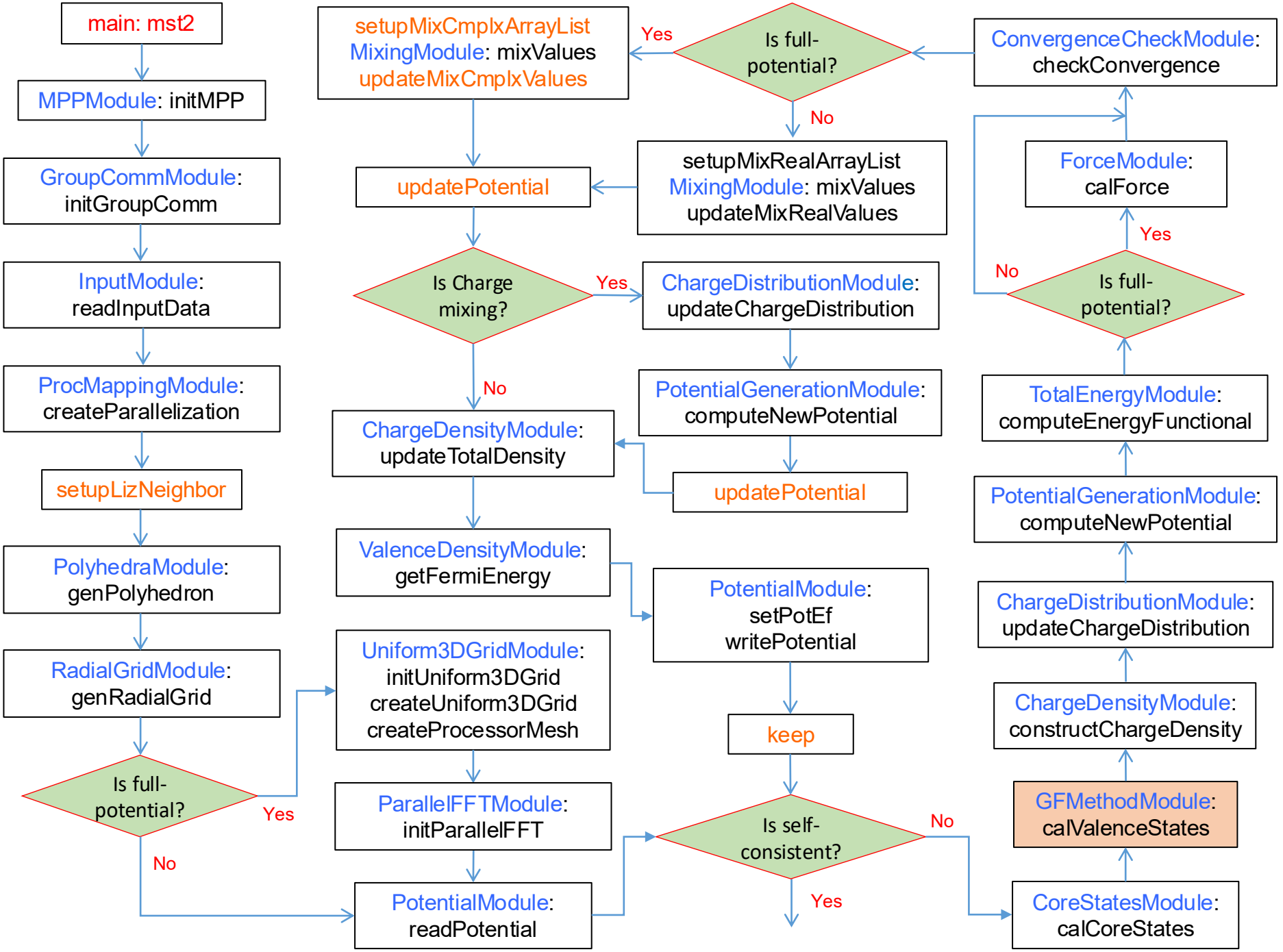
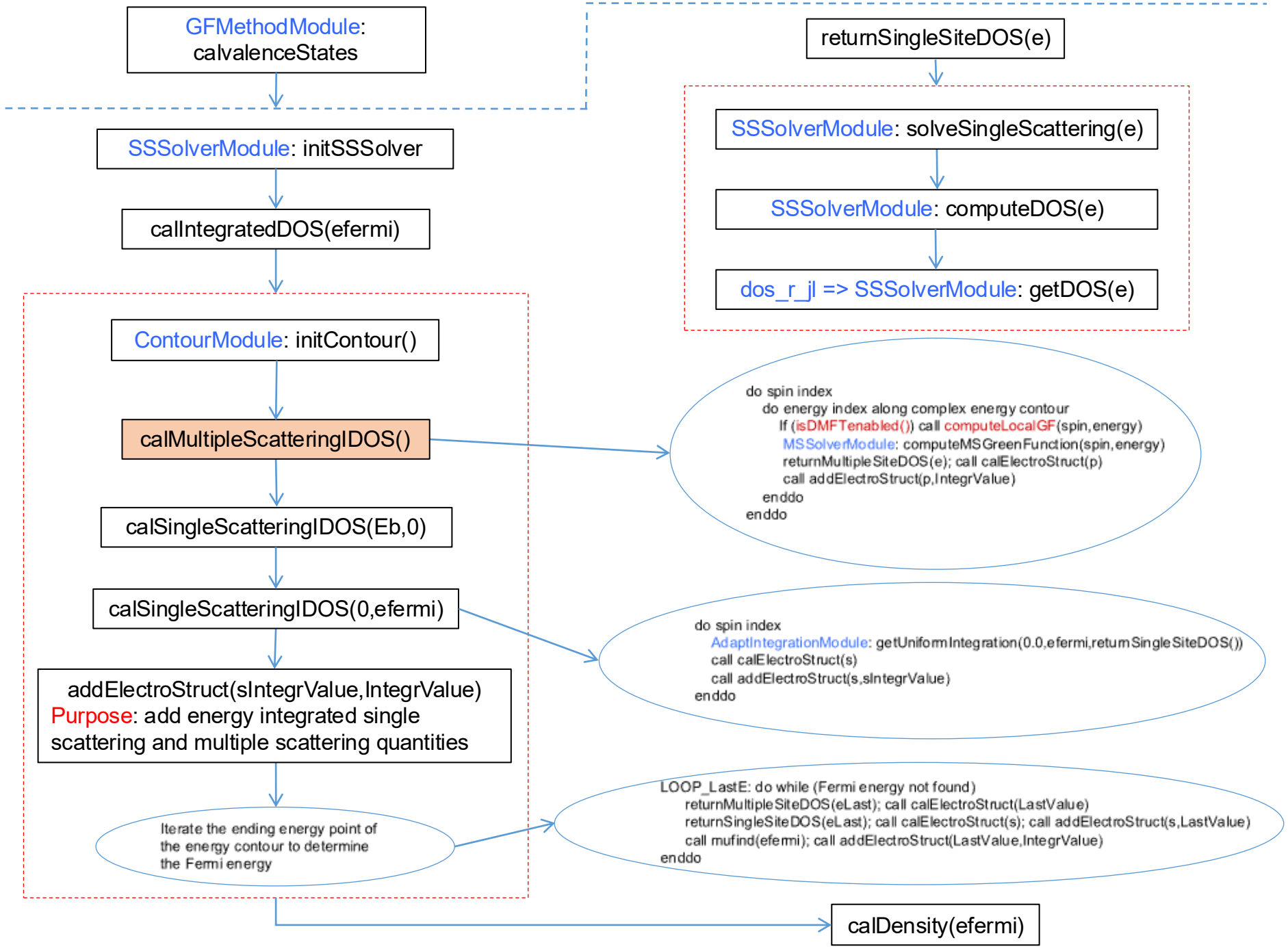
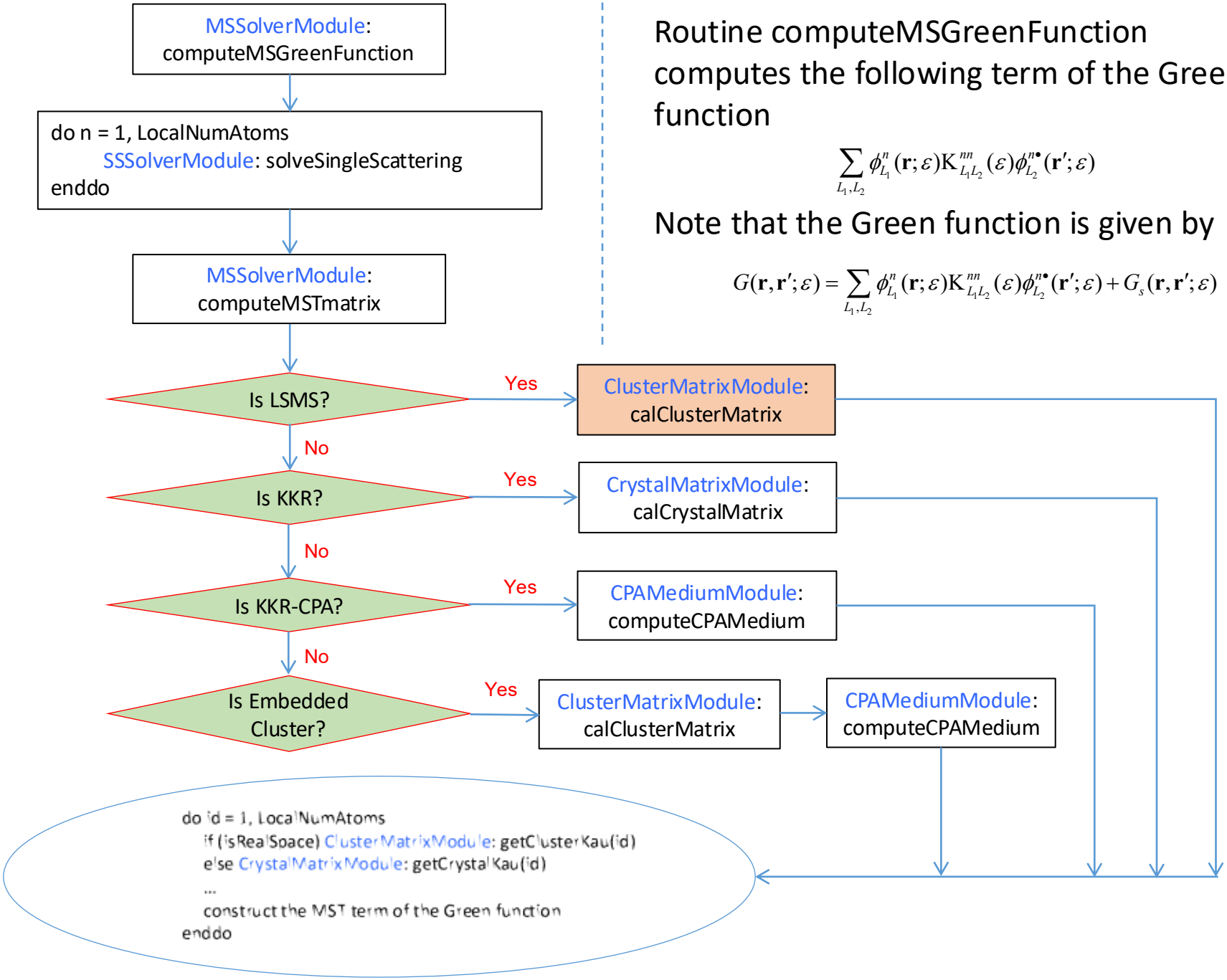

main: mst2
MPPModule: initMPP
GroupCommModule: initGroupComm
InputModule: readInputData
ProcMappingModule: createParallelization
setupLizNeighbor
PolyhedraModule: genPolyhedron
RadialGridModule: genRadialGrid
Is full-potential?
Yes
No
Uniform3DGridModule: initUniform3DGrid createUniform3DGrid createProcessorMesh
ParallelFFTModule: initParallelFFT
PotentialModule: readPotential
ChargeDensityModule: updateTotalDensity
ValenceDensityModule: getFermiEnergy
PotentialModule: setPotEf writePotential
keep
Is self-consistent?
No
Yes
CoreStatesModule: calCoreStates
GFMethodModule: calValenceStates
ChargeDensityModule: constructChargeDensity
ChargeDistributionModule: updateChargeDistribution
PotentialGenerationModule: computeNewPotential
TotalEnergyModule: computeEnergyFunctional
Is full-potential?
No
Yes
ForceModule: calForce
ConvergenceCheckModule: checkConvergence
Is full-potential?
Yes
No
setupMixCmplxArrayList MixingModule: mixValues updateMixCmplxValues
setupMixRealArrayList MixingModule: mixValues updateMixRealValues
updatePotential
Is Charge mixing?
Yes
No
ChargeDistributionModule: updateChargeDistribution
PotentialGenerationModule: computeNewPotential
updatePotential


GFMethodModule:
calvalenceStates
SSSolverModule: initSSSolver
callntegratedDOS(efermi)
ContourModule: initContour()
calMultipleScatteringIDOS()
calSingleScatteringIDOS(Eb,0)
calSingleScatteringIDOS(0,efermi)
addElectroStruct(sIntegrValue,IntegrValue)
Purpose: add energy integrated single scattering and multiple scattering quantities
Iterate the ending energy point of the energy contour to determine the Fermi energy
calDensity(efermi)
returnSingleSiteDOS(e)
SSSolverModule: solveSingleScattering(e)
SSSolverModule: computeDOS(e)
dos_r_jl => SSSolverModule: getDOS(e)
do spin index
do energy index along complex energy contour
If (isDMFTenabled()) call computeLocalGF(spin,energy)
MSSolverModule: computeMSGreenFunction(spin,energy)
returnMultipleSiteDOS(e); call calElectroStruct(p)
call addElectroStruct(p,IntegrValue)
enddo
enddo
do spin index
AdaptIntegrationModule: getUniformIntegration(0.0,efermi,returnSingleSiteDOS())
call calElectroStruct(s)
call addElectroStruct(s,sIntegrValue)
enddo
LOOP_LastE: do while (Fermi energy not found)
returnMultipleSiteDOS(eLast); call calElectroStruct(LastValue)
returnSingleSiteDOS(eLast); call calElectroStruct(s); call addElectroStruct(s,LastValue)
call mufind(efermi); call addElectroStruct(LastValue,IntegrValue)
enddo

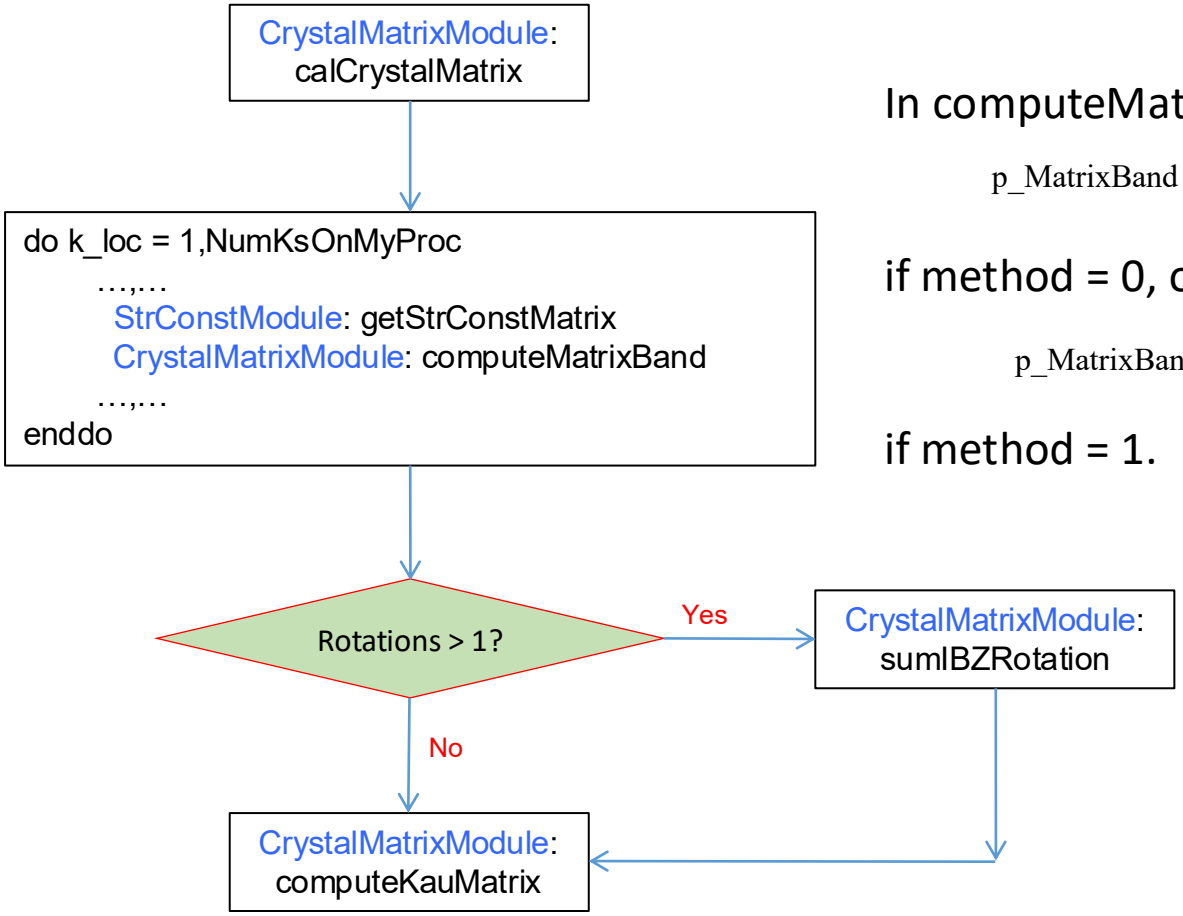


Routine computeMSGreenFunction computes the following term of the Green function

$$\sum_{L_1,L_2} \phi_{L_1}^{n}(\mathbf{r};\varepsilon)\mathrm{K}_{L_1L_2}^{nn}(\varepsilon)\phi_{L_2}^{n\bullet}(\mathbf{r}';\varepsilon)$$

Note that the Green function is given by

$$G(\mathbf{r},\mathbf{r}';\varepsilon) = \sum_{L_1,L_2} \phi_{L_1}^{n}(\mathbf{r};\varepsilon)\mathrm{K}_{L_1L_2}^{nn}(\varepsilon)\phi_{L_2}^{n\bullet}(\mathbf{r}';\varepsilon) + G_s(\mathbf{r},\mathbf{r}';\varepsilon)$$



In computeMatrixBand, the routine calculates

$$\text{p\_MatrixBand} = \left[1-\underline{\underline{\hat{\Omega}}}(\varepsilon)\underline{\underline{S}}^{T\bullet}(\varepsilon)\left(i\sqrt{\varepsilon}+\underline{\underline{B}}(\mathbf{k};\varepsilon)\right)\frac{\underline{\underline{S}}(\varepsilon)}{\sqrt{\varepsilon}}\right]^{-1}-1$$

if method = 0, or

$$\text{p\_MatrixBand} = -\left[\underline{\underline{S}}^{T\bullet}(\varepsilon)\underline{\underline{C}}(\varepsilon)+\frac{1}{\sqrt{\varepsilon}}\underline{\underline{S}}^{T\bullet}(\varepsilon)\underline{\underline{B}}(\mathbf{k};\varepsilon)\underline{\underline{S}}(\varepsilon)\right]^{-1}$$

if method = 1.

In computeKauMatrix, the routine calculates

$$\underline{\mathrm{K}}^{nn}(\varepsilon)=\frac{1}{\Omega_{BZ}}\int_{\Omega_{BZ}}\underline{\mathrm{K}}^{nn}(\overset{\mathrm{r}}{k};\varepsilon)d^3\overset{\mathrm{r}}{k}=\sqrt{\varepsilon}\,\frac{1}{\Omega_{BZ}}\int_{\Omega_{BZ}}\left\{\left[1-\underline{\underline{\hat{\Omega}}}(\varepsilon)\underline{\underline{S}}^{T\bullet}(\varepsilon)\left(i\sqrt{\varepsilon}+\underline{\underline{B}}(\mathbf{k};\varepsilon)\right)\frac{\underline{\underline{S}}(\varepsilon)}{\sqrt{\varepsilon}}\right]^{-1}_{nn}-1\right\}d^3\mathbf{k}\cdot\underline{\hat{\Omega}}^{n}(\varepsilon),\quad \text{if method} = 0;$$

or

$$\underline{\mathrm{K}}^{nn}(\varepsilon)=\frac{1}{\Omega_{BZ}}\int_{\Omega_{BZ}}\underline{\mathrm{K}}^{nn}(\overset{\mathrm{r}}{k};\varepsilon)d^3\overset{\mathrm{r}}{k}=\sqrt{\varepsilon}\,\frac{1}{\Omega_{BZ}}\int_{\Omega_{BZ}}\left\{-\left[\underline{\underline{S}}^{T\bullet}(\varepsilon)\underline{\underline{C}}(\varepsilon)+\frac{1}{\sqrt{\varepsilon}}\underline{\underline{S}}^{T\bullet}(\varepsilon)\underline{\underline{B}}(\mathbf{k};\varepsilon)\underline{\underline{S}}(\varepsilon)\right]^{-1}_{nn}\right\}d^3\mathbf{k}-\sqrt{\varepsilon}\underline{\hat{\Omega}}^{n}(\varepsilon),\quad \text{if method} = 1.$$

ClusterMatrixModule:
calClusterMatrix(e)

```
…,…
do my_atom = 1, LocalNumAtoms
!  For each atom on my MPI process treated as a center atom
   do j = 0, Neighbor%NumAtoms
!     For each atom in the LIZ of the center atom
      …,…
      do i = 0, Neighbor%NumAtoms
!        For each atom in the LIZ of the center atom
         …,…
         rij = posj - posi
         gij => RSpaceStrConstModule:getStrConstMatrix(e,rij)
         …,…
!        Construct BigMatrix
         …,…
      enddo
   enddo
enddo
…,…
```

$$\left[i\underline{s}_i(\varepsilon)-\underline{c}_i(\varepsilon)\right]^{-1} \rightarrow \texttt{p\_jinvi}$$

$$\underline{\underline{I}}-\frac{1}{\sqrt{\varepsilon}}\left[i\underline{\underline{s}}(\varepsilon)-\underline{\underline{c}}(\varepsilon)\right]^{-1}\underline{\underline{g}}(\varepsilon)\underline{\underline{s}}(\varepsilon) \rightarrow \texttt{BigMatrix}$$

call invertMatrixBlock to get p_BlockMatrix

τ-matrix:

$$\underline{\tau}^{11}(\varepsilon)=\begin{bmatrix}\left[\underline{t}^1(\varepsilon)\right]^{-1} & -\underline{g}^{12}(\varepsilon) & \text{L} & -\underline{g}^{1M}(\varepsilon)\\ -\underline{g}^{21}(\varepsilon) & \left[\underline{t}^2(\varepsilon)\right]^{-1} & \text{L} & -\underline{g}^{2M}(\varepsilon)\\ \text{M} & \text{M} & \text{O} & \text{M}\\ -\underline{g}^{M1}(\varepsilon) & -\underline{g}^{M2}(\varepsilon) & \text{L} & \left[\underline{t}^M(\varepsilon)\right]^{-1}\end{bmatrix}^{-1}_{11}$$

$$=\left[\underline{\underline{t}}^{-1}(\varepsilon)-\underline{\underline{g}}(\varepsilon)\right]^{-1}_{11}$$

Note: There are *M* atoms in the LIZ of the center atom. In this expression for the τ-matrix of the center atom, the center atom is numbered as atom 1, and the rest of the atoms in the LIZ are numbered as 2, 3, …, *M*.

Note: The diagonal blocks of BigMatrix are identity matrix.

$$\underline{t}^i(\varepsilon)=\left(i\sqrt{\varepsilon}\underline{I}-\sqrt{\varepsilon}\underline{c}^i(\varepsilon)\left[\underline{s}^i(\varepsilon)\right]^{-1}\right)^{-1}$$

$$\hat{\underline{\Omega}}^i(\varepsilon)=\left[\left[\underline{s}^i(\varepsilon)\right]^{T\bullet}\left(i\underline{s}^i(\varepsilon)-\underline{c}^i(\varepsilon)\right)\right]^{-1}$$

$\underline{J}(\varepsilon)=i\underline{s}^i(\varepsilon)-\underline{c}^i(\varepsilon)$ is called Jost matrix

$$\underline{p}^1(\varepsilon)=\underline{I}-\left(\left[\underline{\underline{I}}-\frac{1}{\sqrt{\varepsilon}}\left[i\underline{\underline{s}}(\varepsilon)-\underline{\underline{c}}(\varepsilon)\right]^{-1}\underline{\underline{g}}(\varepsilon)\underline{\underline{s}}(\varepsilon)\right]^{-1}_{11}\right)^{-1} \rightarrow \texttt{p\_BlockMatrix}$$

$$\underline{W}^{11}(\varepsilon)=\left[\underline{\underline{I}}-\frac{1}{\sqrt{\varepsilon}}\left[i\underline{\underline{s}}(\varepsilon)-\underline{\underline{c}}(\varepsilon)\right]^{-1}\underline{\underline{g}}(\varepsilon)\underline{\underline{s}}(\varepsilon)\right]^{-1}_{11}-\underline{I}=\left(\underline{I}-\underline{p}^1(\varepsilon)\right)^{-1}-\underline{I}=\left(\underline{I}-\underline{p}^1(\varepsilon)\right)^{-1}\underline{p}^1(\varepsilon) \rightarrow \texttt{wau\_g}$$

$$\underline{\mathrm{K}}^{11}(\varepsilon)=\varepsilon\left[\underline{s}^1(\varepsilon)\right]^{-1}\left(\underline{\tau}^{11}(\varepsilon)-\underline{t}^1(\varepsilon)\right)\left[\underline{s}^1(\varepsilon)\right]^{-T\bullet}=\sqrt{\varepsilon}\left(\left[\underline{\underline{I}}-\frac{1}{\sqrt{\varepsilon}}\left[i\underline{\underline{s}}(\varepsilon)-\underline{\underline{c}}(\varepsilon)\right]^{-1}\underline{\underline{g}}(\varepsilon)\underline{\underline{s}}(\varepsilon)\right]^{-1}_{11}-\underline{I}\right)\hat{\underline{\Omega}}^1(\varepsilon)=\sqrt{\varepsilon}\underline{W}^{11}(\varepsilon)\hat{\underline{\Omega}}^1(\varepsilon) \rightarrow \texttt{Tau00\%kau\_l}$$

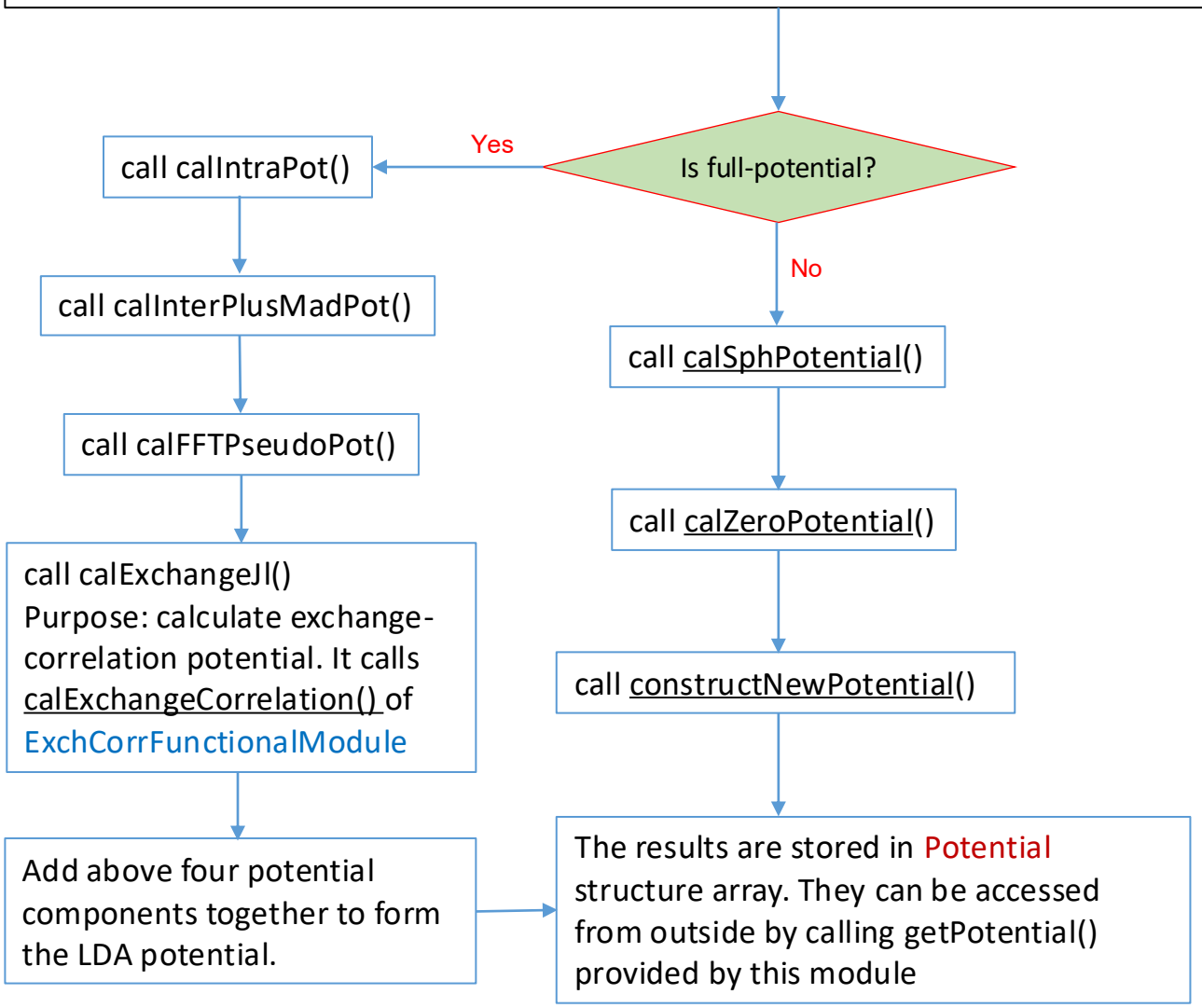
$$\underline{\tau}^{11}(\varepsilon)=\frac{1}{\varepsilon}\underline{s}^1(\varepsilon)\underline{\mathrm{K}}^{11}(\varepsilon)\left[\underline{s}^1(\varepsilon)\right]^{T\bullet}+\underline{t}^1(\varepsilon) \rightarrow \texttt{Tau00\%tau\_l}$$

ChargeDensityModule:
Subroutine constructChargeDensity()
Called by: main()
Purpose: Construct total charge density, and moment density, pseudo charge density, and non-overlapping charge density on radial grid.

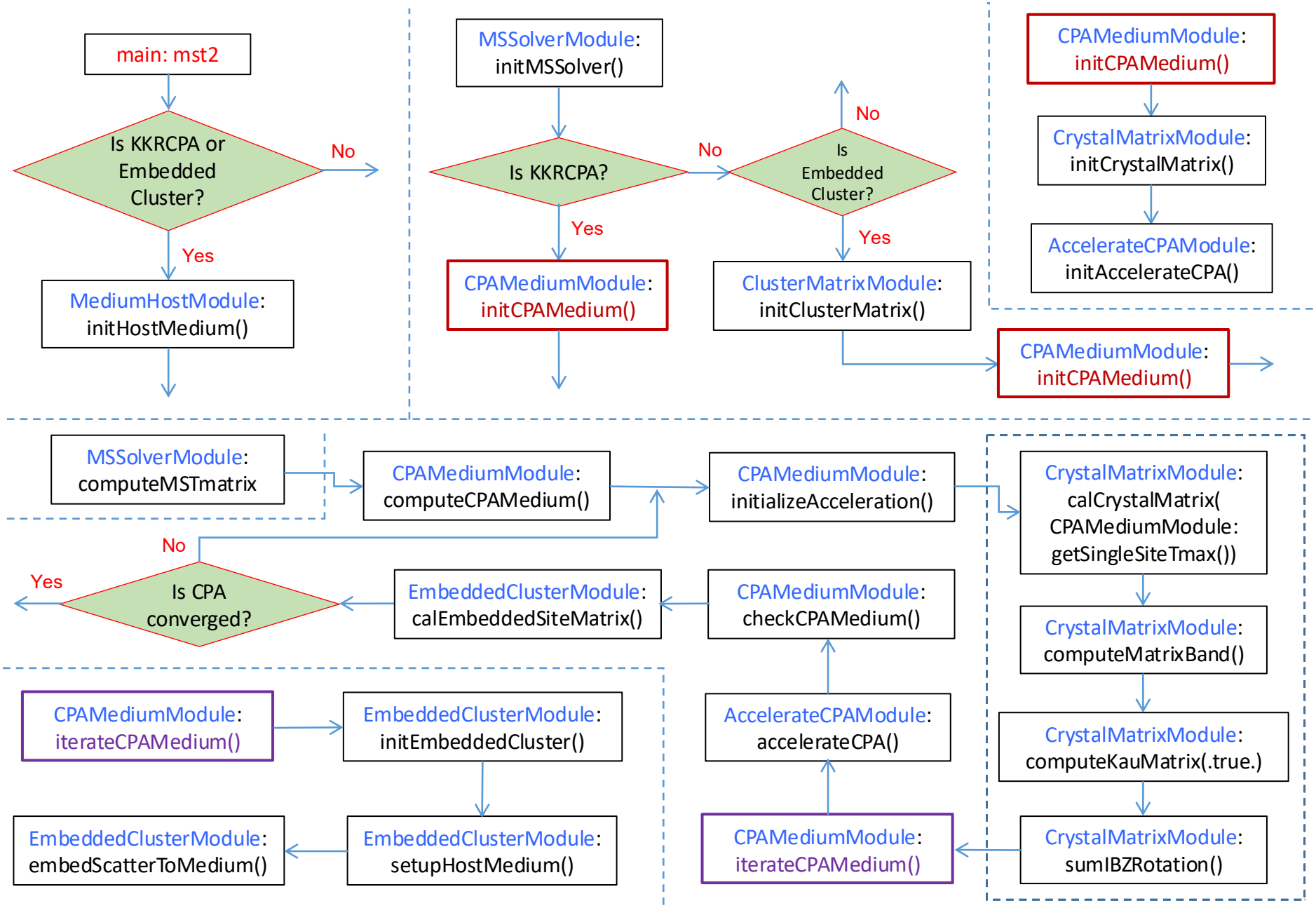
ChargeDistributionModule:
Subroutine updateChargeDistribution()
Called by: main()
Purpose: Determine the charge and moment on each atomic site, and generate a charge and moment distribution table.

Subroutine constructDensityOnGrid(density,density_type,lmax,is,targeted_grid_type,FFTMode,VisMode):
Called by: main();
calExchCorrPot() in PotentialGenerationModule
calExchCorrEnergy() PotentialGenerationModule
calFFTPseudoPot() in PotentialGenerationModule
Purpose: Construct the density of the given type on the targeted grid (either “uniform grid” or “visualization grid”) by projecting the density from the radial grid to the targeted grid. The density (charge or moment) on the radial grid is accessed by calling getChargeDensity() or getMomentDensity() provided by ChargeDensityModule. The resulting density on the targeted grid is stored in **DataStorage** system managed by DataServiceCenterModule, and is accessible by calling getDataStorage()

PotentialGenerationModule:
Subroutine computeNewPotential()
Called by: main()
Purpose: Given a charge density, which is accessible by calling getDataStorage() function provided by DataServiceCenterModule if the density is on the uniform grid or by calling getChargeDensity() provided by ChargeDensityModule if the density is on the radial grid, calculate the new LDA potential.



Calculation with a host medium (e.g. CPA medium or Typical medium)



Acceleration of LSMS

MuST/MST/src/ClusterMatrixModule.F90::
calClusterMatrix

```
#ifdef ACCEL
      pBigMatrix => aliasArray2_c(BigMatrix, dsize, dsize)
      call invertMatrixLSMS_CUDA(my_atom, pBlockMatrix, kkrsz_ns,      &
                                 pBigMatrix, dsize )
#else
      call invertMatrixBlock( my_atom, pBlockMatrix, kkrsz_ns, kkrsz_ns, &
                              BigMatrix, dsize, dsize )
#endif
```

Without GPU Acceleration

MuST/MST/src/MatrixBlockInversionModule.F90::
invertMatrixBlock

With GPU Acceleration

MuST/MST/Accelerator/invertMatrixLSMS_CUDA.F90

```
#ifdef CUDA
      …,…
      call cusolver_lsms_c(dsize,pBigMatrix,kkrsz_ns,tmpmatrixC)
      …,…
#else
!     The following line under this condition seems from an unfinished
!     work and needs to be corrected.
!     However, under the current setting , this condition is not reached
      call zblock_lu_CPU(a,lda,blk_sz,nblk,ipvt,mp,idcol,k)
#endif
```

MuST/MST/Accelerator/cusolver_LSMS_c.cu

```
extern "C"
void cusolver_lsms_c_(int *m, double _Complex *a,
int *block_size, double _Complex *b)
{
    …,…
}
```