d_1+d_2+d_3) & (\lambda+\mathcal{G})d_4 & (\lambda+\mathcal{G})d_6 \\ (\lambda+\mathcal{G})d_4 & (\lambda+\mathcal{G})d_2 + \mathcal{G}(d_1+d_2+d_3) & (\lambda+\mathcal{G})d_5 \\ (\lambda+\mathcal{G})d_6 & (\lambda+\mathcal{G})d_5 & (\lambda+\mathcal{G})d_3 + \mathcal{G}(d_1+d_2+d_3) \end{bmatrix}, \tag{20}$$

For the 2D case, the vector $\mathbf{g}$ and $\mathbf{d}$ are reduced to $\mathbf{g} = [g_1 \quad g_2]^T$and $\mathbf{d} = [d_1 \quad d_2 \quad d_3]^T$, respectively, and all the components with ξ_3 in matrix $\mathbf{A}$ and vector $\hat{\boldsymbol{\xi}}$ will be deleted.

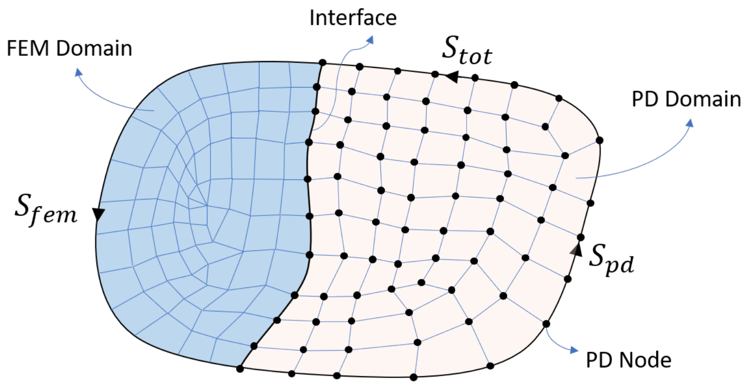


Fig. 6: 2D schematic of the coupling PD with FEM.

For plane strain problems, the matrix **G** is reduced to

$$\mathbf{G} = \begin{bmatrix} (\lambda+\mathcal{G})d_1 + \mathcal{G}(d_1+d_2) & (\lambda+\mathcal{G})d_3 \\ (\lambda+\mathcal{G})d_3 & (\lambda+\mathcal{G})d_2 + \mathcal{G}(d_1+d_2) \end{bmatrix}, \tag{21}$$

and for plane stress problems,

$$\mathbf{G} = \begin{bmatrix} \frac{E}{2(1-\nu)}d_1 + \mathcal{G}(d_1+d_2) & \frac{E}{2(1-\nu)}d_3 \\ \frac{E}{2(1-\nu)}d_3 & \frac{E}{2(1-\nu)}d_2 + \mathcal{G}(d_1+d_2) \end{bmatrix}, \tag{22}$$

here ν is Poisson's ratio, E is Young's modulus.

4.2 PDLSM-FEM

The coupled PDLSM-FEM is based on the weighted residual method (WRM) with the problem domain divided into a PD domain and a FEM domain, both of which are discretized into uniform or non-uniform elements, as illustrated in Fig. 6. To distinguish, the elements in the PD domain are called PD elements, and elements in the FEM domain are still called finite elements. Note that PD nodes on the interface of the PD domain and the FEM domain are also nodes of the finite element, and the interaction domains of these PD nodes are only determined from the PD domain, and the volumes of these PD nodes are calculated only from the PD elements. This method of coupling is straightforward and is easily introduced into commercial finite element codes. The formulation of the coupling framework can be converted conveniently between pure PDLSM, pure FEM, and coupled PDLSM-FEM. Applying the WRM on the equilibrium equations and the boundary conditions over the whole problem domain leads to

$$\int_{V_{tot}} \delta\mathbf{u}^T(\mathbf{L} + \mathbf{b} - \rho\ddot{\mathbf{u}})dV - \int_{S_{tot}} \delta\mathbf{u}^T(\boldsymbol{\sigma}\mathbf{n} - \mathbf{T})dS = 0, \tag{23}$$

where $\mathbf{T}$ is the external traction on the surface boundary, $\mathbf{n}$ is the unit normal vector of the boundary, V_{tot} and S_{tot} are the volume and surface of the whole domain respectively, and $\delta(\bullet)$ represents a virtual value. For the PD domain V_{pd}, the internal force vector takes on the integral form of Eq. (19), and for the FEM domain V_{fem}, it takes on the differential form of

$$\mathbf{L} = \nabla \cdot \boldsymbol{\sigma}. \tag{24}$$

For the surface of the PD domain, S_{pd}, the stress in the item $\delta\mathbf{u}^T\boldsymbol{\sigma}\mathbf{n}$ is derived based on the integral form of Eq. (18), and for the surface of the FEM domain, S_{fem}, $\delta\mathbf{u}^T\boldsymbol{\sigma}\mathbf{n}$ is canceled by applying Gauss theorem as

$$\int_{V_{fem}} \delta\mathbf{u}^T\mathbf{L}dV - \int_{S_{fem}} \delta\mathbf{u}^T\boldsymbol{\sigma}\mathbf{n}dS = -\int_{V_{fem}} \nabla\delta\mathbf{u} : \boldsymbol{\sigma}dV. \tag{25}$$

Therefore, Eq. (23) becomes to

$$\begin{aligned}&\int_{V_{fem}} \left[\delta\mathbf{u}^T\rho\ddot{\mathbf{u}} + \delta(\{\varepsilon^{fem}\}^T)\{\sigma^{fem}\}\right] dV + \int_{V_{pd}} \delta\mathbf{u}^T(\rho\ddot{\mathbf{u}} - \mathbf{L}^{pd})dV + \int_{S_{pd}} \delta\mathbf{u}^T\boldsymbol{\sigma}^{pd}\mathbf{n}dS = \\ &\int_{S_{tot}} \delta\mathbf{u}^T\mathbf{T}dS + \int_{V_{fem}} \delta\mathbf{u}^T\mathbf{b}dV + \int_{V_{pd}} \delta\mathbf{u}^T\mathbf{b}dV.\end{aligned} \tag{26}$$

It is shown in [1] that Eq. (26) can be transformed into the final governing equation for PDLSM-FEM as follows:

$$\mathbf{M}\ddot{\mathbf{u}}_g + \mathbf{K}\mathbf{u}_g = \mathbf{F}, \tag{27}$$

where $\mathbf{u}_g$, $\mathbf{M}$, $\mathbf{K}$ and $\mathbf{F}$ represent the global nodal displacement vector, the global equivalent nodal mass matrix, the global stiffness matrix, and the global equivalent nodal force, respectively. Details of the derivation are presented in [1].

4.3 Surface effect and volume correction

The derivation of Eq. (18) and Eq. (19) does not require the interaction domain H_x to be a sphere, as many published PD models required. In fact, the interaction domain H_x can be of arbitrary shape. Thus, the surface effect [5] can be ignored and the volume correction [6] is not required.

4.4 Boundary conditions

In many published PD theories, constraints are applied through a nonzero volume rather than on a surface, commonly introduced within a layer of thickness δ, and traction at boundaries is introduced in these PD model as body forces $\mathbf{b}$ within a layer Δx.

In PDLSM-FEM, the constraints and traction boundary conditions (BCs) are treated as same as the FEM method, as shown below.

- **Essential BCs**. The displacement set of all nodes (global displacement vector) can be expressed as

$$\mathbf{u}_g = \begin{bmatrix} \mathbf{u}_u \\ \mathbf{u}_p \end{bmatrix}, \tag{28}$$

  here $\mathbf{u}_u$ is the set of all unknown displacement components and $\mathbf{u}_p$ is the set of prescribed displacement components. Eq. (27) can be decomposed as

$$\begin{bmatrix} \mathbf{M}_{uu} & \mathbf{M}_{up} \\ \mathbf{M}_{pu} & \mathbf{M}_{pp} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{u}}_u \\ \ddot{\mathbf{u}}_p \end{bmatrix} + \begin{bmatrix} \mathbf{K}_{uu} & \mathbf{K}_{up} \\ \mathbf{K}_{pu} & \mathbf{K}_{pp} \end{bmatrix} \begin{bmatrix} \mathbf{u}_u \\ \mathbf{u}_p \end{bmatrix} = \begin{bmatrix} \mathbf{F}_u \\ \mathbf{F}_p \end{bmatrix}. \tag{29}$$

  As $\mathbf{u}_p$ is the prescribed displacement constraint, Eq. (29) reduces to

$$\mathbf{M}_{uu}\ddot{\mathbf{u}}_u + \mathbf{K}_{uu}\mathbf{u}_u = \mathbf{F}_u - \mathbf{M}_{up}\ddot{\mathbf{u}}_p - \mathbf{K}_{up}\mathbf{u}_p \tag{30}$$

  For static problem, Eq. (29) reduces to

$$\mathbf{K}_{uu}\mathbf{u}_u = \mathbf{F}_u - \mathbf{K}_{up}\mathbf{u}_p \tag{31}$$

- **Natural BCs**. PDLSM-FEM treats natural BCs as same as we do in FEM, as shown in below:

$$\int_{S_t} \mathbf{T} dS = \mathbf{F}. \tag{32}$$

  where $\mathbf{T}$ is the traction applied on surface S_t.

4.5 Solvers

- **Implicit static or quasi-static solver**. For static or quasi-static or problem, the acceleration vector $\ddot{\mathbf{u}}_g$ is negligibly small, and Eq. (27) is reduced to

$$\mathbf{K}\mathbf{u}_g = \mathbf{F}, \tag{33}$$

  which is the system equation to be implicitly solved by Parallel Direct Sparse Solver (PDSS) of Intel Math Kernel Library (MKL). Comparing to explicit dynamic relaxation method, directly and implicitly solving Eq. (33) avoids a large number of iterations and is highly computationally efficient.

- **Explicit dynamic solver**. A explicit central difference formula is used in PDLSM-FEM for explicit dynamic solver:

$$\ddot{\mathbf{u}}^n = \frac{\mathbf{u}^{n+1} - 2\mathbf{u}^n + \mathbf{u}^{n-1}}{\Delta t^2} \tag{34}$$

Note that the mass matrix from PD domain contribution, $\mathbf{M}^{pd}$, is a lumped diagonal mass matrix, while that from FEM domain contribution, $\mathbf{M}^{fem}$, can be either consistent mass matrix as

$$\mathbf{M}^{fem} = \int_V \boldsymbol{\mathcal{N}}^T \rho \boldsymbol{\mathcal{N}} dv, \tag{35}$$

or lumped diagonal mass matrix. Here $\boldsymbol{\mathcal{N}}$ is the shape function. For explicit dynamic solutions, using lumped diagonal mass matrix $\mathbf{M}^{fem}$ may be more efficient since it is easier to solve for $\mathbf{M}^{-1}$.

- **Implicit dynamic solver**. Newmark's method is the most widely used in implicit algorithm. It is first assumed that

$$\mathbf{u}_g^{n+1} = \mathbf{u}_g^n + \Delta t \dot{\mathbf{u}}_g^n + \Delta t^2 \left[\left(\frac{1}{2} - \beta \right) \ddot{\mathbf{u}}_g^n + \beta \ddot{\mathbf{u}}_g^{n+1} \right] \tag{36}$$

$$\dot{\mathbf{u}}_g^{n+1} = \dot{\mathbf{u}}_g^n + \Delta t \left[(1-\gamma) \ddot{\mathbf{u}}_g^n + \gamma \ddot{\mathbf{u}}_g^{n+1} \right] \tag{37}$$

Substituting Eqs. (37) and (37) into Eq. (27) leads to

$$\ddot{\mathbf{u}}_g^{n+1} = \mathbf{K}_N^{-1} \mathbf{F}_{res}^{n+1} \tag{38}$$

where

$$\mathbf{K}_N^{-1} = \beta \Delta t^2 \mathbf{K} + \mathbf{M} \tag{39}$$

$$\mathbf{F}_{res}^{n+1} = \mathbf{F}^{n+1} - \mathbf{K} \left\{ \mathbf{u}_g^n + \Delta t \dot{\mathbf{u}}_g^n + \Delta t^2 \left(\frac{1}{2} - \beta \right) \ddot{\mathbf{u}}_g^n \right\} \tag{40}$$

Newmark's method, like most implicit methods, is unconditionally stable if $\gamma \geq 0.5$ and $\beta \geq (2\gamma + 1)^2/16$.

4.6 Failure criteria

PDLSM-FEM solver enables 2 criteria for 2D crack propagation simulations: maximum circumferential tensile stress and maximum principal stress. More failure criteria are under development.

4.6.1 Maximum circumferential tensile stress

For a general mixed-mode simulation, PDLSM-FEM solver employs the maximum circumferential tensile stress [7] theory to determine the crack propagation direction. Based on this theory, the hoop stress reaches its maximum value on the plane of zero shear stress. For LEFM, the size of plastic zone at the crack tip is negligible and the singular term solutions of stresses at the crack tip are used to determine the crack propagation angle θ_c, which is is defined as the angle between the line of crack and the crack growth direction (Fig. 7). The value of θ_c is determined by requiring the shear stress to become zero, and is found to be

$$\theta_c = \begin{cases} 2\tan^{-1}\left(\frac{K_I}{4K_{II}} - \frac{1}{4}\sqrt{\left(\frac{K_I}{K_{II}}\right)^2 + 8}\right) & \text{for} K_{II} > 0, \\ 2\tan^{-1}\left(\frac{K_I}{4K_{II}} + \frac{1}{4}\sqrt{\left(\frac{K_I}{K_{II}}\right)^2 + 8}\right) & \text{for} K_{II} < 0, \end{cases} \tag{41}$$

For a crack to propagate, the maximum circumferential tensile stress must reach a critical value. This results in an equivalent SIF in mixed-mode condition as

$$K_{eq} = K_I \cos^3\frac{\theta_c}{2} - \frac{3}{2}K_{II}\cos\frac{\theta_c}{2}\sin\theta_c. \tag{42}$$

with θ_c determined by Eq. (41). If $K_{eq} > K_{Ic}$, in which K_{Ic} is the fracture toughness, the crack will propagate in the direction of θ_c, as shown in Fig. 7. The stress intensity factor (SIFs) are calculated based on the interaction integral, a extension of J-integral. Fig. 8 illustrates the selected integral path for evaluations of SIFs, and more details can be found in paper [1] published by the authors.

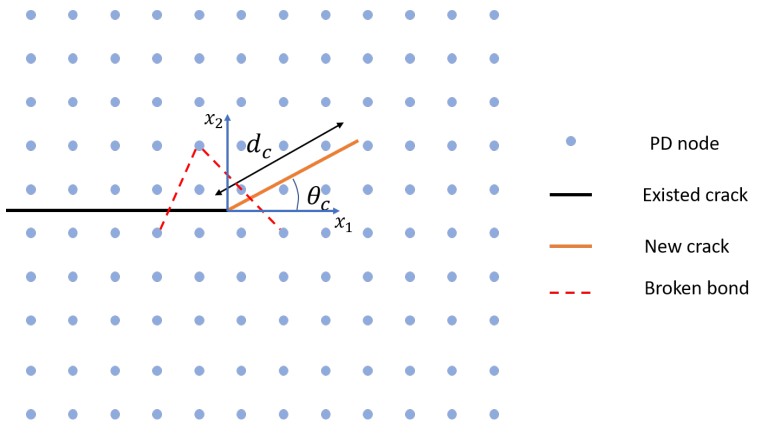


Fig. 7: Schematic of crack growth.

Most published criteria deal with only conditions (a) and (b), i.e., the onset of propagation and direction of propagation, and do not address condition (c), i.e., the amount of crack propagation. In the physical world, how much the crack will propagate depends on the body geometry and loading mechanism. If crack growth is unstable, once the propagation is triggered, K_{eq} will remain larger than K_{Ic}, and the crack will propagate in an unstable manner until ultimate failure. If crack growth is stable, once the crack starts to propagate, K_{eq} will drop below K_{Ic}, and crack growth will stop. Crack growth therefore is infinitesimal in stable growth, and a load increase is needed for further crack growth.

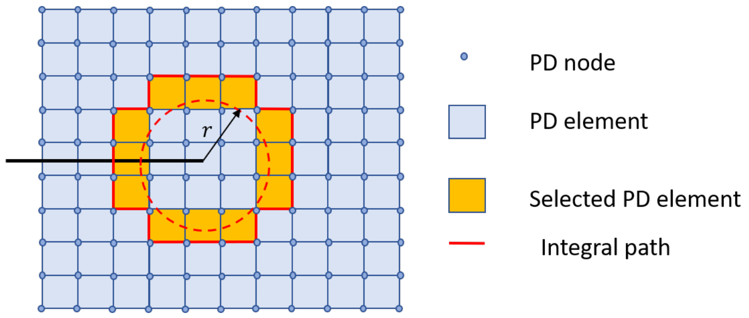


Fig. 8: Contour of the I-integral.

In numerical simulation, once the criterion for crack propagation is met, a somewhat arbitrary yet reasonable amount of crack growth must be assumed in each load step. The assumed crack growth amount is ultimately limited to the resolution of the numerical model. Since the fine PD elements near the crack tip represent the limit of the model resolution, we propose that the crack propagation amount should be comparable to the maximum characteristic length $\Delta_{max} = \sqrt{A_{max}}$ of PD nodes (A_{max} represents maximum PD node's area), for example α times the Δ_{max}, with α in the range of 0.5 to 1.5.

In the simulation, we implement the technique to model crack growth as following. We pre-define the amount of crack growth d_c based on the PD elements size in each load step. Whenever the condition $K_{eq} > K_{Ic}$ is reached, the crack will propagate an amount d_c along the direction of θ_c. All bonds crossing the path of d_c will be treated as broken with their bond status parameter μ updated from 1 to 0.

The value of the integral radius $r = m_r\Delta_{(n)}$ and crack grow amount $d_c = \alpha\Delta_{max}$ can be specified by the commands:

```
# syntax: FC 1 m_r alpha. Example:
FC 1 6.0 1.0
```

4.6.2 Maximum principal stress

The maximum principal stress criterion is modified from that of [8] and is similar to the *scheme C* of paper [9]. The maximum principal stress of bond $\boldsymbol{\xi}_{(kj)}$, $\boldsymbol{\sigma}^{max}_{(kj)}$, is defined as

$$\sigma^{max}_{(kj)} = \max\{\sigma^{pd}_{1(kj)},\ \sigma^{pd}_{2(kj)},\ \sigma^{pd}_{3(kj)}\}, \tag{43}$$

in which $\sigma^{pd}_{1(kj)}$, $\sigma^{pd}_{2(kj)}$, and $\sigma^{pd}_{3(kj)}$ are the principal stresses corresponding to the average stress of nodes $\mathbf{x}_{(k)}$ and $\mathbf{x}_{(j)}$ defined as

$$\boldsymbol{\sigma}^{pd}_{(kj)} = \left(\boldsymbol{\sigma}^{pd}_{(k)} + \boldsymbol{\sigma}^{pd}_{(j)}\right)/2 \tag{44}$$

When $\sigma_{(kj)}^{max}$ reaches the uniaxial tensile strength of the material, σ_{ult}, a virtual crack with length Δ_{max} is defined as

$$\left[\mathbf{c}_{(kj)} + \frac{\Delta_{max}}{2}\mathbf{v}_{(kj)}, \quad \mathbf{c}_{(kj)} - \frac{\Delta_{max}}{2}\mathbf{v}_{(kj)}\right], \tag{45}$$

where Δ_{max} is the largest characteristic length among all PD nodes, $\mathbf{c}_{(kj)} = \left(\mathbf{x}_{(k)} + \mathbf{x}_{(j)}\right)/2$ is the center of the bond $\boldsymbol{\xi}_{(kj)}$, and the vector $\mathbf{v}_{(kj)}$ is the unit vector which is normal to the direction of the maximum principle stress $\sigma_{(kj)}^{max}$. Then all bonds crossing the virtual crack are broken.

If there are more than k_{top} bonds with $\sigma_{(kj)}^{max} > \sigma_{ult}$, number of k_{top} virtual cracks are formed based on these bonds with the largest maximum principle stress. If there are bonds with $\sigma_{(kj)}^{max} > \sigma_{ult}$ but the number is no more than k_{top}, virtual cracks are formed based on all these bonds. All bonds crossing the virtual cracks are broken and the stiffness matrix is updated due to the bonds breakage. The command to set the maximum principle stress criterion is:

```
#syntax: FC 2 TopK. TopK=0 means no limitation. Example:
FC 2 4
```

References

[1] Q. Liu, X. Xin, J. Ma, Y. Wang, Simulating quasi-static crack propagation by coupled peridynamics least square minimization with finite element method, Engineering Fracture Mechanics 252 (2021) 107862. doi:10.1016/j.engfracmech.2021.107862.

[2] Q. Liu, X. J. Xin, J. Ma, Coupled peridynamics least square minimization with finite element method in 3d and implicit solutions by message passing interface, Journal of Peridynamics and Nonlocal Modeling (2021). doi:10.1007/s42102-021-00060-3.

[3] E. Madenci, M. Dorduncu, X. Gu, Peridynamic least squares minimization, Computer Methods in Applied Mechanics and Engineering 348 (2019) 846–874. doi:10.1016/j.cma.2019.01.032.

[4] E. Madenci, M. Dorduncu, A. Barut, N. Phan, Weak form of peridynamics for nonlocal essential and natural boundary conditions, Computer Methods in Applied Mechanics and Engineering 337 (2018) 598–631. doi:10.1016/j.cma.2018.03.038.

[5] Q. V. Le, F. Bobaru, Surface corrections for peridynamic models in elasticity and fracture, Computational Mechanics 61 (4) (2018) 499–518. doi:10.1007/s00466-017-1469-1.

[6] P. Seleson, Improved one-point quadrature algorithms for two-dimensional peridynamic models based on analytical calculations, Computer Methods in Applied Mechanics and Engineering 282 (2014) 184–217. doi:10.1016/j.cma.2014.06.016.

[7] F. Erdogan, G. Sih, On the crack extension in plates under plane loading and transverse shear (1963).

[8] E. Madenci, M. Dorduncu, A. Barut, N. Phan, A state-based peridynamic analysis in a finite element framework, Engineering Fracture Mechanics 195 (2018) 104–128. doi:10.1016/j.engfracmech.2018.03.033.

[9] T. Ni, M. Zaccariotto, Q.-Z. Zhu, U. Galvanetto, Static solution of crack propagation problems in Peridynamics, Computer Methods in Applied Mechanics and Engineering 346 (2019) 126–151. doi:10.1016/j.cma.2018.11.028.