Structure of OpenMX extension: NEB

- Introduction of the NEB method
- Implementation of NEB in OpenMX
- Usage of the NEB functionality
- Examples
- Relevant routines
- Parallelizstion
- Close loot at codes

References

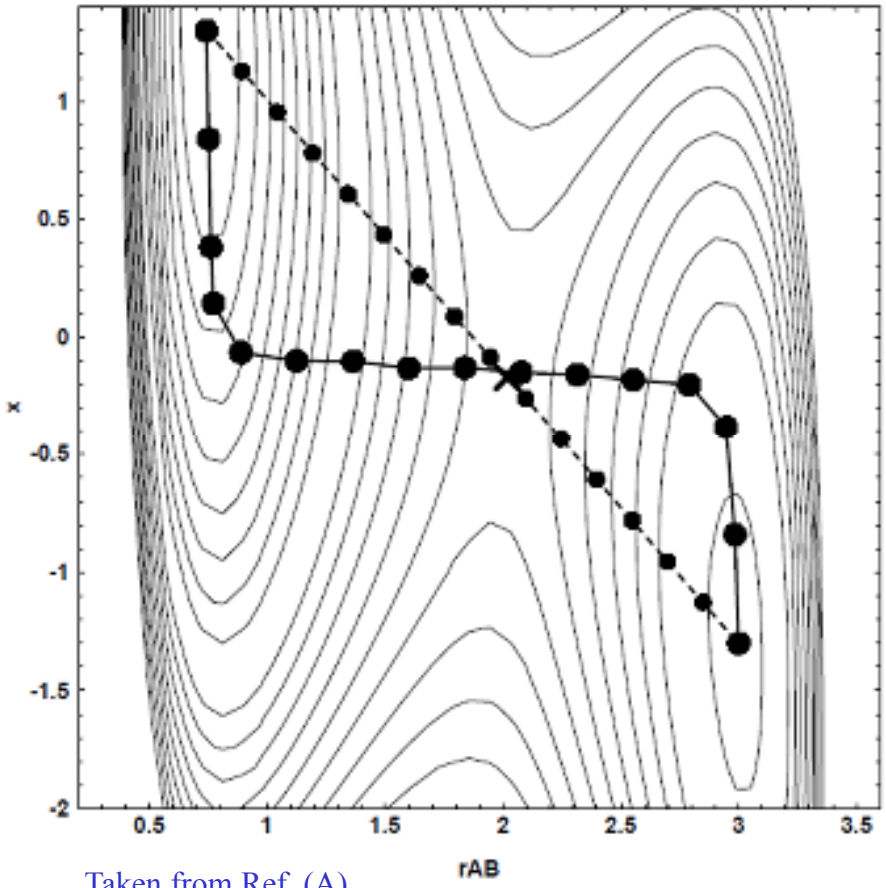
(A)
H. Jonsson, G. Mills, and K. W. Jacobsen, in *Classical and Quantum Dynamics in Condensed Phase Simulations*, edited by B. J. Berne, G. Ciccotti, and D. F. Coker (World Scientific, Singapore, 1998), p. 385.

(B)
G. Henkelman and H. Jonsson, JCP 113, 9978 (2000).

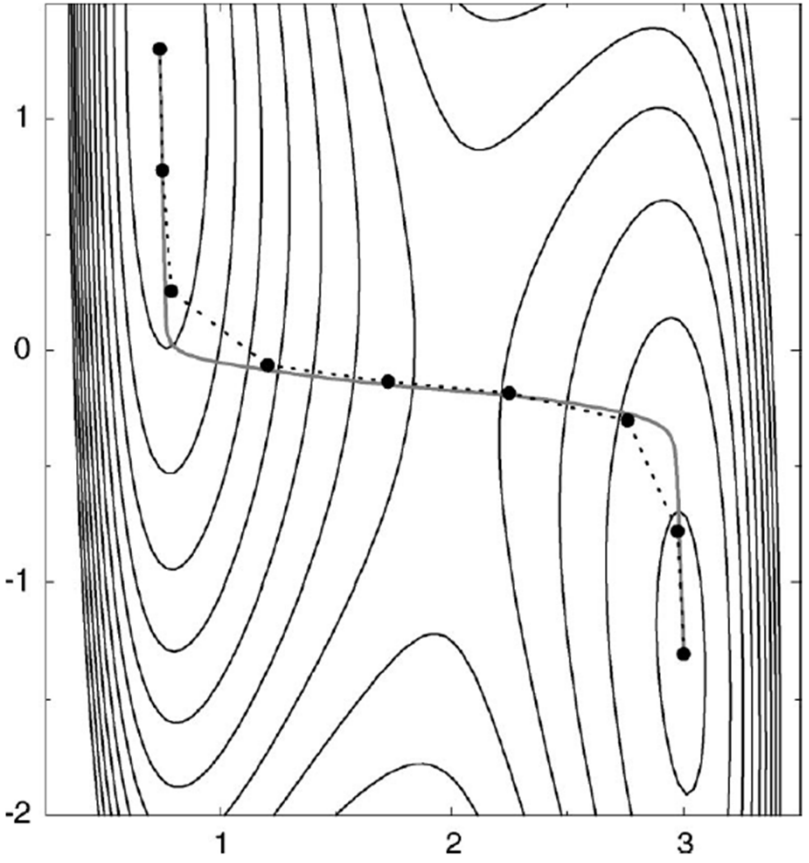
In later slides, they are referred as Refs. (A) and (B).

Nudged Elastic Band (NEB) method

The NEB method provides a way to find a minimum energy pathway (MEP) connecting two local minima.



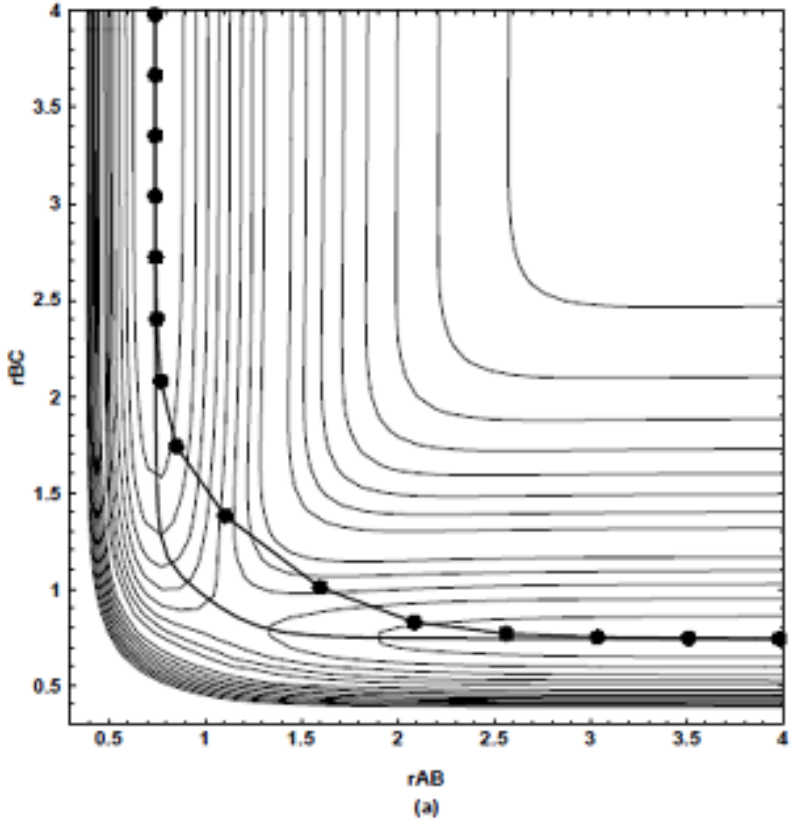
Taken from Ref. (A).



Taken from Ref. (B).

Plain Elastic Band (PEB) method

A simple idea to find a MEP is to introduce an interaction between neighboring images by a spring. The optimization of the object function *S* tries to shorten the length of MEP.



Taken from Ref. (A).

$$S(\mathbf{R}_1, \mathbf{R}_2, \cdots, \mathbf{R}_{P-1}) = \sum_{i=0}^{P} E(\mathbf{R}_i) + \sum_{i=1}^{P} \frac{Pk}{2} (\mathbf{R}_i - \mathbf{R}_{i-1})^2$$

$$\frac{\partial S(\mathbf{R}_1, \mathbf{R}_2, \cdots, \mathbf{R}_{P-1})}{\partial \mathbf{R}_k} \rightarrow 0$$

The idea is called a plain elastic band (PEB) method. However, the PEB method tends to cause a drift of energy pathway as shown in the left figure.

One should consider another way to avoid the drift of the energy pathway.

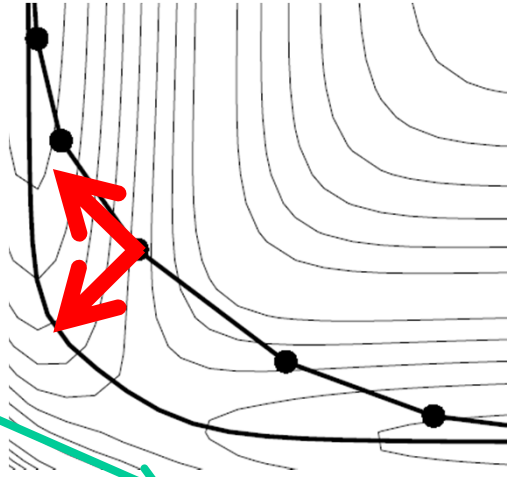
Nudged Elastic Band (NEB) method

The force can be divided to two contributions:

Parallel force

Perpendicular force

$$S(\mathbf{R}_1, \mathbf{R}_2, \cdots, \mathbf{R}_{P-1}) = \sum_{i=0}^{P} E(\mathbf{R}_i) + \sum_{i=1}^{P} \frac{Pk}{2}(\mathbf{R}_i - \mathbf{R}_{i-1})^2$$



causing non-equidistance distribution of images along the energy pathway.

$$\left(\frac{\partial E(\mathbf{R}_k)}{\partial \mathbf{R}_k}\right)_{\parallel} \quad \left(\frac{\partial E(\mathbf{R}_k)}{\partial \mathbf{R}_k}\right)_{\perp} \quad \left(\frac{\partial E_{\text{spring}}}{\partial \mathbf{R}_k}\right)_{\parallel} \quad \left(\frac{\partial E_{\text{spring}}}{\partial \mathbf{R}_k}\right)_{\perp}$$

causing the drift of energy pathway upward along the perpendicular direction.

To calculate the force, only two terms are taken into account among four contributions.

$$\mathbf{F}_k = -\left(\frac{\partial E(\mathbf{R}_k)}{\partial \mathbf{R}_k}\right)_{\perp} - \left(\frac{\partial E_{\text{spring}}}{\partial \mathbf{R}_k}\right)_{\parallel}$$

The treatment allows us to avoid the drift of energy pathway, while the physical meaning of the object function is not clear anymore.

Implementation of NEB in OpenMX

Based on Ref.[1], a nudged elastic band (NEB) method has been implemented in OpenMX. The detail of the implementation is summarized as follows:

Implementation

:

- ✓ Calc. of tangents: Eqs. (8)-(11) in Ref. (B)
- ✓ Calc. of perpendicular forces: Eq. (4) in Ref. (B)
- ✓ Calc. of parallel forces: Eq. (12) in Ref. (B)
- ✓ Optimization method: a hybrid DIIS+BFGS optimizer

In order to minimize user's efforts in using it, the functionality of NEB has been realized as one of geometry optimizers with the following features:

Features:

- ✓ Easy to use
- ✓ Hybrid OpenMP/MPI parallelization
- ✓ Initial path by the straight line or user's definition
- ✓ Only three routines added

How to perform the NEB calculation

The NEB calculation is performed by the following three steps:

1. Geometry optimization of a precursor,
2. Geometry optimization of a product,
3. Optimization of a minimum energy path (MEP) connecting the precursor and product,

where in the three calculations users have to use the same computational parameters such as unit cell, cutoff energy, basis functions, pseudopotentials, and electronic temperatures to avoid numerical inconsistency.

After the calculations 1 and 2, files *.dat# are generated. By using the atomic coordinates in the files *.dat#, one can easily construct an input file for the calculation 3. Once you have an input file for the calculation 3, the execution of the NEB calculation is same as for the conventional OpenMX calculation such as

```
mpirun –np 32 openmx input.dat –nt 4
```

Examples

Two input files are provided as examples.

C2H4_NEB.dat

Cycloaddition reaction of two ethylene molecules to cyclobutane

Si8_NEB.dat

Diffusion of an interstitial hydrogen atom in the diamond Si

The input file, C2H4_NEB.dat, will be used to illustrate the NEB calculation in the proceeding explanation.

Providing two terminal structures

The atomic coordinates of the precursor are specified in the input file by

```
<Atoms.SpeciesAndCoordinates
  1   C  -0.66829065594143   0.00000000101783  -2.19961193219289   2.0   2.0
  2   C   0.66817412917689  -0.00000000316062  -2.19961215251205   2.0   2.0
  3   H   1.24159214112072  -0.92942544650857  -2.19953308980064   0.5   0.5
  4   H   1.24159212192367   0.92942544733979  -2.19953308820323   0.5   0.5
  5   H  -1.24165800644131  -0.92944748269232  -2.19953309891389   0.5   0.5
  6   H  -1.24165801380425   0.92944749402510  -2.19953309747076   0.5   0.5
  7   C  -0.66829065113509   0.00000000341499   2.19961191775648   2.0   2.0
  8   C   0.66817411530651  -0.00000000006073   2.19961215383949   2.0   2.0
  9   H   1.24159211310925  -0.92942539308841   2.19953308889301   0.5   0.5
 10   H   1.24159212332935   0.92942539212392   2.19953308816332   0.5   0.5
 11   H  -1.24165799549343  -0.92944744948986   2.19953310195071   0.5   0.5
 12   H  -1.24165801426648   0.92944744880542   2.19953310162389   0.5   0.5
Atoms.SpeciesAndCoordinates>
```

The atomic coordinates of the product are specified in the input file by

```
<NEB.Atoms.SpeciesAndCoordinates
  1   C  -0.77755846408657  -0.00000003553856  -0.77730141035137   2.0   2.0
  2   C   0.77681707294741  -0.00000002413166  -0.77729608216595   2.0   2.0
  3   H   1.23451821718817  -0.88763832172374  -1.23464057728123   0.5   0.5
  4   H   1.23451823170776   0.88763828275851  -1.23464059022330   0.5   0.5
  5   H  -1.23506432458023  -0.88767426830774  -1.23470899088096   0.5   0.5
  6   H  -1.23506425800395   0.88767424658723  -1.23470896874564   0.5   0.5
  7   C  -0.77755854665393   0.00000000908006   0.77730136931056   2.0   2.0
  8   C   0.77681705017323  -0.00000000970885   0.77729611199476   2.0   2.0
  9   H   1.23451826851556  -0.88763828740000   1.23464060936812   0.5   0.5
 10   H   1.23451821324627   0.88763830875131   1.23464061208483   0.5   0.5
 11   H  -1.23506431230451  -0.88767430754577   1.23470894717613   0.5   0.5
 12   H  -1.23506433587007   0.88767428525317   1.23470902573029   0.5   0.5
NEB.Atoms.SpeciesAndCoordinates>
```

Note that these structures were obtained by prior geometry optimizations.

Keywords for the NEB calculation

The NEB calculation can be performed by setting the keyword, MD.Type, as

```
MD.Type                 NEB
```

The number of images in the path is given by

```
MD.NEB.Number.Images      8       # default=10
```

where the two terminals are excluded from the number of images.

The spring constant is given by

```
MD.NEB.Spring.Const     0.1       # default=0.1(hartee/borh^2)
```

In most cases, the obtained path does not largely depend on the value.

The optimization of MEP is performed by a hybrid DIIS+BFGS scheme which is controlled by the following keywords:

```
MD.Opt.DIIS.History       4       # default=7
MD.Opt.StartDIIS          10      # default=5
MD.maxIter                100     # default=1
MD.Opt.criterion          1.0e-4  # default=1.0e-4 (Hartree/bohr)
```

The specification of these keywords are same as for the geometry optimization.
So, see the manual for the details.

Execution of the NEB calculation

One can perform the NEB calculation for C2H4_NEB.dat by

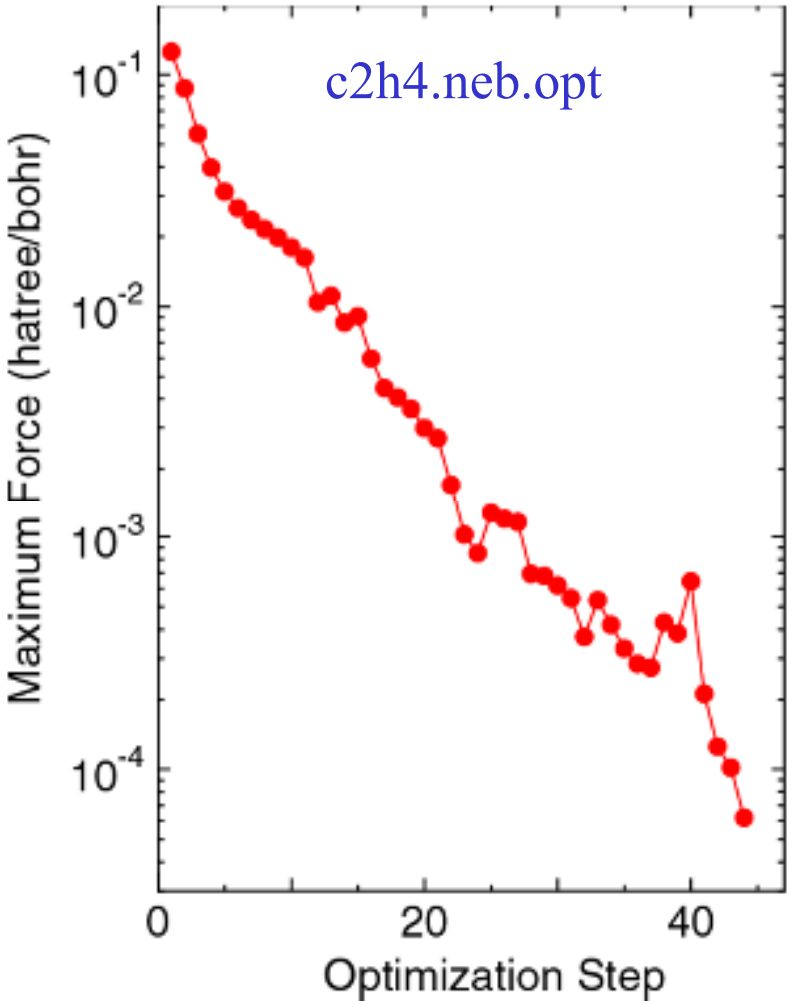
```
mpirun –np 16 openmx C2H4_NEB.dat
```

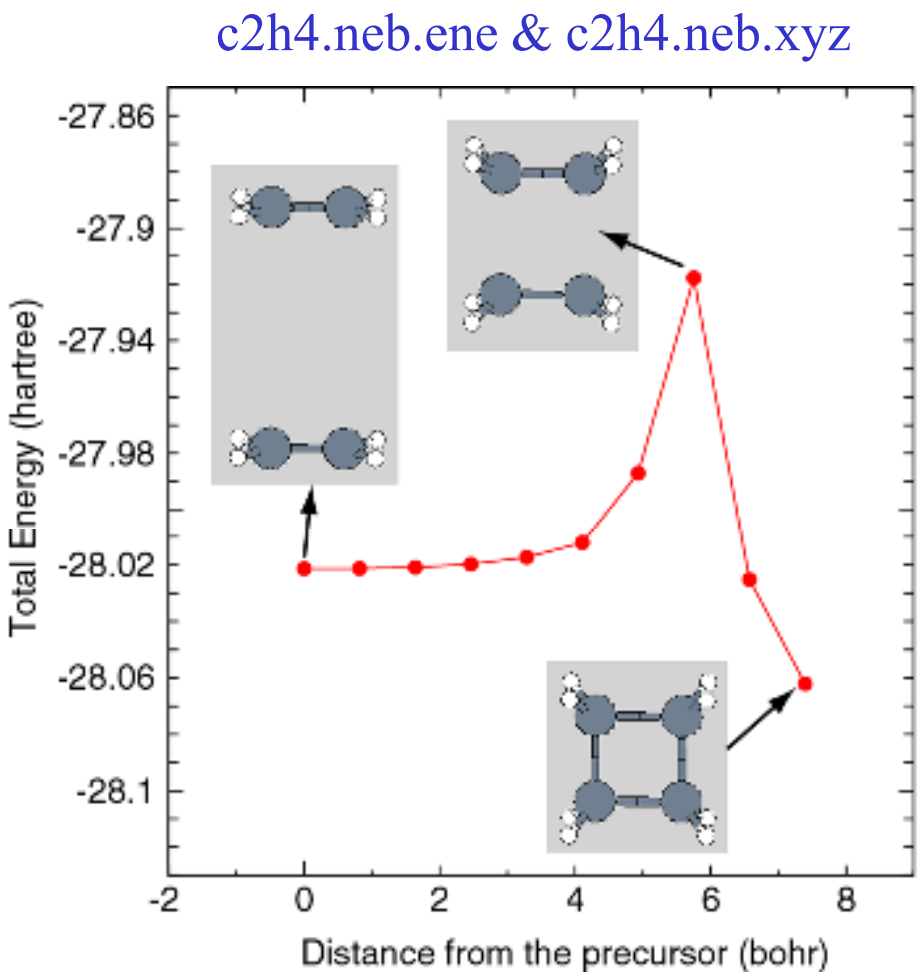
If the calculation successfully completed, the following four files are generated.

c2h4.neb.opt	history of optimization for finding MEP
c2h4.neb.ene	total energy of each image
c2h4.neb.xyz	atomic coordinates of each image in XYZ format
C2H4_NEB.dat#	input file for restarting

c2h4.neb.opt, c2h4.neb.ene, and c2h4.neb.xyz

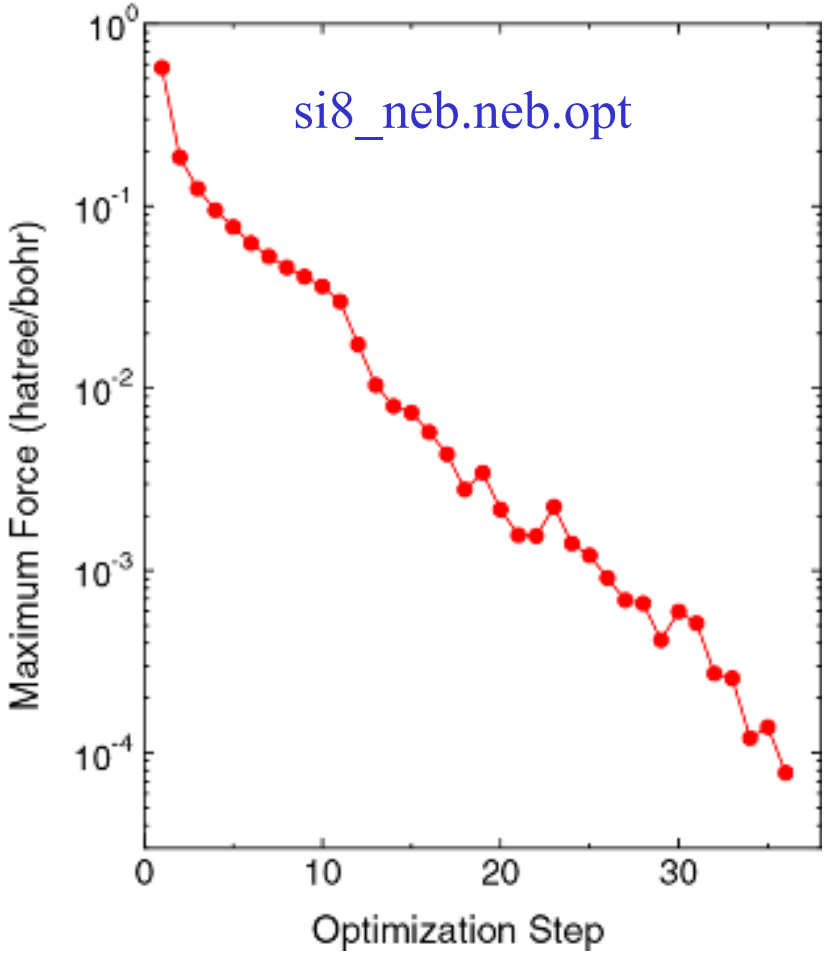
They can be used to analyze MEP as follows:



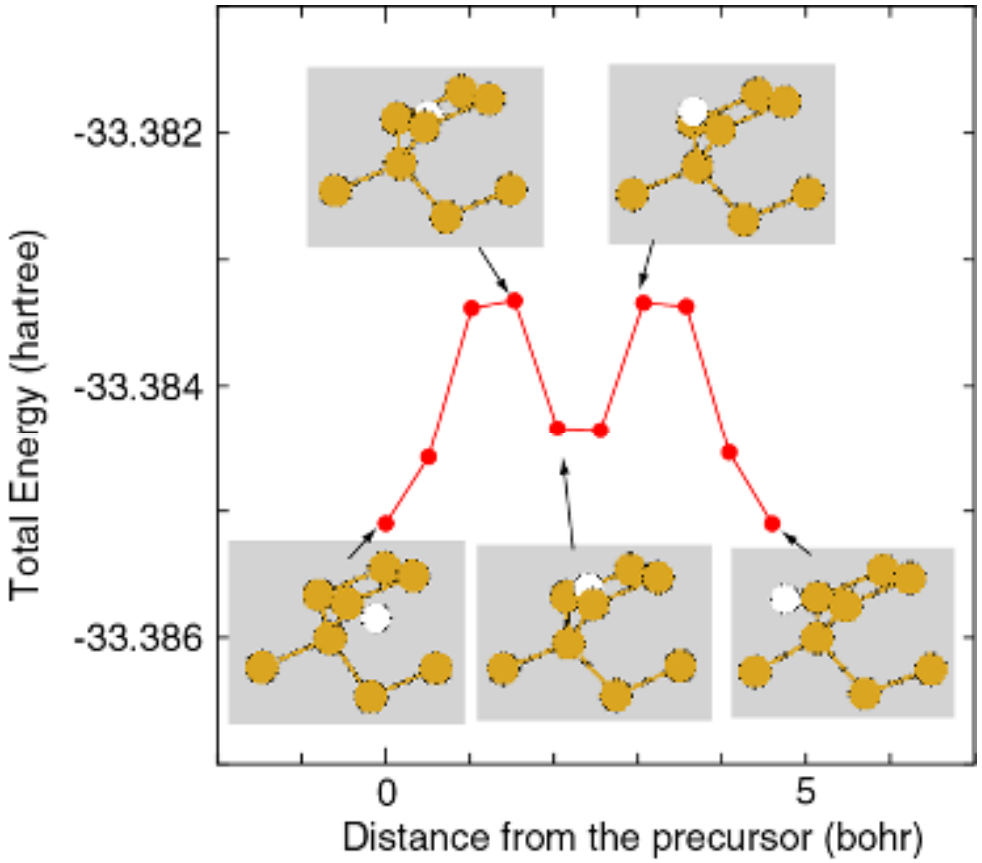


In case of Si8_NEB.dat

As well as the case of C2H4_NEB.dat, one can perform the NEB calculation by Si8_NEB.dat. After the successful calculation, the following results are obtained.



si8_neb.neb.ene & si8_neb.neb.xyz



Restarting the NEB calculation

It often happens that the convergence is not achieved even after the maximum optimization step. In such a case, one has to continue the optimization as a new job starting from the last optimization step in the previous job. A file, *.dat#, is generated after every optimization step. The file contains a series of atomic coordinates for images in the last step. Using *.dat# one can restart the optimization.

User defined initial path

As default, the initial path connecting the precursor and the product is a straight line connecting them. However, in some cases the geometrical structure of images generated on the straight line can be very erratic so that distance between atoms can be too close to each other. In this case, one should explicitly provide the atomic coordinates of images. The user defined initial path can be provided by the same way as for the restarting. Then, one has to provide atomic coordinates for each image by the following keywords:

```
<NEB1.Atoms.SpeciesAndCoordinates
  1   Si   -0.12960866043083    0.13490502997627   -0.12924862991035     2.0     2.0
  2   Si   -0.40252421446808    5.19664433048606    4.91248322056082     2.0     2.0
  …..
NEB1.Atoms.SpeciesAndCoordinates>

<NEB2.Atoms.SpeciesAndCoordinates
  1   Si   -0.08436294149342   -0.02173837971883   -0.08374099211565     2.0     2.0
  2   Si   -0.33677725120015    5.10216241168093    5.01087499461541     2.0     2.0
  ……
NEB2.Atoms.SpeciesAndCoordinates>
```

For all the images of which number is given by MD.NEB.Number.Images, the atomic coordinates need to be provided.
Also, it is required for a keyword to be switched on as

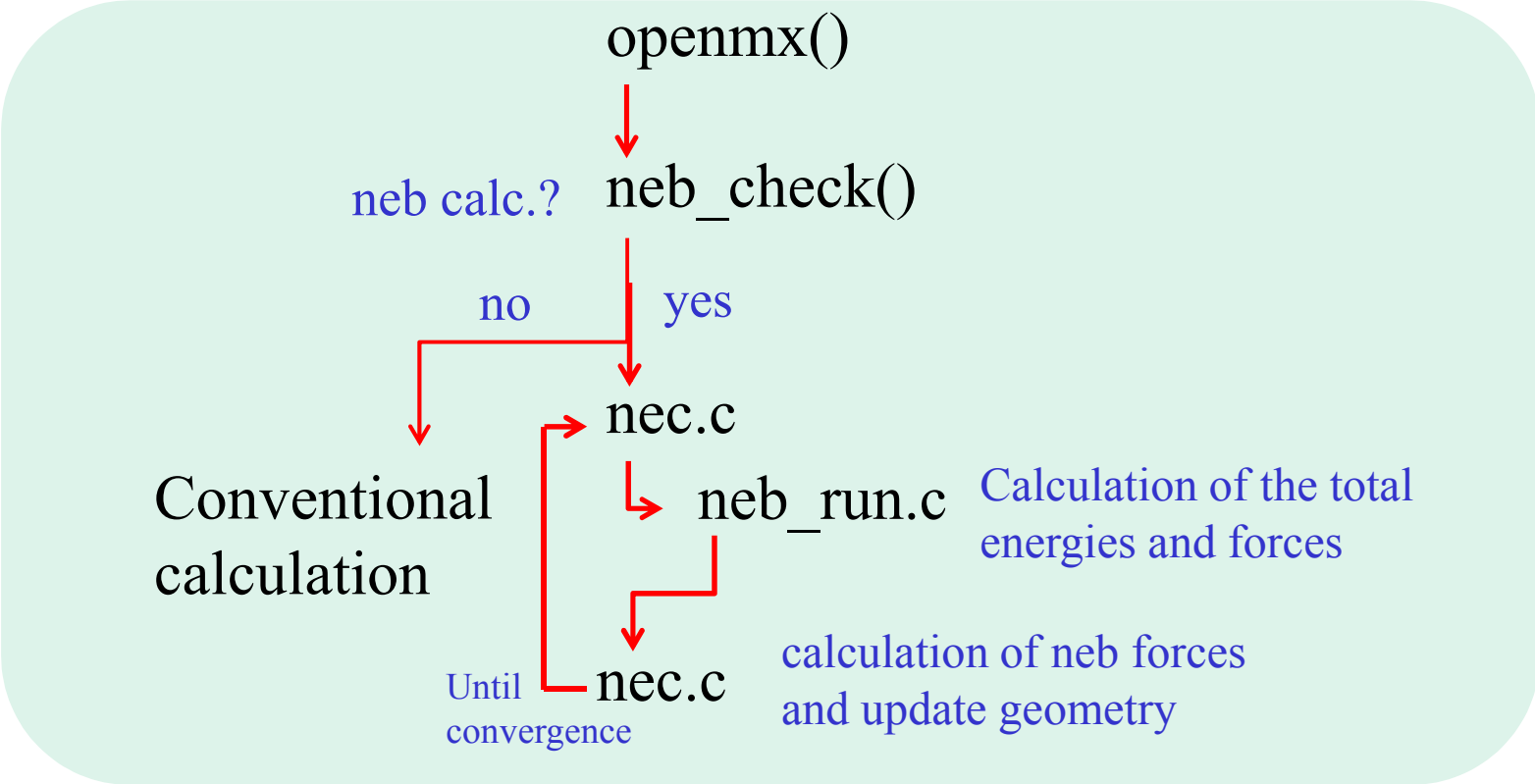
scf.restart on

Monitoring the NEB calculation

In the NEB calculation, the standard output will display only that for the image 1, and those for the other images will not be displayed. However, there is no guarantee that the SCF iteration converges for all the images. In order to monitor the SCF convergence for all the images, temporary files can be checked by users. In the NEB calculation, an input file is generated for each image, whose name is *.dat_#, where # runs from 0 to MD.NEB.Number.Images+1, and ‘system.name’ is modified as the original system.name_#. So, one can check the SCF convergence by monitoring system.name.DFTSCF whether it converges or not.

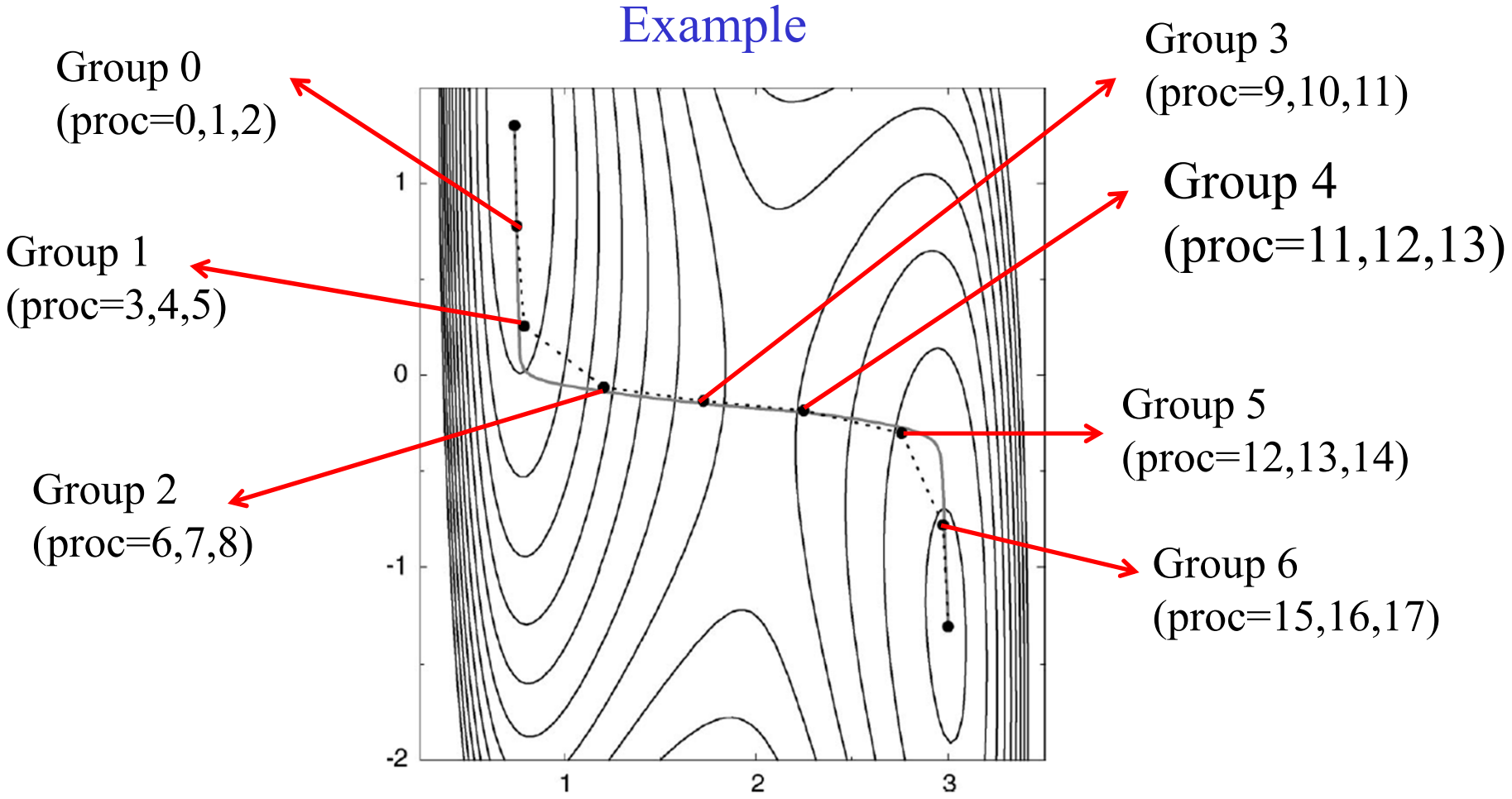
Relevant routines

Only three routines are added to implement the NEB functionality. They are neb.c, neb_run.c, and neb_check.c. The main routine is neb.c. It may be easy to implement related methods by employing the structure of neb.c.



Parallelization

Example



MPI communication groups are generated in nec(), and the calculation of each image is performed with each group in neg_run().

Close look at code: call neb_check() from openmx()

In openmx()

```
/*******************************************************
  check the NEB calculation or not, and if yes, go to
  the NEB calculation.
*******************************************************/

if (neb_check(argv)) neb(argc,argv);

/*******************************************************
 allocation of CompTime and show the greeting message
*******************************************************/

CompTime = (double**)malloc(sizeof(double*)*numprocs);
for (i=0; i<numprocs; i++){
  CompTime[i] = (double*)malloc(sizeof(double)*20);
```

Check the NEB calculation or not.

In neb_check()

```
int neb_check(char *argv[])
{
  int i,j,flag;
  char *s_vec[40];
  int i_vec[40];

  if (input_open(argv[1])==0){
    MPI_Finalize();
    exit(0);
  }

  i=0;
  s_vec[i]="NOMD";                i_vec[i]=0;  i++;
  s_vec[i]="NVE" ;                i_vec[i]=1;  i++;
  s_vec[i]="NVT_VS";              i_vec[i]=2;  i++; /* modifS
  s_vec[i]="OPT";                 i_vec[i]=3;  i++;
  s_vec[i]="EF";                  i_vec[i]=4;  i++;
  s_vec[i]="BFGS";                i_vec[i]=5;  i++;
  s_vec[i]="RF";                  i_vec[i]=6;  i++; /* RF ms
  s_vec[i]="DIIS";                i_vec[i]=7;  i++;
  s_vec[i]="Constraint_DIIS";     i_vec[i]=8;  i++; /* not u
  s_vec[i]="NVT_NH";              i_vec[i]=9;  i++;
  s_vec[i]="Opt_LBFGS";           i_vec[i]=10; i++;
  s_vec[i]="NVT_VS2";             i_vec[i]=11; i++; /* modifS
  s_vec[i]="EvsLC";               i_vec[i]=12; i++;
  s_vec[i]="NEB";                 i_vec[i]=13; i++;

  s_vec[i]="NVT_NH";              i_vec[i]=9;  i++;
  s_vec[i]="Opt_LBFGS";           i_vec[i]=10; i++;
  s_vec[i]="NVT_VS2";             i_vec[i]=11; i++; /* mod:
  s_vec[i]="EvsLC";               i_vec[i]=12; i++;
  s_vec[i]="NEB";                 i_vec[i]=13; i++;

  j = input_string2int("MD.Type",&MD_switch, i, s_vec,i_vec);
  if (j==-1){
    MPI_Finalize();
    exit(0);
  }

  input_close();

  flag = 0;
  if (MD_switch==13) flag = 1;

  return flag;
}
```

If the NEB calculation, return 1.

Close look at code: Generation of MPI comm groups in neb()

If the total number of processes is larger than the number of images, generate MPI comm groups.

```
/*****************************************************
   allocate processes to each image in the World1
*****************************************************/

Num_Comm_World1 = NEB_Num_Images;

if ( Num_Comm_World1<=numprocs ){

  parallel1_flag = 1;

  NPROCS_ID1 = (int*)malloc(sizeof(int)*numprocs);
  Comm_World1 = (int*)malloc(sizeof(int)*numprocs);
  NPROCS_WD1 = (int*)malloc(sizeof(int)*Num_Comm_World1);
  Comm_World_StartID1 = (int*)malloc(sizeof(int)*Num_Comm_World1);
  MPI_CommWD1 = (MPI_Comm*)malloc(sizeof(MPI_Comm)*Num_Comm_World1);

  Make_Comm_Worlds(MPI_COMM_WORLD, myid, numprocs, Num_Comm_World1, &myworld1, MPI_CommWD1,
                   NPROCS_ID1, Comm_World1, NPROCS_WD1, Comm_World_StartID1);

  MPI_Comm_size(MPI_CommWD1[myworld1],&numprocs1);
  MPI_Comm_rank(MPI_CommWD1[myworld1],&myid1);

}
else {
  parallel1_flag = 0;
  myworld1 = 0;
}
```

Identifiers of generated MPI comm world are stored.

Index of generated MPI comm world are stored.

Close look at code:

Calculation of total energy and forces of each image in neb()

```
/*****************************************************
   optimization for finding a minimum energy path
   connecting the two terminal structures.
*****************************************************/

iter = 1;
MD_Opt_OK = 0;

do {

  /* generate input files */

  generate_input_files(fname_original,iter);

  /* In case of parallel1_mode==1 */

  if (parallel1_flag==1){

    sprintf(fname1,"%s_%i",fname_original,myworld1+1);
    argv[1] = fname1;

    neb_run( argv,MPI_CommWD1[myworld1],myworld1+1,neb_atom_coordi$
             WhatSpecies_NEB,Spe_WhatAtom_NEB,SpeName_NEB );

    /* MPI: All_Grid_Origin */

    for (i=0; i<=(NEB_Num_Images+1); i++){
      for (j=0; j<=3; j++) Tmp_Grid_Origin[i][j] = 0.0;
    }

    if (myid1==Host_ID){
      Tmp_Grid_Origin[myworld1+1][1] = Grid_Origin[1];
      Tmp_Grid_Origin[myworld1+1][2] = Grid_Origin[2];
      Tmp_Grid_Origin[myworld1+1][3] = Grid_Origin[3];
    }

    MPI_Barrier(MPI_COMM_WORLD);

    for (p=1; p<=NEB_Num_Images; p++){
      MPI_Allreduce( &Tmp_Grid_Origin[p][0], &All_Grid_Origin[p][0$
                     4, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD );
    }

    /* MPI: neb_atom_coordinates */

    /* MPI: neb_atom_coordinates */

    for (p=1; p<=NEB_Num_Images; p++){
      for (i=0; i<=atomnum; i++){
        for (j=0; j<20; j++){
          tmp_neb_atom_coordinates[p][i][j] = 0.0;
        }
      }
    }

    if (myid1==Host_ID){
      for (i=0; i<=atomnum; i++){
        for (j=0; j<20; j++){
          tmp_neb_atom_coordinates[myworld1+1][i][j] = neb_atom$
        }
      }
    }

    MPI_Barrier(MPI_COMM_WORLD);

    for (p=1; p<=NEB_Num_Images; p++){
      for (i=0; i<=atomnum; i++){

        MPI_Allreduce( &tmp_neb_atom_coordinates[p][i][0], &neb$
                       20, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD $

      }
    }

  } /* if (parallel1_flag==1) */

  /* In case of parallel1_mode==0 */

  else {

    for (p=1; p<=NEB_Num_Images; p++){

      sprintf(fname1,"%s_%i",fname_original,p);
      argv[1] = fname1;
      neb_run( argv,MPI_COMM_WORLD1,p,neb_atom_coordinates,
               WhatSpecies_NEB,Spe_WhatAtom_NEB,SpeName_NEB );

      /* store Grid_Origin */
      All_Grid_Origin[p][1] = Grid_Origin[1];
```

Input files for all the images are generated.

Calculation of total energy and forces

Close look at code:

Calculation of NEB forces and update coordinates in neb()

```
  /* calculate the gradients defined by the NEB method */
  Calc_NEB_Gradients(neb_atom_coordinates);
  /* make a xyz file storing a set of structures of images at the current step */
  Make_XYZ_File(fileXYZ,neb_atom_coordinates);
  /* update the atomic structures of the images */
  Update_Coordinates(iter,neb_atom_coordinates);
  /* generate an input file for restarting */
  Generate_Restart_File(fname_original,neb_atom_coordinates);
  /* increment of iter */
  iter++;
} while (MD_Opt_OK==0 && iter<=MD_IterNumber);
```

Close look at code:
neb_run()

The basic structure of neb_run() is similar to that of openmx().

```
/* initialize */

init();

/* for DFTD-vdW by okuno */
if(dftD_switch==1) DFTDvdW_init();

/*******************************************************
                    SCF-DFT calculations
*******************************************************/

MD_iter = 1;

CompTime[myid][2] += truncation(MD_iter,Solver==6,1);
CompTime[myid][3] += DFT(MD_iter,(MD_iter-1)%orbitalOpt_per_MDIter+1);

/*******************************************************
 store the total energy, coordinates, and gradients
*******************************************************/

/* total energy */
neb_atom_coordinates[index_images][0][0] = Utot;
```