GPU-based DEM code instruction

Dr. Jieqing Gan

Discrete Element Method (DEM)

Discrete Element Method (DEM)

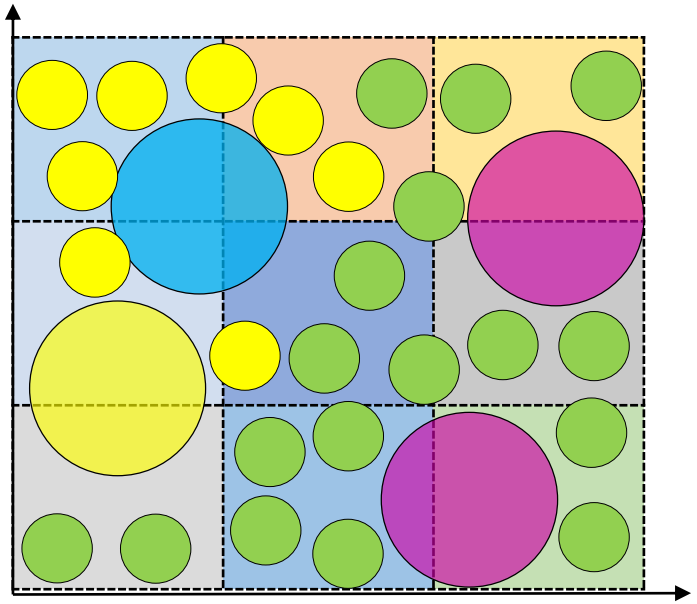
Governing equations: The normal and tangential collisional forces

$$m_i \frac{d\mathbf{v}_i}{dt} = \sum_{j=1}^{k_c} (\mathbf{f}_{c,ij} + \mathbf{f}_{d,ij}) + m_i \mathbf{g}$$

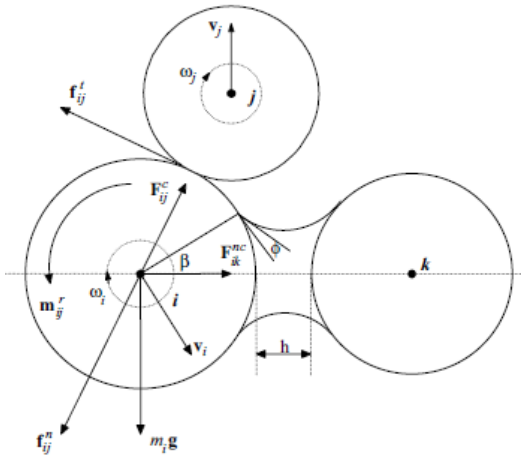
$$I_i \frac{d\boldsymbol{\omega}_i}{dt} = \sum_{j=1}^{k_c} (\mathbf{M}_{t,ij} + \mathbf{M}_{r,ij} + \mathbf{M}_{n,ij})$$

Table 1. Components of forces and torque acting on particle *i*

Forces and torques	Symbols	Equations
Normal elastic force	$\mathbf{f}_{cn,ij}$	$-\frac{4}{3} E^* \sqrt{R^*} \boldsymbol{\delta}_n^{3/2} \mathbf{n}$
Normal damping force	$\mathbf{f}_{dn,ij}$	$-c_n (8 m_{ij} E^* \sqrt{R^* \boldsymbol{\delta}_n})^{1/2} \mathbf{v}_{n,ij}$
Tangential elastic force	$\mathbf{f}_{ct,ij}$	$-\mu_s \lvert\mathbf{f}_{cn,ij}\rvert (1-(1-\boldsymbol{\delta}_t / \boldsymbol{\delta}_{t,\max})^{3/2}) \boldsymbol{\delta}_t \quad (\boldsymbol{\delta}_t < \boldsymbol{\delta}_{t,\max})$
Tangential damping force	$\mathbf{f}_{dt,ij}$	$-c_t (6\mu_s m_{ij} \lvert\mathbf{f}_{cn,ij}\rvert \sqrt{1-\lvert V_t\rvert / \boldsymbol{\delta}_{t,\max}} / \boldsymbol{\delta}_{t,\max})^{1/2} \mathbf{v}_{t,ij} \quad (\boldsymbol{\delta}_t < \boldsymbol{\delta}_{t,\max})$
Coulumb friction force	$\mathbf{f}_{t,ij}$	$-\mu_s \lvert\mathbf{f}_{cn,ij}\rvert \boldsymbol{\delta}_t \quad (\boldsymbol{\delta}_t \geq \boldsymbol{\delta}_{t,\max})$
Torque by normal force	$\mathbf{M}_{n,ij}$	$\mathbf{R}_{ij} \times (\mathbf{f}_{cn,ij} + \mathbf{f}_{dn,ij})$
Torque by tangential force	$\mathbf{M}_{t,ij}$	$\mathbf{R}_{ij} \times (\mathbf{f}_{ct,ij} + \mathbf{f}_{dt,ij})$
Rolling friction torque	$\mathbf{M}_{r,ij}$	$\mu_{r,ij} R_i \lvert\mathbf{f}_{n,ij}\rvert \hat{\boldsymbol{\omega}}_{t,ij}^n$



Computational domain



Forces acting on particle *i*

CPU-DEM and GPU-DEM

CPU-based DEM

1) **Number of particles** < 300,000,

computation time: Days to months.

2) **Complex particulate systems**:

Complex geometries

Complex wall movements

Particle shapes

Particle size distribution

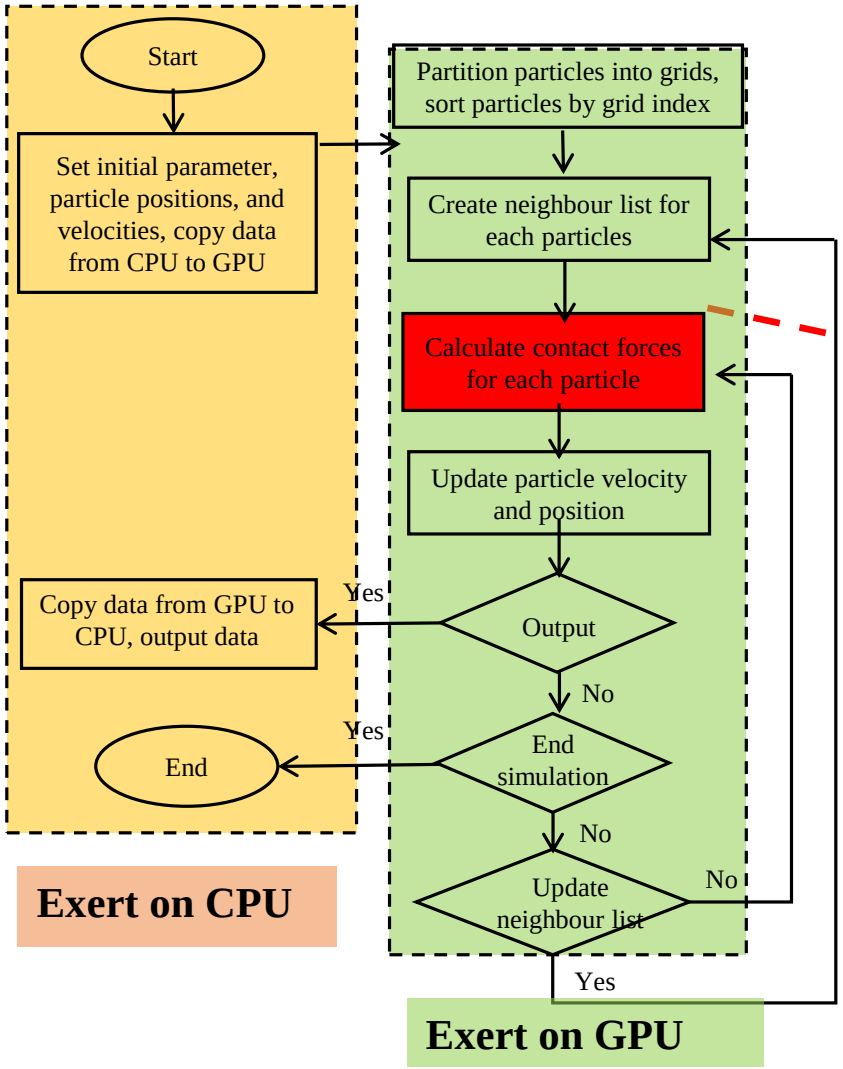
GPU-based DEM:

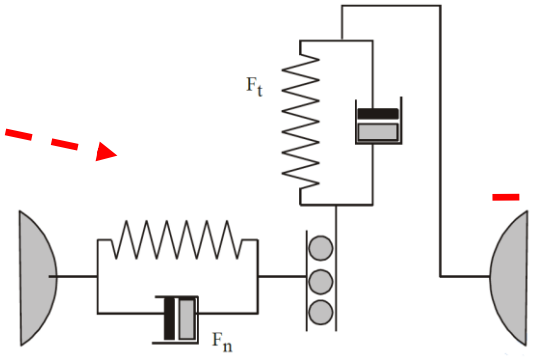
- Deal with billions of particles with high efficiency
- Complex system with funcitons:

arbitrary wall geometry, non-spherical particles, wide size distribution, CFD-GPU-DEM

GPU-DEM

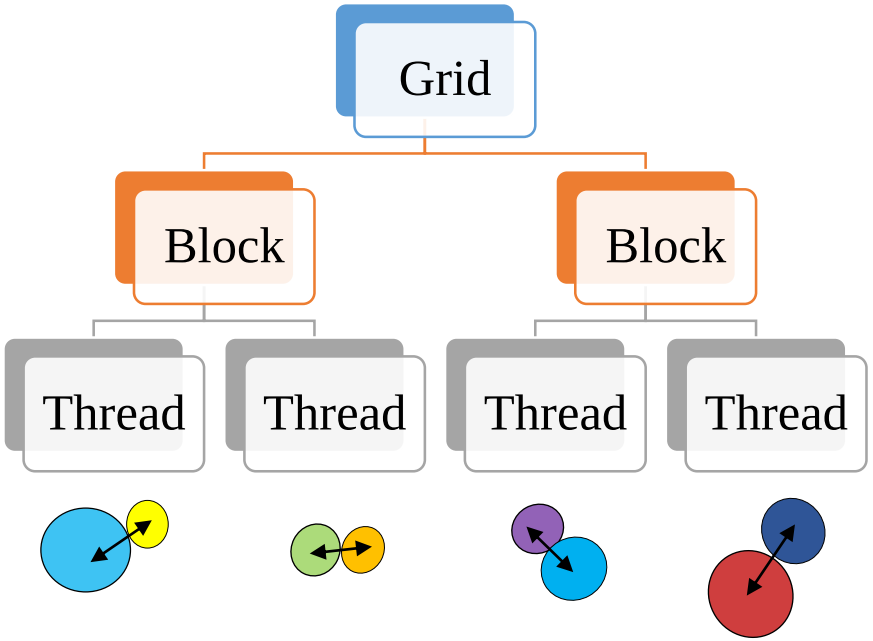
GPU-DEM flow chart





Normal and tangential forces

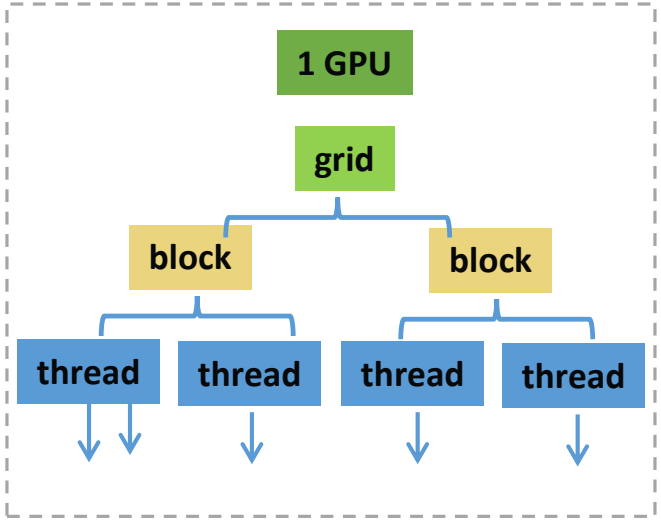
Graphic Processing Unit (GPU) thread hierarchy:



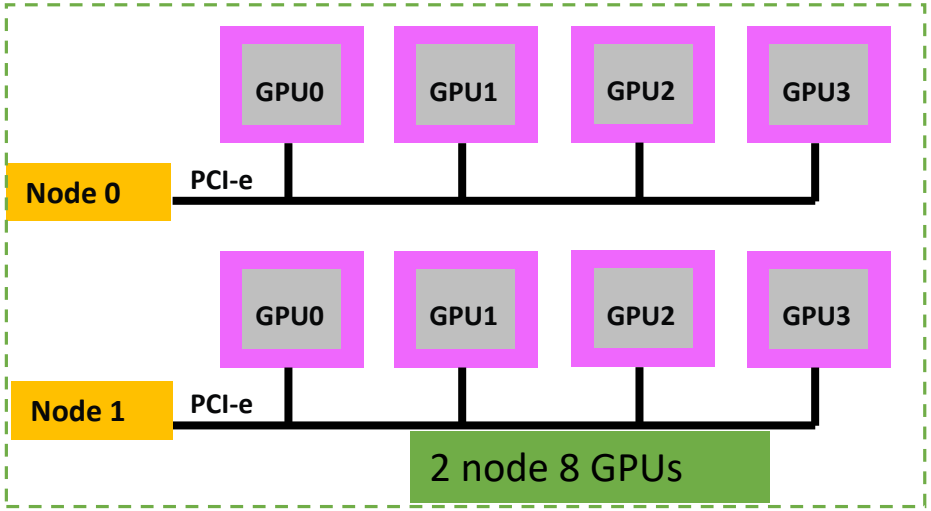
Monash Massive3 V100 GPU cluster:

```
Max threads per block:  1024
Max thread dimensions xyz: (2048, 2048, 64)
Max grid dimensions xyz: (65536, 65536, 65536)
Total threads:
1024*2048*2048*64*65536*65536*65536
```

GPU Thread hierarchy:



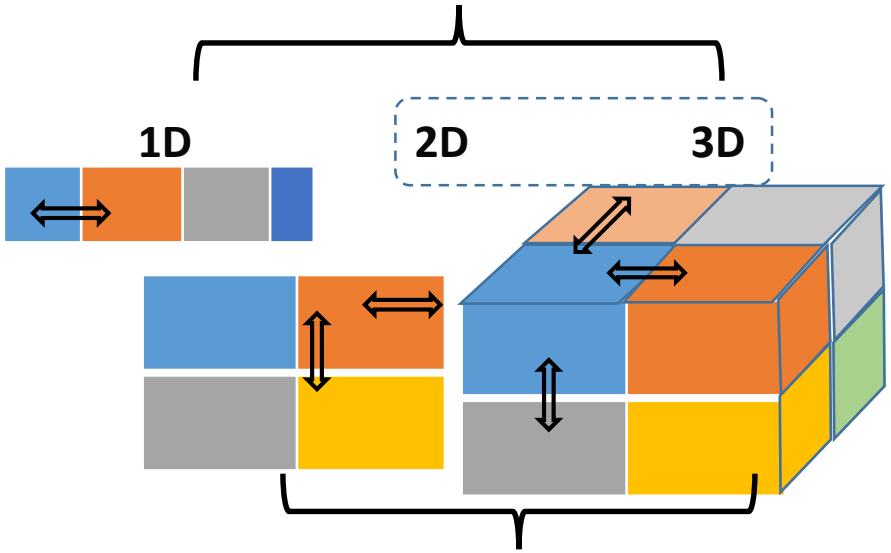
Multiple GPUs:



MPI-GPU-DEM (not applicable to current code)

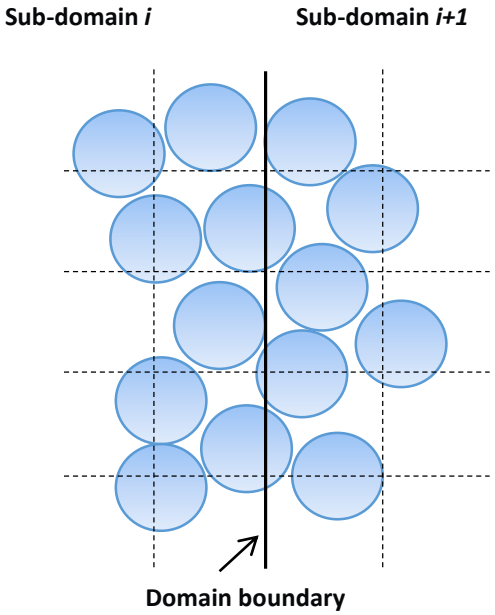
Computation domain

How to divide domain?



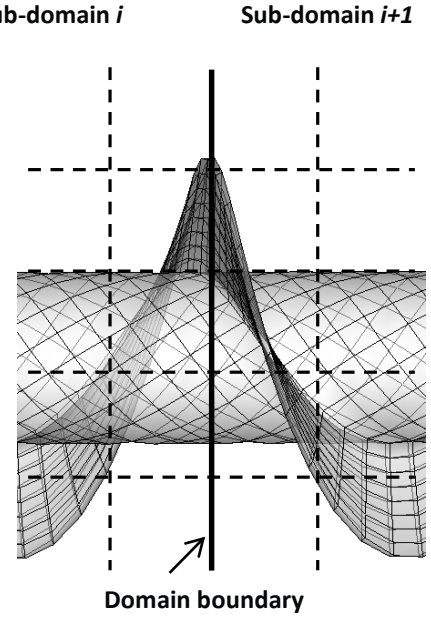
1D division ← **More time for data communication (note: this is performed on CPU)**

Particles at the domain boundary



Send/receive particle information: Position, velocity, temperature, etc. to/from Neighbour domain

Wall surfaces at the domain boundary

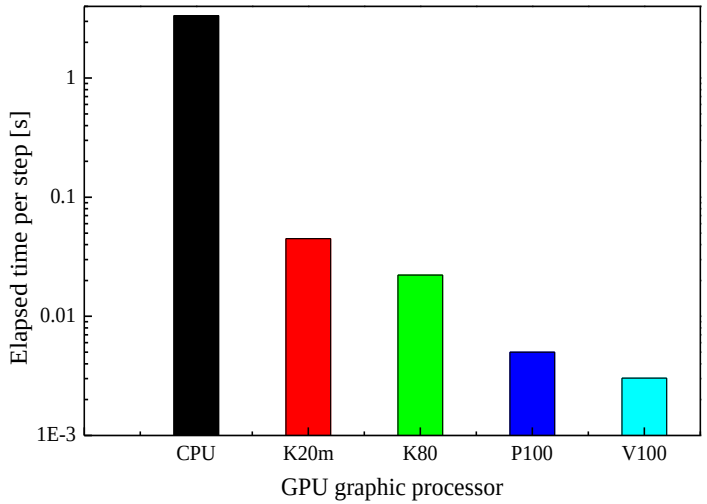


Send/receive wall information: vertex, surface

GPU - DEM

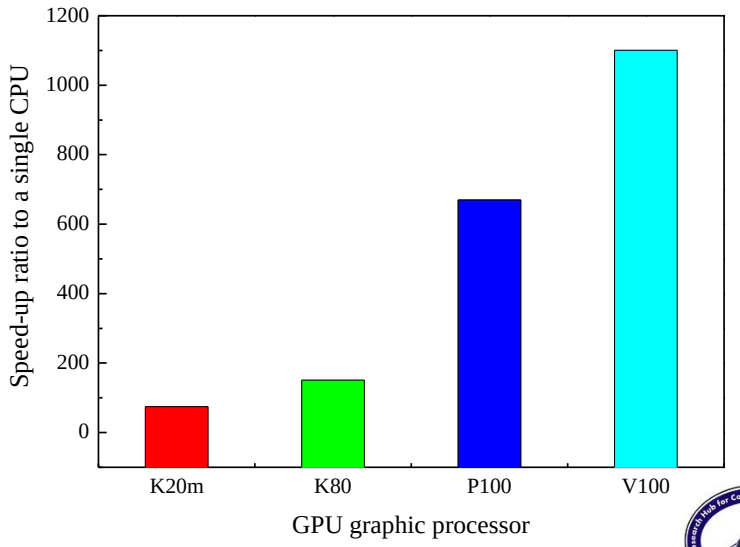
Specification of different GPU graphic processors

Model name	Tesla K20m (UNSW)	Tesla K80 (Massive 3)	Tesla P100 (Massive 3)	Tesla V100 (Massive 3)
Memory clock, GHz	2.6	5	715	876
Maximum band width, GB/s	208	480	732.2	897.0
Peak performance of double precision, TFlops	1.17	2.91	4.763	7.066

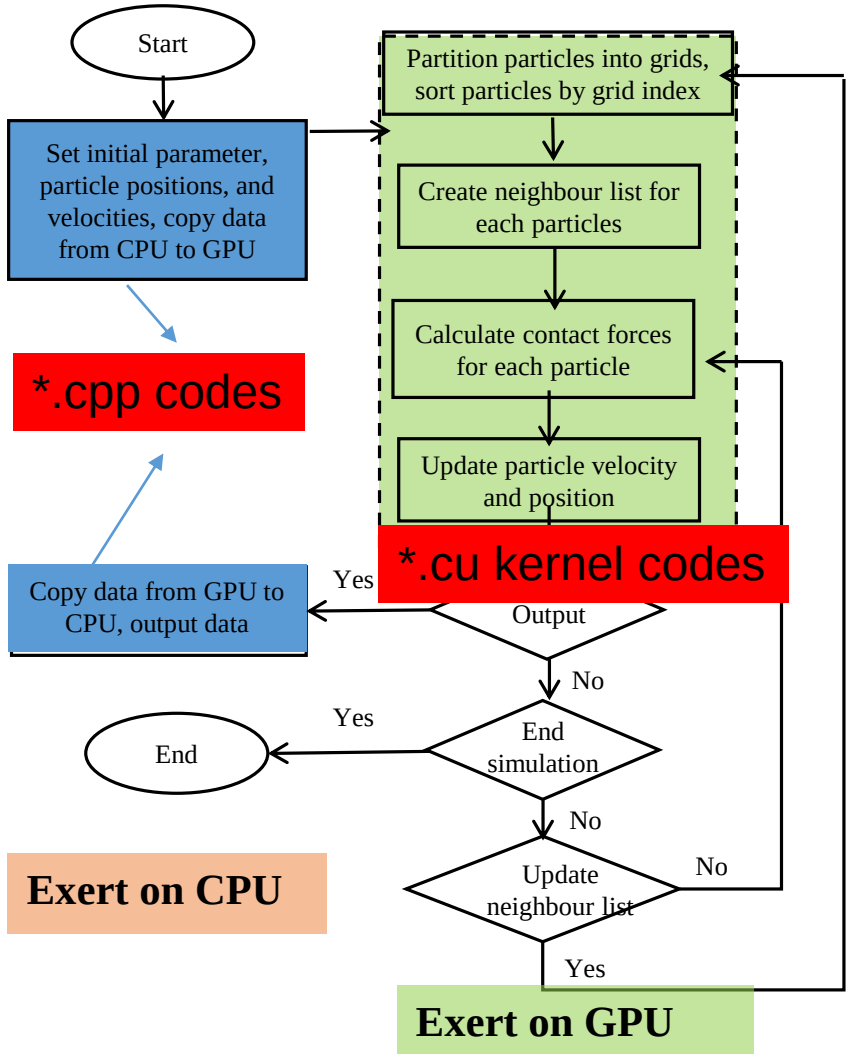


For the **homogenous** case of packing in a rectangular box with 300,000 spheres :

Speed-up ratio of V100 to a single CPU >1000 times



GDEM code implementation



Three layers:

1) *.cpp codes: e.g., **dempacking.cpp (call other codes inside it)**
   run on CPU for case setting:
   read input data,
   set initial variables,
   set material/feed stream properties.
   set boundary conditions (meshes).
   set operation conditions (mesh movements: translation, rotation and vibration)
   write data to output files
   allocate/copy data between CPU and GPU/ free data arrays
2) *.cu codes: e.g., **dempacking.cu**
   call *.cu kernel codes.
3) *.cu kernel codes: e.g. **neiglist.cu**
   run on GPU devices. __device__ , __global__
   to run key DEM steps,
   parallelized by particles or grids or contact pairs: e.g., each thread calculate the neighbor list of a particle, each thread calculate the contact between a contact pair

dempacking.cpp

1) Initialization DEM

```
// GPU start
     InitialCUDA();

     AllocateGPUArray();
     AllocateCollisionArray();
     CopyConstantsToCUDA();

//--- treat mesh data, once for all-------------------
---
     TreatMesh();

     ReadRestartData();

     if(nrestart==0)
     {
     ReadParticleData();

     AllocateGPUParticleArrays();
     CopyDataHostToDevice();
 m_hiall_total=1;
      }
```

Global GPU arrays allocation

Restart simulation from previous existing data

input

CPU array allocation /initialization

```
int main(int argc, char* argv[])
{
   MPI_Init(&argc,&argv);
   StartupMPI();
   readinputparameters();
//---------------------
   readmaterialproperties(); //get diam
   SetMatMaxRad();

   SetConstants();
   readmeshinfo();
   SetGeometryBoundary();

   readfeedinfo();
   SetFeedArrays();
//---------------------
   readmovementinfo();
//-------------------------------------------------
      SetParNumInDomain(&ntot,&m_maxIlistntot);
      AllocateCPUParticleArray(ntot);
      AllocateCPUNeigListArray(m_maxIlistntot);

       InitialCPUParticleArray(ntot);
       InitialCPUNeigListArray(m_maxIlistntot);

       AllocateMaterialArray();
       InitialMaterialArray();
//-----------------------------------------
      InitialGlobalVar();
  allocatebuf=ntot;
  m_maxIlistbuf=allocatebuf*10;
  AllocateMPIParticleBuffer(allocatebuf); //newly added
```

dempacking.cpp

2) Run DEM iteration

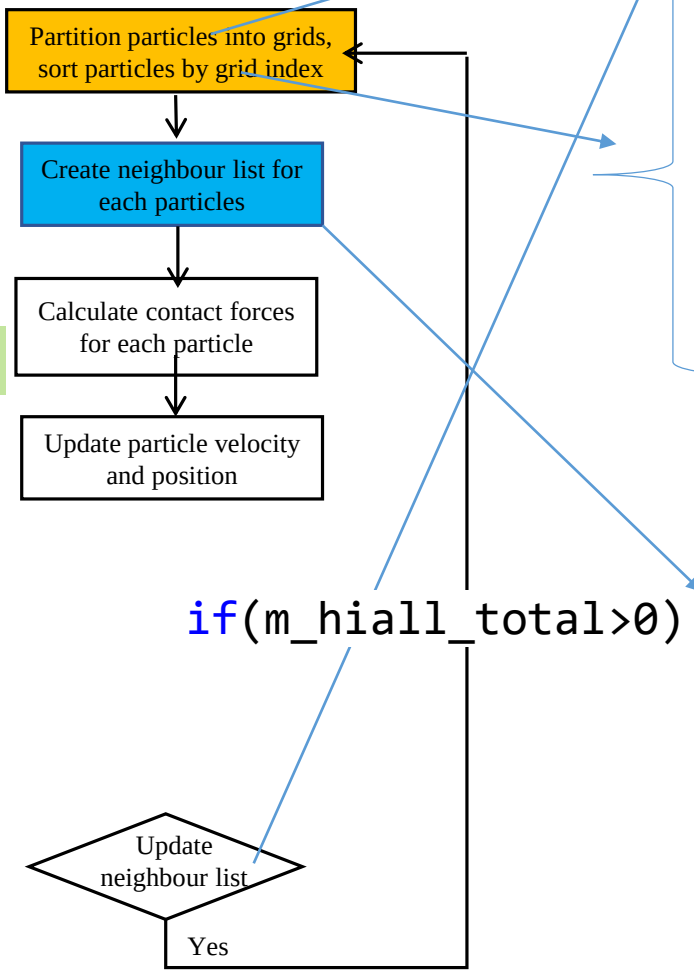
2-1): for each time step it, check if we need to add particles into domain,

if yes, add, else continue other steps.

```
while( it< tstop)
 {
     it=it+1;
     timet=dt*it;
     tsd=(it)*real_dt;

// poured packing
     FeedId=IsTimeToFeed();

     if(FeedId<NumOfFeed)
 {
 // add particles to system
        itime++;

       if(it==1|| itime>tnewpar)
 {
         itime=0;
         m_oldnumParticles=m_numParticles;
```

dempacking.cpp

- 2-2) key DEM steps 1-2



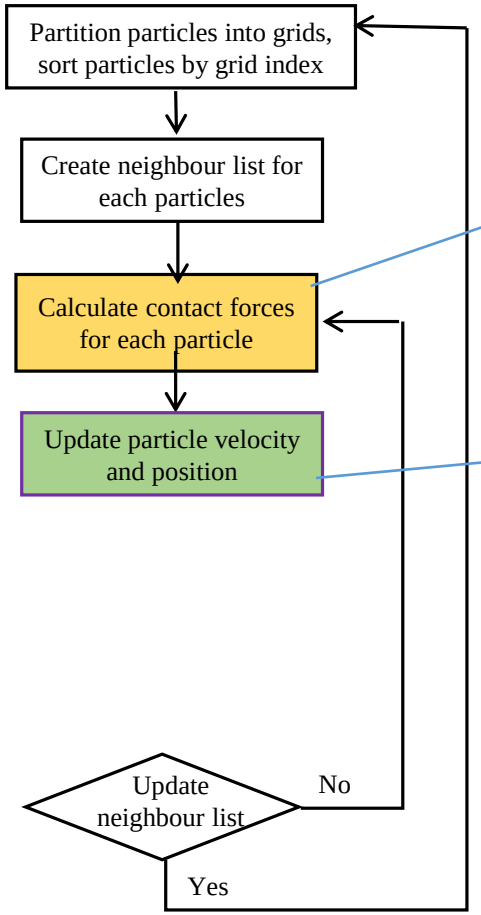
Exert on GPU

```
if(m_hiall_total>0)
```

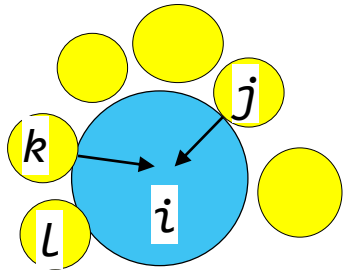
```
 if(m_hiall_total>0)
   {
// calculate hash
    calcHash();

// sort particles based on hash-----------------------------
    RadixSort();
    findBoundryNum( ); //check if particles are in the boundary or halo
or inside

TreatBoundHalo();

ReOrderParticleArrays(); //if out side, remove it, and reset particle
index for others.

mpiDataSendRecv();

calculateCN(); //calculate how many particles around particle I
according particle size
prefixsumCN( ); //calculate to CN and start of totalCN for i

findBCellStart( ); //to getm_dCellStart, m_dCellStartB

neighborarray(); // to get m_dAnei, m_dnjgi, m_dnjli
prefixsum( );

pairlist();
// updated neighbour list
m_hiall=0;
if(idelete==1)idelete=0;
}
```

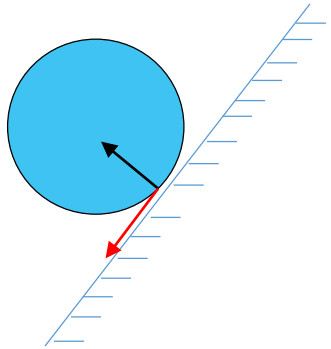
dempacking.cpp

- 2-3) key DEM steps 3-4




```
calculateforcepair(); //i-j, i-k, i-l

// sum all the forces on particle i
sumallforcei();

// calculate particle-wall forces -----
----------------
calculateforcepw();

updateposition();
```





dempacking.cpp

- 2-4) output data for post-processing

Steps:
1) Set Output frequency (set in input data), important, will affect data file size
2) List variables to output, copy data from GPU to CPU
3) Output from to data file according to visualization data file format.

```
if(it==1 || (it%freqppor)==0 && m_numParticles>=0)
{
  WriteDataToPpor();
 if(m_hminhz_total<hmax-2.0 && hminflag==1)
  WriteOutEngdsp();
}
//-------------------------------------------------
-----------------
   if(it==1 || it%freqparticle==0) //freqoutput
         {
       WriteParticleToTecplot();
   //exit(1);
         }
```

dempacking.cpp

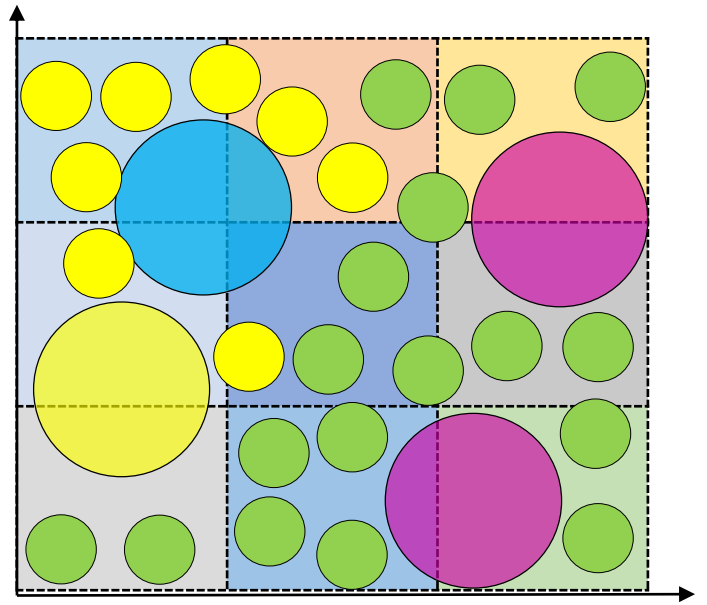
- 2-4) check if need to end simulation, or if reach CPU wall time,
- If need, free CPU/GPU arrays

```
if(it>=tstop)
   {
        FreeGPUData();

        FreeCPUData();

        exit(1);
       }

} // while, end of iteration
```

Feed.cpp

Materials with a size distribution is add in 3 size groups:

- m_NumAddH
- m_NumAddM
- m_NumAddL



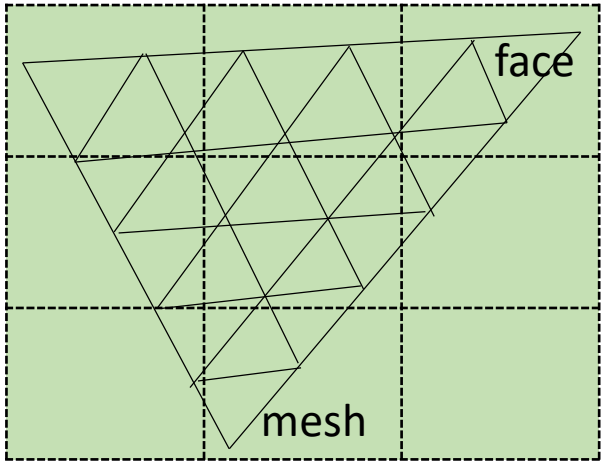
First check if a feed grid is occupied with particles, if no, add

Limitation: can not generate dense particle flow.

Boundary.cpp

- Set computation domain size and DEM grids
- Treat wall mesh:

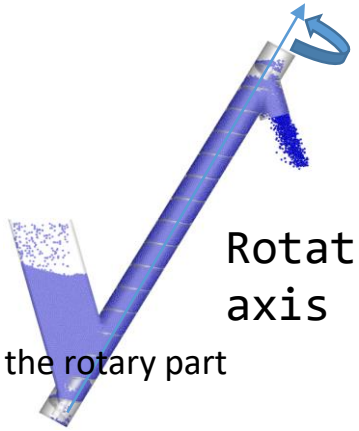
  to generate a list of possible contact wall faces of a DEM grid



Computation domain

Wall Movement: CPU level

mesh



Movement.cpp

- IsTimeToTranslateHost(),
- IsTimeToRotateHost(),
- vibration():

to check if is time to move wall

`RotationPositionConvertHost()`

dempacking.cuh:

```
struct Rotation
{
uint meshId;
double starttime;
double endtime;
double3 axis;
double3 pointonaxis;
double rotatespeed;
};
```

WriteData.cpp

```
for(j=0;j<Nnode[i];j++)
    {
 // treat particle position when they are near or inside the rotary part
   px=m_hPosnode[i][j*3];
   py=m_hPosnode[i][j*3+1];
   pz=m_hPosnode[i][j*3+2];

     int RotateId=IsTimeToRotateHost(it, real_dt,NumOfRotation);
     if(RotateId<NumOfRotation &&  m_hRotation[RotateId].meshId==i)
     {
     double px1,py1,pz1;
     double rotaterate =-m_hRotation[RotateId].rotatespeed*2*pi/60*real_dt/dt;
//reduced rotate rate
     double theta=rotaterate*it*dt;
     RotationPositionConvertHost(px,py,pz,m_hRotation[RotateId].axis.x,\
m_hRotation[RotateId].axis.y, m_hRotation[RotateId].axis.z,theta,&px1,&py1,&pz1);
fprintf(fp1,"%10.4f%10.4f%10.4f%2d%2d%2d%2d%2d%2d\n",px1*diam,py1*diam,pz1*d
iam,0,0,0,0,0,0);
     }
     else
   fprintf(fp1,"%10.4f%10.4f%10.4f%2d%2d%2d%2d%2d%2d\n",
           px*diam,py*diam,pz*diam,0,0,0,0,0,0);
    }
```

dempacking.cu

```
void calcHash(uint prehead,double*  Pos, double* rad,
              uint*   particleHash,
              double   worldOrigin[3],
              uint    gridSize[3],
              double cellSize[3],
              uint    gridSizel[3],
              double cellSizel[3],
              int     numBodies)
{
    int numThreads = min(256, numBodies);
    int numBlocks = (int) ceil(numBodies / (double) numThreads);

    // execute the kernel
    calcParticleHashD<<< numBlocks, numThreads >>>(prehead,
                                          numBodies,
                                          (double3 *) Pos,rad,
                                          (uint2 *) particleHash,
                                          make_double3(worldOrigin[0],worldOrigin[1],worldOrigin[2]),
                                          make_uint3(gridSize[0], gridSize[1], gridSize[2]),
                                          make_double3(cellSize[0], cellSize[1], cellSize[2]),
                                          make_uint3(gridSizel[0], gridSizel[1], gridSizel[2]),
                                          make_double3(cellSizel[0], cellSizel[1], cellSizel[2]));
}
```

Parallelized by:
numBodies (particles, contact pairs, number of DEM grids)

*.cu kernel code function

neiglist.cu

- **__device__** function:

  to realize a specific function

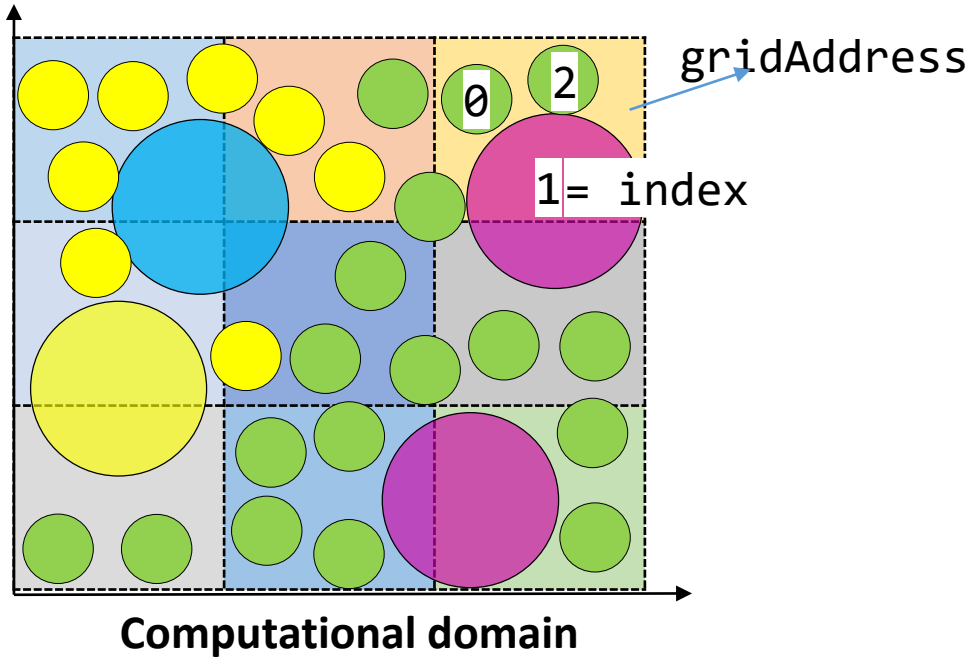
```
// calculate position in uniform grid
__device__ int3 calcGridPos(double3 p,
                  double3 worldOrigin,
                  double3 cellSize
                  )
{
   int3 gridPos;

if(GDiv[0].x==1)
{
  // gridPos.x = floor((p.x - worldOrigin.x) / cellSize.x);
   gridPos.x = floor((p.x - worldOrigin.x) / cellSize.x)+1;  //divide domain in x direction
   gridPos.y = floor((p.y - worldOrigin.y) / cellSize.y);
   gridPos.z = floor((p.z - worldOrigin.z) / cellSize.z);
   }
else if(GDiv[0].z==1)
{
gridPos.x = floor((p.x - worldOrigin.x) / cellSize.x);
   gridPos.y = floor((p.y - worldOrigin.y) / cellSize.y);
   gridPos.z = floor((p.z - worldOrigin.z) / cellSize.z)+1;  //local grid position, divide domain in z
direction
}

   return gridPos;
}
```

- **__global__** function

```
// calculate grid hash value for each particle
__global__ void calcParticleHashD(uint prehead,uint numBodies,
          double3* pos,
          double* rad,
          uint2*  particleHash,
            double3 worldOrigin,
          uint3   gridSize,
          double3  cellSize,
          uint3   gridSizel,
          double3  cellSizel)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x; //check parallelized by
particles, grids, or pairs
    uint index=i+prehead;

    if(i<numBodies)
   {
    double3 posi = pos[index];
    int3 gridPos;
    uint gridAddress;

    if(isnan(posi.x))
    {
    gridAddress = 0xffffffff;
    }

    // get address in grid
    if(outlimit(posi)==1) // outside boundary limit
    {
     gridAddress=0xffffffff;
     }
    else
    {
    gridPos = calcGridPos(posi, worldOrigin,cellSizel);
    gridAddress = calcGridAddress(gridPos, gridSizel);
    }

   // store grid hash and particle index
    particleHash[i] = make_uint2(gridAddress, index);
}
}
```

neiglist.cu

- **calcParticleHashD()**



Similar functions:

- calcParticleInFeedAreaHD()
- calcParticleInFeedAreaLD()

```
// calculate grid hash value for each particle
__global__ void calcParticleHashD(uint prehead,uint numBodies,
          double3* pos,
          double* rad,
          uint2*  particleHash,
            double3 worldOrigin,
          uint3   gridSize,
          double3  cellSize,
          uint3   gridSizel,
          double3  cellSizel)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x; //check parallelized by
particles, grids, or pairs
    uint index=i+prehead;

    if(i<numBodies)
   {
    double3 posi = pos[index];
    int3 gridPos;
    uint gridAddress;

    if(isnan(posi.x))
    {
    gridAddress = 0xffffffff;
    }

    // get address in grid
    if(outlimit(posi)==1) // outside boundary limit
    {
     gridAddress=0xffffffff;
     }
    else
    {
    gridPos = calcGridPos(posi, worldOrigin,cellSizel);
    gridAddress = calcGridAddress(gridPos, gridSizel);
    }

   // store grid hash and particle index
    particleHash[i] = make_uint2(gridAddress, index);
}
}
```

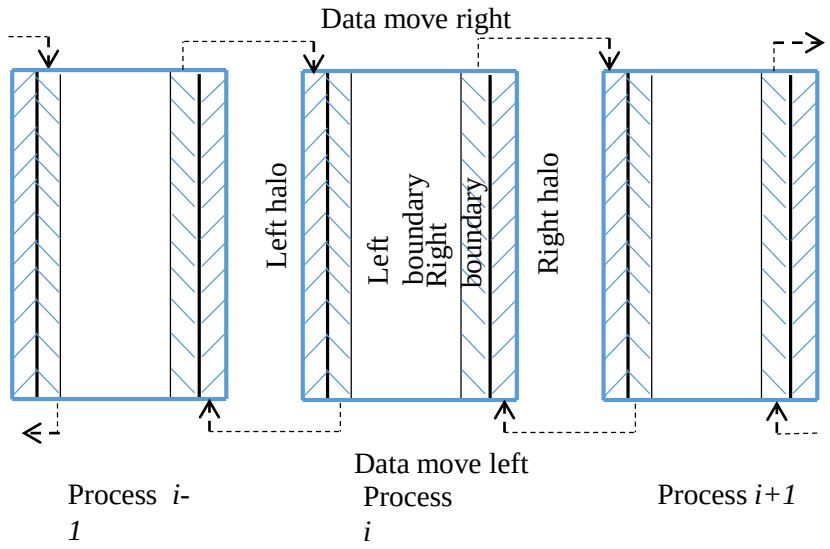
neiglist.cu

- **RandomAddParticlesHD()**
- For each grid in feed area, check if(NumParticleInHCell[index]==0),no particles in the cell, then add particle position and matId

```
// calculate grid hash value for each particle
__global__ void RandomAddParticlesHD(
          uint oldNumParticle,
          uint *NrandomH,
          uint Nallocate,
          uint Numgrids,
          uint3   gridSize,
          double* rad,
          double radHmax,
           uint FeedId,
          uint*  NumParticleInHCell,
          double3* pos,
           uint* matId,
          uint* Addcount)
{
    // thread number= number of grids to generate in feed area
    int id= blockIdx.x*blockDim.x + threadIdx.x;  //local number
    double3 p;
    int3 gridPos;

    if(id<Numgrids)
    {
    int index=NrandomH[id];
    if(index<Numgrids)
    {
    if(NumParticleInHCell[index]==0) //no particles in cell
    {
    uint icount=atomicAdd(&Addcount[0],1);

    if(icount<Nallocate)
    {
if(GDiv[0].x==1)
{
gridPos.x=(int) index/(gridSize.y*gridSize.z);
    gridPos.z=(int) (index-gridPos.x*gridSize.y*gridSize.z)/gridSize.y;
    gridPos.y=index-gridPos.x*gridSize.y*gridSize.z - gridPos.z*gridSize.y;
}

    double3 cellSize=2.05*radHmax*make_double3(1.0);

    p.x=(gridPos.x+0.5)*cellSize.x+m_dfeedlim[FeedId].xmin;
    p.y=(gridPos.y+0.5)*cellSize.y+m_dfeedlim[FeedId].ymin;
    p.z=(gridPos.z+0.5)*cellSize.z+m_dfeedlim[FeedId].zmin;

    pos[oldNumParticle+icount]=p;
    matId[oldNumParticle+icount]= m_dfeedlim[FeedId].matId;

    } // if(icount<Nallocate)
    } //if(NumParticleInHCell[index]==0)
  }
  }
}
```

neiglist.cu

MPI




```
// find the number of left and right boundry data Nlb and Nrb, and
total number of particle inside the domain
__global__ void findBoundryNumD(uint prehead,
                uint2*  particleHash, //for inner particles, sorted
                uint    numBodies,
                uint    NgridRbs, //index not number
                uint    NgridRb, uint   NgridLb,
                uint    NgridRh, uint   NgridLh,
                uint   *Pidrbs,//return
                uint   *Pidlb,
                uint   *Pidrb,
                uint   *Pidlh,
                uint   *Pidrh,
                uint*  Nin)
```

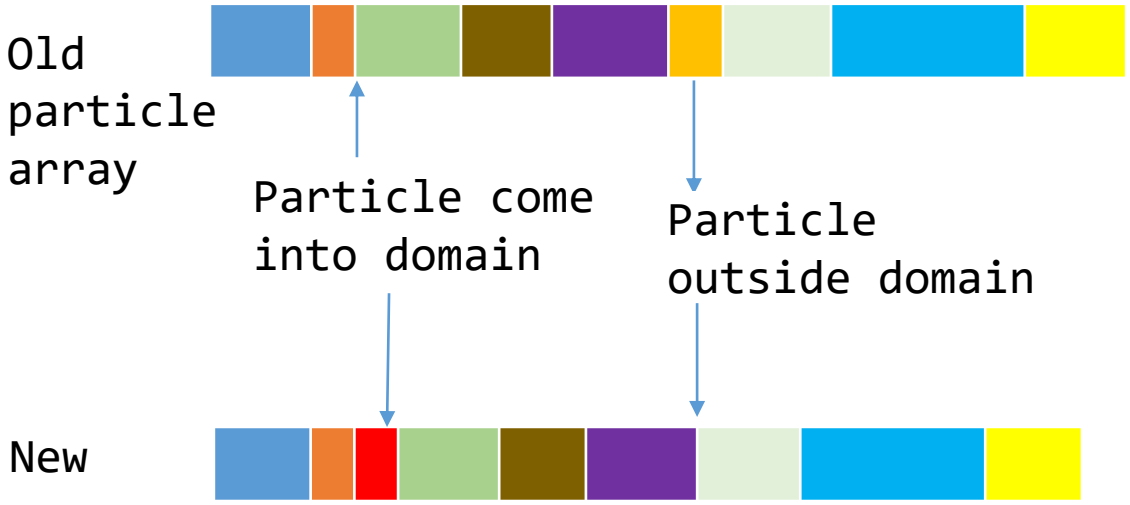
```
// find start of each cell in sorted particle list by comparing with previous hash value
// one thread per particle
__global__ void findBCellStartD(
                uint   numBodies,
 uint   gridStart,uint Ngridlb,uint Ngridrbs,
        uint   Nlh1, uint Nlb1, uint Nrh1,uint Nrb1,
               uint   Pidlh,
               uint   Pidrh,
               uint   Pidlb,
               uint   Pidrb,
               uint2* particleHash, //return
               uint*  cellStart,//return, include the prehead
               uint*  cellStartB //return
               )
{
    uint i= blockIdx.x*blockDim.x + threadIdx.x; // for all particles

    if(i<numBodies)
   {
    int cell= particleHash[i].x-gridStart;

   if(Nlh1!=0 && i < Nlh1)  // left  halo
    {
      if (i > 0)
        {
         if (cell != particleHash[i-1].x-gridStart && cell>=0) cellStartB[cell] = i;
         }
      else
        {
          if (cell>=0) cellStartB[cell] = i;
         }
    }
    else if(Nlb1!=0 && (i >=Pidlh && i < Pidlh+Nlb1)) // left boundry, write data to cellStartB
    {

      if(i > Pidlh)
        {
        if (cell != particleHash[i-1].x-gridStart && cell>=0) cellStartB[cell] = i;
         }
else
        {
         if (cell>=0) cellStartB[cell] = i;
        }
    }
```

Particle index

Grid address

neiglist.cu

- resortParticleIdD()



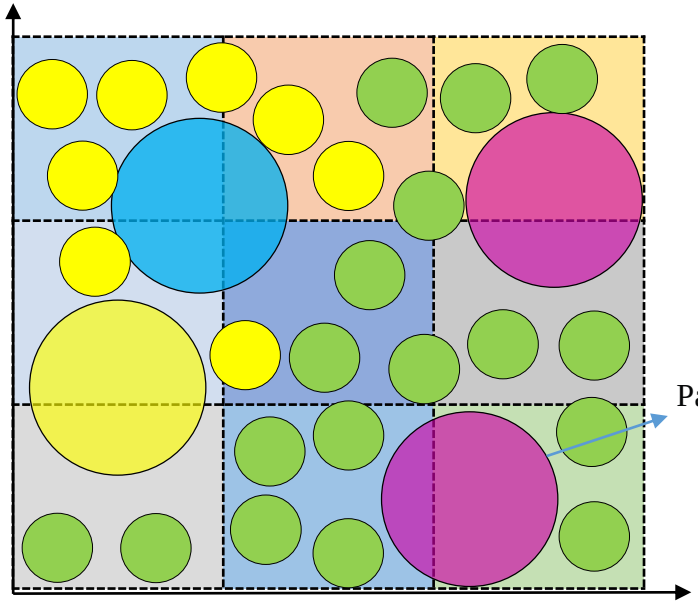
Reorder the particle index for all particle properties arrays (array length=Np)

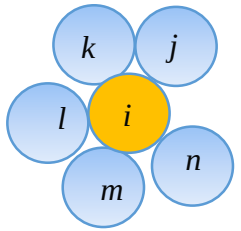
```
// resort particle index according to cell index
// need to memset both ends of arrays
__global__ void resortParticleIdD(uint numBodies,
        int    maxCnPerParticle,uint prehead,
           //uint idelete,
    uint Nlh, uint Nrh,
    uint Nlh1, uint Nlb1, uint Nrb1,

           //reserve prehead for the incoming particles
           uint* oldIP,
    uint2* particleHash,uint2* particleHash0,
    double3* pos,double3* oldpos,
    double3* pdis,double3* oldpdis,
    double3* angv,double3* oldangv,

    double* rad,double* oldrad,
    double* rmass,double* oldrmass,
    double* inert,double* oldinert,
    uint *matId,uint* oldmatId,

   double3* disptw, double3* olddisptw,
   double* fricpw,double* oldfricpw,
   uint3* contactEnd, uint3* oldcontactEnd,
   double3* Engdisp,  double3* oldEngdisp,
   double* Engdispw, double* oldEngdispw,
   double4* EngdispVarw, double4* oldEngdispVarw)
```

neiglist.cu

- neighborarrayD()

```
__global__ void neighborarrayD(uint prehead,
        uint  numBodies,
        uint  gridStart,
        uint Ngridlb, uint Ngridrbs,
        uint Ngridlh, uint Ngridrb,

        uint pidlh,uint pidrbs,uint pidrb,uint pidrh,
        uint Nlh1,uint Nlb1,uint Nrb,uint Nrh,

        double* size,
        uint* Cn,
        uint* sumCN,
        uint2*   particleHash, //gridadress is global value
        uint*    cellStart,
        uint*    cellStartB,

        double3  worldOrigin, //local

        uint3   gridSizeh,
        uint3   gridSizel,

        double3  cellSizeh,
        double3  cellSizel,

        int      maxCnPerParticle,
        double3* pos, //update here
        double*  rad,

// return value
        uint* Anei,
        uint* njgi,
        uint* njli)
```





Parallelized by particles

Anei array

jgi + jli neighbours

For each particle *i*,

1) use `particleHash to get grid address`
   `search the neighbor grids (>= 9 in 2D and 27 in 3D)`
2) `For each grid use` cellstart to get the neighbor particle index j,k,l, etc.
3) Check if distance (i-k)< critical distance of the neighbor list, if yes, add to Anei
4) For Anei, if (j>i), add to the beginning (jgi++), else add to the end (jli++), CN[i] makes sure Anei is long enough to hold the total number of neighbor particles

neiglist.cu

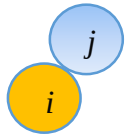
- pairlistD()
- For each particles, create pair list

For each pair i –j,
check oldLn[oldIlist] if j appears in the oldLn list, if yes, contact in previous time step, set
dispt[Ilist]=olddispt[oldIlist];
fricp[Ilist]=oldfricp[oldIlist];
Which will be used in force calculate for the tangential force component.

```
pairlistD(uint    prehead,
                          uint   Pidlh,
                          uint   Pidrb,
                          uint   Nlb1,
                          uint   Nrb1,

                          uint    oldIPlength,
                         // int     idelete,
                           uint* Cn,
                           uint* sumCN,

                          uint*   oldIP,
                          uint2*  particleHash,
                          uint*   sjgi,
                          uint*   oldsjgi,
                          uint*   oldLn,// return
                          uint*   Anei,
                          uint*   njgi,
                          uint*   njli,
                          double3 *olddispt,
                          double  *oldfricp,
                          int     maxCnPerParticle,
                          uint    numBodies,
  double* oldEngdisppair,
          double* Engdisppair,
  uint* oldcontactEndpair,
  uint* contactEndpair,
                          // return value
                          uint*   Lp,
                          uint*   Ln,
                          double3 *dispt,// return
                          double *fricp)// return
```

neiglist.cu

- **calculateforcepairD()**



Parallelized by contact candidate pair

// cauculate force and torque for each contact particle pair

calforceij(posi,posj,pdisi,pdisj,angvi,angvj,radi,radj,
rmassi,rmassj,dispti,fricpi,matIdi,matIdj,
dt,&newEngdispVarpairi,
&olddispti,&oldfricpi,&icontacti,&fi,&ti,&tj);

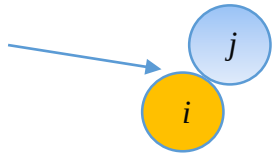
```
if(icontacti==1) //if contact, set force, torque to forcepair array index Ilist
{
forcepair[Ilist] = fi;
torqpairi[Ilist] =ti;
torqpairj[Ilist] =tj;
dispt[Ilist]=olddispti;
fricp[Ilist]=oldfricpi;
contactpair[Ilist]=1;
}
```

```
calculateforcepairD(uint     prehead,
                    int totalcontacts,
                    double3* pos,
                double3* pdis,
                double3* angv,
                double* rad,
                double* rmass,
                      uint* matId,
                    uint* Lp,
                    uint* Ln,
                    uint* oldLn,
                    double3 *dispt,
                    double *fricp,
                    double3 *olddispt,
                    double* oldfricp,
    double* oldEngdisppair,
uint* oldcontactEndpair,

   double* Engdisppair,
   double4* EngdispVarpair,//input && output
   double diam,
   double xmin,
   double hmin,
   double hmax,
                    double  dt,
                    // return value
                    double3* forcepair,
                    double3* torqpairi,
     double3* torqpairj,
                    double3* pospairi,
                    double3* pospairj,
                    uint* contactpair,

   uint* contactEndpair
   )
```

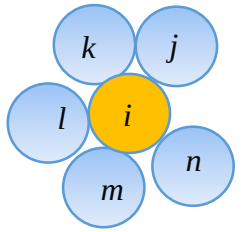
neiglist.cu

- sumallforceiD

```
sumallforceiD(uint     prehead,
uint numBodies,
int     maxParticlesPerCell,
uint      totalcontacts,
uint*   Cn,
uint*   sumCN,

uint* njgi,
uint* njli,

double3* forcepair,
double3* torqpairi,
double3* torqpairj,
uint* contactpair,
uint* Anei,

double fmat, double Wmin, double rmin, double diam,
uint* contactEndpair,
double* Engdisppairi,
double* rad,

// return value
double3* forcei,
double3* torquei,
uint* contacti,
uint3* contactEnd,
double3* Engdisp,
double* BriProb)
```



Force pair i-j

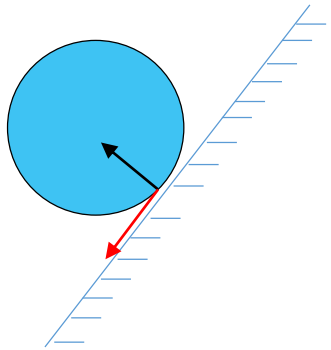


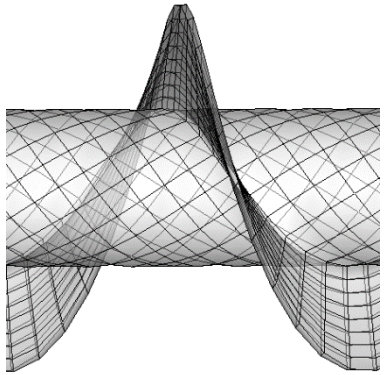
All forces on i

```
for(i=0;i<njgi[index];i++)
{
 k=Anei[sumCN[index]+i];

  if(k != 0xffffffff)
  {
  force += forcepair[k];
  torque += torqpairi[k];
  contact += contactpair[k];
  }
}

for(i=0;i<njli[index];i++)
{
 k=Anei[sumCN[index]+Cn[index]-1-i]-numBodies;

  if(k>=0 && k<totalcontacts)
  {
  force -= forcepair[k];
  torque += torqpairj[k];
  contact += contactpair[k];

  }
}
```

neiglist.cu

- calculateforcepwD()





Wall mesh

For each particle, use particle position to get grid position and grid address index:
gridPos = calcGridPos(posi,meshOri,cellSize);
gridAddress = calcGridAddress(gridPos,meshSi);

Pre-treat meshes to generate a possible contact list of mesh face index list for each grid address GridIdx, the use FaceHashStart[gridAddress+Meshstart[m]] to get the start Face index of the list, check if particle contact with this face by using **hopp_hit(),**

If contact, calculate p-w forces using calforceiw(), and add forces and torques to particle i

See treatmesh.cu for details

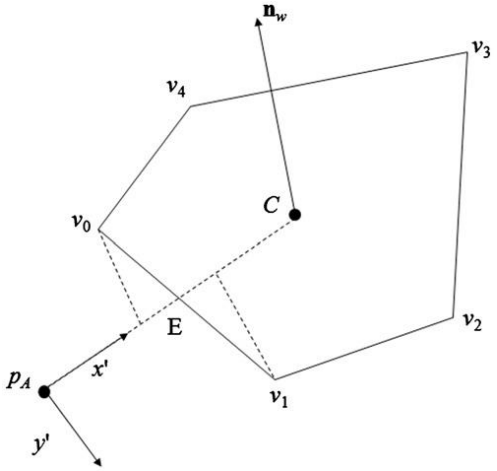
```
calculateforcepwD(uint    prehead, uint
numBodies,
uint Nmesh,
        uint3*   meshSize,
        double3* meshOrigin,
        double3* meshEnd,

        double3  cellSize,
    double3* pos,
    double3* pdis,
    double* rad,
    double* rmass,
    double3* angv,

        // wall input ------------
        uint* FaceHashStart,
        uint2* FaceHash,
        uint4* Fnodeid,
        double3* Posnode,
        uint4* sharePE,
        uint* sharePV,
        uint4* Ehead,
        uint* Vhead,

        uint* Nfacestart,
        uint* Nnodestart,
        uint* FaceHstart,
        uint* Meshstart,
         uint NumOfRotation,
         uint NumOfTranslation,
         uint NumOfVibration,

         uint it,double dt,
         double real_dt,
         double diam,
 double fmat,double Wmin, double rmin,
 double* Engdispw,
 double4* EngdispVarw,//input && output
    double xmin,
  double hmin,
  double hmax,
        // return value-------------
        double3* disptw,
        double* fricpw,

        double3* force,
        double3* torque,
        uint3* contactEnd,
        double3* Engdisp,
        double* BriProb)
```

neiglist.cu

- **hopp_hit()**

  Check if particle contact with mesh face,

  if yes, return overlap and contact normal direction vector



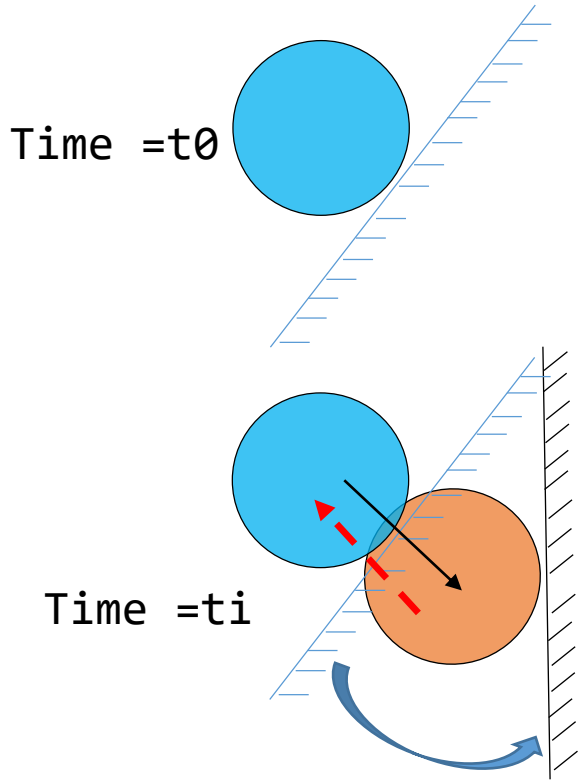

```
__device__ void hopp_hit(uint index,
                         double3 posi,
                         double radi,
                         double radmin,
                         uint edgenum,
                         double3 v1,double3 v2,double3 v3,double3 v4,
                         uint4 spe,
                         uint4 spv,
                        // uint4 spvv,
                         uint4 eh,
                         uint4 vh,
                       //  uint4 vhh,
                         // return value --------------
                         short *ihit,
                         double4 *disth,
                         uint *mark)
```

Ref: Junwei Su , ZhaolinGu, XiaoYunXu. Discrete element simulation of particle flow inarbitrarily complex geometries. Chemical Engineering Science. 66 (2011): 6069-6088

Wall Movement: GPU level

- In calculateforcepwD()

```
 for(m=0;m<Nmesh;m++) //m=0,fix; m=1,move
    {
……
// treat particle position when they are near or inside the rotary part
       RotateId=IsTimeToRotate(it, real_dt,NumOfRotation);
      if(RotateId<NumOfRotation && m_dRotation[RotateId].meshId==m)
      {
      irotation=RotateId+1;
      rotaterate =-m_dRotation[RotateId].rotatespeed*2*pi/60*real_dt/dt;
//reduced rotate rate
      theta=rotaterate*it*dt;
      posi=RotationPositionConvert(posi,m_dRotation[RotateId].axis, theta);
      }
      else
      irotation=0;
…….
```



Note: Instead of moving wall, we move particle in the reverse way of wall, then detect contact of particle-wall (then treat interaction vector back), because the contact face list is pretreated in order to reduce computation time.

neiglist.cu

- **calforceiw()**

Calculate p-w forces and torques according to equations, a wall is treated as a particle

Forces and torques	Symbols	Equations
Normal elastic force	$\mathbf{f}_{cn,ij}$	$-\frac{4}{3}E^*\sqrt{R^*}\boldsymbol{\delta}_n^{3/2}\mathbf{n}$
Normal damping force	$\mathbf{f}_{dn,ij}$	$-c_n(8m_{ij}E^*\sqrt{R^*\boldsymbol{\delta}_n})^{1/2}\mathbf{v}_{n,ij}$
Tangential elastic force	$\mathbf{f}_{ct,ij}$	$-\mu_s\lvert\mathbf{f}_{cn,ij}\rvert(1-(1-\boldsymbol{\delta}_t/\boldsymbol{\delta}_{t,\max})^{3/2})\boldsymbol{\delta}_t \quad (\boldsymbol{\delta}_t<\boldsymbol{\delta}_{t,\max})$
Tangential damping force	$\mathbf{f}_{dt,ij}$	$-c_t(6\mu_s m_{ij}\lvert\mathbf{f}_{cn,ij}\rvert\sqrt{1-\lvert V_t\rvert/\boldsymbol{\delta}_{t,\max}}/\boldsymbol{\delta}_{t,\max})^{1/2}\mathbf{v}_{t,ij} \quad (\boldsymbol{\delta}_t<\boldsymbol{\delta}_{t,\max})$
Coulumb friction force	$\mathbf{f}_{t,ij}$	$-\mu_s\lvert\mathbf{f}_{cn,ij}\rvert\boldsymbol{\delta}_t \quad (\boldsymbol{\delta}_t\geq\boldsymbol{\delta}_{t,\max})$
Torque by normal force	$\mathbf{M}_{n,ij}$	$\mathbf{R}_{ij}\times(\mathbf{f}_{cn,ij}+\mathbf{f}_{dn,ij})$
Torque by tangential force	$\mathbf{M}_{t,ij}$	$\mathbf{R}_{ij}\times(\mathbf{f}_{ct,ij}+\mathbf{f}_{dt,ij})$
Rolling friction torque	$\mathbf{M}_{r,ij}$	$\mu_{r,ij}R_i\lvert\mathbf{f}_{n,ij}\rvert\hat{\boldsymbol{\omega}}_{t,ij}^n$

```
__device__ void calforceiw(
        // input from particle index
                double3 posi,
double3 pdisi,
double radi,
double rmassi,
double3 angvi,
                double4 disth,
double3 olddptw,
double oldfcw,
// input value
double dt,

uint itranslate,
uint irotation,
uint ivibrate,

double4 *newEngdispVarw,
// return value
double3 *dptw,
double *fcw,
        double3 *forceiw,
double3 *toriw)
```

Wall movements

neiglist.cu

- **updatepositionD()**
- At each time step, use updated force and torque of each particle to update the position and (translation and angular) velocity of each particle

```
__global__ void updatepositionD(uint prehead,
        uint  numBodies,
        double3* pos, //input, return
        double3* pdis, //input, return
        double3* angv,   //input, return
        double* rad,
        double* rmass,
        double* inert,
        double3* force,
        double3* torque,
        double3* qi,    //input, return
        double*  keng,
        double radmin,
        double  dt,
double diam,
double hmin,
        double hmax,
        double xmin,
// return value
         uint* partiall,
 double* partminhz,
         double* partmaxhz,
         double* partkeng)//return
```

Data output

- WriteParticleToTecplot()

Step 1) copy data from GPU to CPU,
gather CPU data to host process for MPI case (not applicable to current code)

```
cpuErrchk(cudaMemcpy(outputpos,m_dpos+m_hprehead*3,m_
numParticles*3*sizeof(double),cudaMemcpyDeviceToHost)
);

MPI_CHECK(MPI_Gatherv(outputpos,m_numParticles*3,MPI_
DOUBLE,hpos_total,count,displs,MPI_DOUBLE,0,MPI_COMM_
WORLD));
     free(outputpos);
```

Step 2) write data to data file path and file name according visualization software data format

```
if(rank==0)
     {
    totalNode=0;
    totalFace=0;

   for(i=0;i<Nmesh;i++)
   {
   totalNode+=Nnode[i];
   totalFace+=Nface[i];
   }
     if((fp1=fopen("output/particle.dat","wb+"))==NULL){
     printf("Cannot open file particle01.dat! %d\n",it);
     exit(0);
 }

 tsd=(it)*real_dt;
 sprintf(str1,"%06.3f",tsd);

      if(it==1)
      {
      fprintf(fp1,"TITLE     = \"GAMBIT to Fluent File\"\n");
      fprintf(fp1,"VARIABLES = \"X\"\n");
      fprintf(fp1,"\"Y\"\n");
      fprintf(fp1,"\"Z\"\n");
  fprintf(fp1,"\"Velocity\"\n");
      fprintf(fp1,"\"Engdisp\"\n");
  fprintf(fp1,"\"Engdispiiw\"\n");
  fprintf(fp1,"\"Engdispiw\"\n");
      fprintf(fp1,"\"Radius\"\n");
      fprintf(fp1,"\"CollisionNum\"\n");

      fprintf(fp1,"DATASETAUXDATA Common.VectorVarsAreVelocity=\"TRUE\"\n");
      }

    for(i=0;i<Nmesh;i++)
    {
     fprintf(fp1,"ZONE T=\"ZONEW%s\"\n",m_hmeshFile[i].name);
      fprintf(fp1," N=%8d, E=%8d, ZONETYPE=FEQuadrilateral\n",Nnode[i],Nface[i]);
      fprintf(fp1," DATAPACKING=POINT\n");
      fprintf(fp1," AUXDATA Common.BoundaryCondition=\"Wall\"\n");
      fprintf(fp1," AUXDATA Common.IsBoundaryZone=\"TRUE\"\n");
      fprintf(fp1," DT=(DOUBLE DOUBLE DOUBLE )\n");
```

Tecplot format

Dimensionless parameters

Parameters are reduced/dimentionless in the code, need to be vonvert to the real when output variables:

- Particle size, length related variables: reduced by "diam"

```
m_hfeedpro[i].xmin /=diam;
m_hfeedpro[i].xmax /=diam;
```

- Time step: real_dt=dt*sqrt(diam/9.81); `// in initialization.cpp`
- Velocity: `vel_real =vel_reduced*sqrt(gg*diam)/dt;`

In WriteParticleToTecplot():

```
Double velmag=sqrt(hVel_total[3*i]*hVel_total[3*i]+
hVel_total[3*i+1]*hVel_total[3*i+1]+
hVel_total[3*i+2]*hVel_total[3*i+2]);
velmag=velmag*sqrt(gg*diam)/dt; //unit m/s
```

- Forces: reduced by fac

```
fac=(pi*denp*9.81*diam*diam*diam)/6.0; // in initialization.cpp
```

```
for(i=0;i<m_totalcontactsSum;i++)
{ forcepairmag=sqrt(hforcepair_total[i*3]*hforcepair_total[i*3]+ \
                    hforcepair_total[i*3+1]*hforcepair_total[i*3+1]+ \
                    hforcepair_total[i*3+2]*hforcepair_total[i*3+2]); //unit Fij/mg
```

Write data for rerun

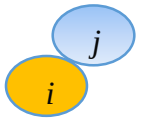
- Copy data from GPU to CPU before write.
- Write all the variables needed for rerun to data file, e.g. “preflow.dat” file

```
void WriteParticleForRerun()
{
       sprintf(str1,"%02d",rank);
       sprintf(str,"preflow%s.dat",str1);

        if((fp1=fopen(str,"wb+"))==NULL)printf("Cannot open file %s!",str);
        fprintf(fp1,"%8d %8d %8d %8d %8d %8d %8d %8d %8d %8d %8d %8d\n",
it,m_numParticles,m_totalcontacts,itime, \
FeedId,m_htotalout,NallocateH,NallocateM,NallocateL,Feedstart,hminflag,itstarthmin);

if(m_numParticles>0)
{
  CopyParticleArrayDeviceToHost(m_hprehead, m_numParticles);
  CopyNeigListArrayDeviceToHost(m_totalcontacts);

        for(i=0;i<m_numParticles;i++){
        fprintf(fp1,"%15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %4d\n",
m_hpos[3*i],  m_hpos[3*i+1],  m_hpos[3*i+2],
m_hpdis[3*i], m_hpdis[3*i+1], m_hpdis[3*i+2],
            m_hangv[3*i], m_hangv[3*i+1], m_hangv[3*i+2],
            m_hrad[i], m_hrmass[i], m_hinert[i], m_hmatId[i]);
}

        for(i=0;i<m_numParticles;i++){
         fprintf(fp1,"%8u %15.6g %15.6g %15.6g %15.6g\n",
 m_holdsjgi[i],m_holddisptw[3*i],m_holddisptw[3*i+1],m_holddisptw[3*i+2],m_holdfricpw[i]);
}

       for(i=0;i<m_numParticles;i++){
         fprintf(fp1,"%8u %8u %8u %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g\n",
 m_hcontactEnd[3*i],m_hcontactEnd[3*i+1],m_hcontactEnd[3*i+2],m_hEngdisp[i*3],m_hEngdisp[i*3+1],m_hEngdisp[i*3+2],m_hEngdispw[i],\
 m_hEngdispVarw[4*i],m_hEngdispVarw[4*i+1],m_hEngdispVarw[4*i+2],m_hEngdispVarw[4*i+3]);
}

        for(i=0;i<m_totalcontacts;i++){
        fprintf(fp1,"%8u %15.6g %15.6g %15.6g %15.6g",
m_holdLn[i],m_holddispt[3*i],m_holddispt[3*i+1],m_holddispt[3*i+2],m_holdfricp[i]);
        fprintf(fp1,"\n");
        }

for(i=0;i<m_totalcontacts;i++){
        fprintf(fp1,"%8u %15.6g %15.6g %15.6g %15.6g %15.6g",\
         m_holdcontactEndpair[i],m_hEngdisppair[i],m_hEngdispVarpair[4*i],\
 m_hEngdispVarpair[4*i+1],m_hEngdispVarpair[4*i+2],m_hEngdispVarpair[4*i+3]);
        fprintf(fp1,"\n");
        }

       cpuErrchk(cudaMemcpy(m_hcontactSizepair,m_dcontactSizepair,6*Nsect*sizeof(uint),
 cudaMemcpyDeviceToHost));

for(i=0;i<6*Nsect;i++)
fprintf(fp1,"%8u ", m_hcontactSizepair[i]);

fprintf(fp1,"\n");

}
        fclose(fp1);
}
```

Energy dissipation

The dissipated energy is commonly calculated by integrating the normal damping force F_{dn} and the tangential damping force F_{dt} with respect to their overlaps over the entire contact period $t_{contact}$

$$E_d = \int_0^{tcontact} (|\mathbf{f}_{dn}|d\delta_n + |\mathbf{f}_{dt}|d\delta_t)$$



Force pair i-j

1) __device__ void **calforceij()**

*newEngdisppair=
make_double4(fdampn*9.81, length(fdamp)*9.81, dsnorm, dispt);

$|\mathbf{f}_{dn}|$ $|\mathbf{f}_{dt}|$ $d\delta_n$ $d\delta_t$

3) Pair information, need to copy if contact in previous step, so in **pairlistD():**

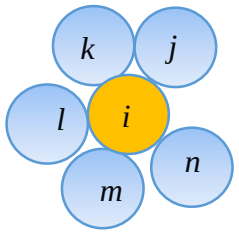
```
Engdisppair[Ilist] = oldEngdisppair[oldIlist];
contactEndpair[Ilist] = oldcontactEndpair[oldIlist];
```

2) **calforceij(&**newEngdispVarpairi);

```
//------ treat energy dissipation -----------------------------
 if(0.5*(posi.z+posj.z)>=hmin && 0.5*(posi.z+posj.z)<hmax && 0.5*(posi.x+posj.x)>xmin) //region to calculate engdisp
 {
 if(oldEngdispVarpairi.z>0.0 && newEngdispVarpairi.z>0.0 && fabs(newEngdispVarpairi.z-oldEngdispVarpairi.z)>deltas &&
(oldEndpair==5 || oldEndpair==10) ) //contact in progress
 {
 Engdisppair[Ilist] += 0.5*(fabs(oldEngdispVarpairi.x)+fabs(newEngdispVarpairi.x))*fabs(newEngdispVarpairi.z-
oldEngdispVarpairi.z)+\
                       0.5*(fabs(oldEngdispVarpairi.y)+fabs(newEngdispVarpairi.y))*fabs(newEngdispVarpairi.w-
oldEngdispVarpairi.w);
 contactEndpair[Ilist]=oldEndpair;
 EngdispVarpair[Ilist]=newEngdispVarpairi;
 }
 else if(oldEngdispVarpairi.z<=0.0 &&newEngdispVarpairi.z>0.0 && oldEndpair==0 && fabs(newEngdispVarpairi.z-
oldEngdispVarpairi.z)>deltas) //start to contact
 {
 Engdisppair[Ilist] = 0.5*(fabs(oldEngdispVarpairi.x)+fabs(newEngdispVarpairi.x))*fabs(newEngdispVarpairi.z-0.0)+\
                       0.5*(fabs(oldEngdispVarpairi.y)+fabs(newEngdispVarpairi.y))*fabs(newEngdispVarpairi.w-0.0);

 if(fabs(veli-velj)>3.5)
 {
 contactEndpair[Ilist]=10; //collision with a static particle
 }
 else if((veli>0.5 && velj>0.5))
 contactEndpair[Ilist]=5; //collision with a dynamic particle
 else
 contactEndpair[Ilist]=0;

 EngdispVarpair[Ilist]=newEngdispVarpairi;
 }
 else if(oldEngdispVarpairi.z>0.0 && (((oldEndpair==5 || oldEndpair==10) && newEngdispVarpairi.z<=0.0) || \
        (oldEndpair==10 && newEngdispVarpairi.z>0.0 && fabs(newEngdispVarpairi.z-oldEngdispVarpairi.z)<deltas)))
  // contact complete
 {
 ……
 }

//-------------------------------------------------------------------------------
 oldEngdisppair[Ilist]=Engdisppair[Ilist];
 oldcontactEndpair[Ilist]=contactEndpair[Ilist];
```

Energy dissipation

- Sum **Engdisppairi to get Engdispi for each particle**



All forces on i

Array length of Engdisppairi =Np

Particle property, so it needs to reorder when reorder particle index

In resortParticleIdD():

oldcontactEnd[i]=contactEnd[Pid];
oldEngdisp[i]=Engdisp[Pid];
oldEngdispw[i]=Engdispw[Pid];
oldEngdispVarw[i]=EngdispVarw[Pid];

__global__ void sumallforceiD()

```
for(i=0;i<njgi[index];i++)
{
 k=Anei[sumCN[index]+i];

  if(k != 0xffffffff)
  {
  force += forcepair[k];
  torque += torqpairi[k];
  contact += contactpair[k];

  if((contactEndpair[k]==1 || contactEndpair[k]==2) &&
      !isnan(Engdisppairi[k])
{
  contactEndi +=1;
  Engdispi += Engdisppairi[k]*Winscale;
  if(contactEndpair[k]==2)
  {
  Engdispiw += Engdisppairi[k]*Winscale;
  contactEndiw +=1;
  }
  }
  }
}
```

Energy dissipation

- WriteOutEngdsp():
- Do different analysis of engdisp data:
  e.g.,

total/bulk average engdisp,

engdisp at different axis position,

engdisp for different size group,

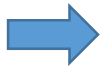
Contact Frequency

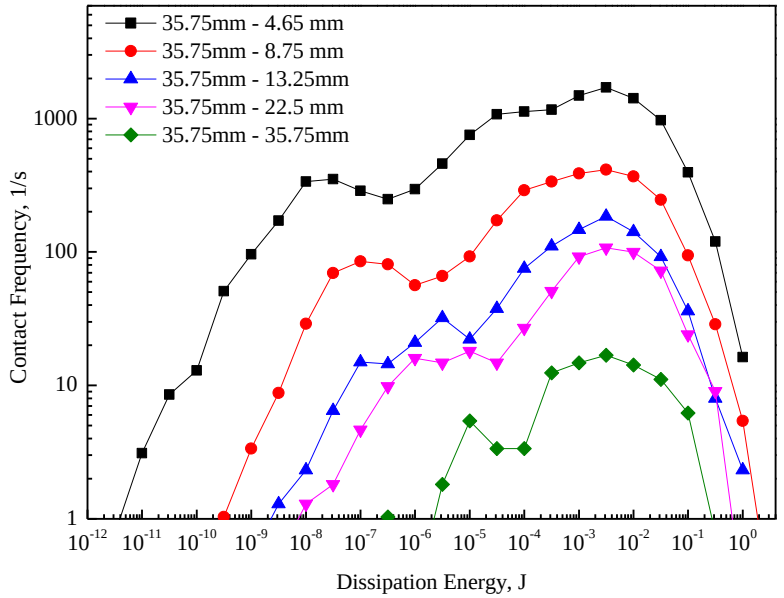
- **calculateContactFrequencyD**() in cu kernel

```
 if(fabs(radi-Size[0])<1e-6 && fabs(radj-Size[8])<1e-6)
//size i contact with size j, add to statistic of Bin
   {
    atomicAdd(&contactSizepair[0*Nsect+Bin],1);
   }
// Bin= (int) ((log10(Engdisppairmag)-log10(Emin))/gap);
```

- WriteOutContactFrequency() in WriteData.cpp

```
 for(i=0;i<Nsect;i++)
       {
double Binstart=pow(10,i*gap+log10(Emin));
        fprintf(fp1,"%15.6g %15.6g %15.6g %15.6g %15.6g %15.6g %15.6g\n",
   Binstart,m_hcontactSizepair_total[0*Nsect+i]/tdrop,m_hcontactSizepair_total[1*Nsect+i]/tdrop,\
   m_hcontactSizepair_total[2*Nsect+i]/tdrop,m_hcontactSizepair_total[3*Nsect+i]/tdrop,\
   m_hcontactSizepair_total[4*Nsect+i]/tdrop,m_hcontactSizepair_total[5*Nsect+i]/tdrop);
   }
```





How to edit, build and run cases

- **Edit**

Try to use Visual studio to edit code

- **Build**

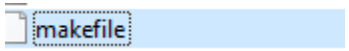
Use makefile

```
BIN              := ../GDEM

# Compilers
CUDA_PATH ?= /usr/local/cuda/8.0.61
MPI_PATH     ?= /usr/local/openmpi/1.10.7-mlx
NVCC        := $(CUDA_PATH)/bin/nvcc
#CXX         := gcc -Xlinker --allow-multiple-definition
CXX         := $(MPI_PATH)/bin/mpic++ -Xlinker --allow-multiple-definition

MPICC       :=  $(MPI_PATH)/bin/mpic++ -Xlinker --allow-multiple-definition
EXEC   ?=
```

Target

Compilers

In winscp file path:



In putty, cd to makefile path and then input “make” to compile:

```
[ganjq@m3-loginl ~]$ cd /projects/hi65/ganjq/mixing/ribbonmixer/colorxyz
[ganjq@m3-loginl colorxyz]$ make
```

After successfully compile, a GDEM (executive file) file should appear in the Bin path,
Otherwise, error information will appear in putty

Files to build

```
# Program-specific
CFILES       := AllocateArrays.cpp \
                Boundary.cpp  \
                Feed.cpp  \
                FreeArrays.cpp  \
                GPUSet.cpp  \
                Initialization.cpp  \
                Materials.cpp  \
                MemcpyHostDevice.cpp \
                Movement.cpp  \
                mpiFunctions.cpp\
                Particle.cpp  \
                ReadData.cpp  \
                WriteData.cpp  \
                dempacking.cpp

CUFILES       := dempacking.cu

OFILES=$(CFILES:.cpp=.o)
CUOFILES=$(CUFILES:.cu=.cu_o)
USERCUOFILES= treatmesh.cu_o
##

OBJ   = $(OFILES) $(CUOFILES) $(USERCUOFILES)
```

Build rule

```
# Build Rules
all: $(BIN) clean

$(BIN): $(OBJ)
     $(CXX) $(OBJ) -o  $(BIN) $(INCD) $(LIBS)

%.o : %.cpp
     $(CXX) -c  $(CCFLAGS) -o $@ $<

%.cu_o : %.cu
     $(NVCC) $(GENCODE_FLAGS) -c $(INCD) -o $@ $<
```

- **Run DEM simulations:**

Run DEM simulation on Massive 3:

- 1) correctly set the input parameters, job script for wall time and memory settings
- 2) run the job script by input “sbatch case” in putty.

```
[ganjq@m3-login1 ~]$ cd /projects/hi65/ganjq/mixing/ribbonmixer/colorxyz
[ganjq@m3-login1 colorxyz]$ sbatch case
```

- 3) check job state by input “show_job”

```
| Job ID    | Job Name | Username | Account | Allocated Gres | Requested GRES | P
riority | Hostname | Requested Memory (MB) | Status    |
+----------+----------+----------+---------+----------------+----------------+--
--------+----------+-----------------------+-----------+
| 17323693 | desktop  | ganjq    | kv42    | gpu:1          | gpu:P4:1       | 9
3743    | m3p008   | 56320                 | CANCELLED |
| 17323694 | desktop  | ganjq    | hi65    | gpu:1          | gpu:P4:1       | 6
9104    | m3p008   | 56320                 | TIMEOUT   |
```

- 4) when the job shows run, check the output.dat file to see the information or check if any error information file such as slurm-xxx.out file
- 5) if successfully completed, get data from output file for post-processing. For any variables, if the information of different time is needed, it should be added to the code before running the case.