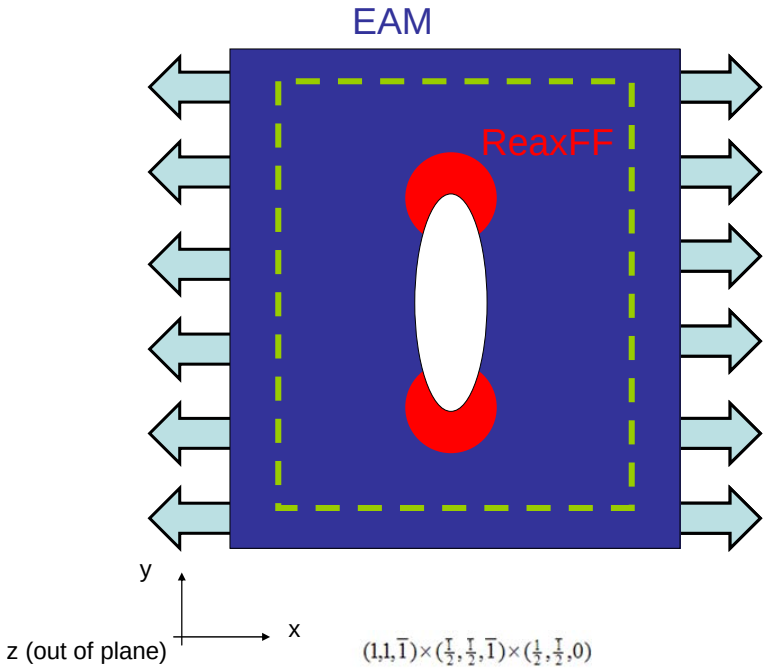
PROBLEM B – Fracture of a single crystal of silicon

This problem set utilizes a new simulation method based on Computational Materials Design Facility (CMDF) to model fracture in a brittle material, silicon.

The CMDF enables easy combination of various computational tools in a Python scripting environment. CMDF features many potentials, simulation methods and analysis tools. Here we focus solely on the methods related to coupling **ReaxFF and Tersoff** to describe fracture in silicon; in a geometry as shown below (reactive regions are determined dynamically, here only shown schematically):



Unlike the previous problem set A (for the FCC nanowire/crystal) where it was sufficient to fill out forms on the website, CMDF requires a complete simulation script as input. The simulation script provides a detailed description of the various steps and their sequence, including (i) creation of silicon crystal, (ii) addition of crack, (iii) application of strain, (iv) and solving the dynamical equations.

On the MIT server, we have provided a first input script that performs a simple CMDF simulation of fracture of silicon (Si_fracture.py). Critical parameters for the simulation are summarized in the beginning:

```
# IMPORTANT:  Simulation parameters
…
```

Please read the papers and slides put on the MIT server; those provide background information about CMDF, ReaxFF and the Hybrid Hamiltonian method. Further, please consider

consulting one of the many Python manuals that can be found online. It will not be necessary to do extensive Python coding; simple changes to the above mentioned section ("`# IMPORTANT ...`") should be sufficient.

As a first step, run the example script and perform an analysis. It should take approximately 2.5 hours. Results and hints are summarized in the CMDF tutorial on the MIT server.

GenePattern website: http://starapp.mit.edu:8083/gp/index.jsp

Notes re. GenePattern website: Please select the option "CMDF" in the section "Tasks". The only file to be uploaded is the CMDF Python script. For easy editing, you may want to install python for Windows (www.python.org) – this will provide you with syntax highlighting. If you have access to a Linux or Unix box, you can use "nedit", "emacs" or "vi".

Note: Please consider the lecture notes posted in the "Lecture 8" section, in particular the documents re. CMDF and mixed/hybrid Hamiltonians.

Manuals/tutorials: http://wiki.python.org/moin/BeginnersGuide (others on www.python.org)

B.1 Mixed Hamiltonians

The idea implemented in CMDF is to use multiple force fields simultaneously, to capture critical quantum chemical effects in regions where it is necessary (crack tip, during reactions, formation/breaking of bonds..), but rely on computationally less demanding methods when possible without sacrificing on the essential physics.

Such hybrid methods have been very successful in describing fracture dynamics of silicon, where empirical approaches alone (e.g. pure Tersoff potential, Stillinger-Weber etc.) have largely failed.

1. In CMDF, we have approximated mixed Hamiltonians by mixing force contributions of different force fields:

$$F_{\mathrm{ReaxFF}-EAM} = \left(w_{\mathrm{ReaxFF}}(x)F_{\mathrm{ReaxFF}} + (1-w_{\mathrm{ReaxFF}})F_{\mathrm{EAM}}\right).$$

   Justify this expression by considering how force contributions are calculated from taking derivatives of the total Hamiltonian.
   Specify the conditions that need to be satisfied for this equation to hold, and use simple graphs to illustrate your point.
2. Estimate a value for the ghost atom region (R_{buf}) based on potential properties (for simplicity, assume a simple LJ potential). Include a schematic to illustrate how you estimate this length scale.
   What values would be reasonable for the Tersoff and ReaxFF potential (see

potential properties below)?
How may this criterion different for organic systems (e.g. proteins) versus metallic systems?

3. An essential component of using mixed Hamiltonians is to find computational domains for each method used in the scheme. For the crack problem in silicon, describe possible methods to determine regions to be described by ReaxFF. What are the essential features?
4. Are sinusoidally varying weight functions a better choice? Discuss why, e.g. using some equations or using schematics.

B.2 Crack speed and fracture dynamics – Mode I

1. Assuming that the Griffith condition describes the critical condition for crack initiation, predict the critical strain for failure onset.
   Hint: First consider the stress intensity factor for the system (see below), then use the equivalence between stress intensity factor and energy release rate *G* to express the critical condition for fracture initiation.
   You may estimate the crack length from measuring the distance between the crack tip in VMD (go to Mouse→Query; then select ..-->Label→Atoms selecting any atom will print its position in the terminal window that is opened with VMD – the black window; similarly, you can measure distances by clicking on …→Bonds).
2. Write out clearly what the time step is, initial static strain, total simulation time and strain rate, for each simulation you do.
   Start with the example script provided. Compare the initial strain used in the script with the prediction by Griffith theory.
   Note: You will likely have to increase the total number of simulation steps to observe fracture of the entire crystal, consider how long a simulation takes (1,000 steps ~ 2.5 hours).
3. Describe qualitatively what kind of fracture dynamics and fracture mechanisms occur.
   Provide a detailed analysis of the fracture process, use snapshots in VMD to illustrate your point.
   Try to take advantage of the computational microscope you have at hand, and analyze how bonds break, rearrange and how these processes relate to the particular fracture surface you observe.
4. Using the web based tool, calculate the speed of the crack as a function of time. Note that you can infer about the current time by considering the frequency of snapshot writing (parameter *iprint*) and the time step used in the simulation.
5. Theoretically, the maximum speed of cracks is related closely to the speed of sound. For example, for mode I tensile loaded cracks the maximum speed of a crack is given by the Rayleigh wave speed (speed of surface elastic waves). Discuss the observed crack speed with respect to this. For mode II, the limiting speed of cracks has been shown to be the longitudinal wave speed.
6. Increase the initial strain from the example script to 8% and observe the difference in fracture mechanics. Does the crack speed become larger?

B.3 Shear loading – Mode II

1. Change the input script and implement shear loading. How does fracture mechanics change compared to the previous loading under mode I? Describe the fracture processes qualitatively.
   Note: To implement shear loading, set the initial tensile strain sx to a smaller value, say 1.01.
   Then set the parameter "shear" to 0.001, for instance.
   Play with different strain rates until you observe crack nucleation within a few thousand steps of simulation.
2. Estimate the speed of the crack under shear loading. How does this relate to the prediction that the longitudinal wave speed is the limiting speed for mode II cracks?

B.4 References and material constants

Wave speed in silicon: The Rayleigh-wave speeds is 4.68 km/sec for the (111) orientation.

Fracture surface energy of silicon: $\gamma_{0,(111)} \approx 1.403$ N/m

Elastic modulus: $E'_{(111)} \approx 2.46 \times 10^{11}$ Pa

Stress intensity factor for geometry considered:

Image removed due to copyright restrictions.
Simple illustration.

$$K_I = \sigma\sqrt{\pi a}$$

$$G = \frac{K_I^2}{E'_{(111)}}$$

Cutoff radii: Tersoff potential: 2.1 Angstrom, ReaxFF potential: 5 Angstrom.

Python script

This representation was obtained by using the IDLE you can download on the Python website; it can be installed by getting Python for Windows.

```
############## Tools ##############
import modbabel
import bgf_reader
import time
from Si110 import *
import cracks
import PLACE
############## Tersoff ##############
import os
import IMD_Tersoff
import IMDTOOLS
############## REAX ##############
import modreax
import REAXTOOLS
###
############## Integrator ##############
#
import ModDynamics
import ModDynamicsCPP
###
import ModMulti
import ModMultiCPP
#
####################################

# Units:  Angstrom, eV, amu
# Dimensions of crystal, in the x, y and z direction
Nx      = 25
Ny      = 40
Nz      = 1

# Temperature (it will be controlled by an NVT scheme)
# This temperature is also the initial temperature
CTemp   = 300.

# Parameter for detection of atoms that are embedded in reactive domain
Ecrit   = -8.0

# Parameters for the mixed Hamiltonian approach
# RRR: Radius of reactive region, triggered by each atom with E>Ecrit
RRR     = 5.00
# a1:  Width of transition region, a2: Width of ghost atom region
a1      = 5.0
a2      = 5.0

# Geometry of the initial ellipsoidal crack (penny-shaped crack)
AA=8.
BB=20.

# IMPORTANT:  Simulation parameters
i               = 0         # Initial step
isteps          = 100      # How many steps total
ifreq           = 50       # How often is geometry (.xyz) file written
itemp           = 10        # How often is temperature controlled (Berendson)
iregionupdate   = 20        # How often is reactive region updated
iprintff        = 5         # How often is temperature, energy etc. printed
istr            = 10        # How often is strain increased by sx and shear
sx              = 1.00      # Increments of tensile strain (note: sx=1+epsilon_xx)
shear           = 0.0000    # Increments of shear strain (note: shear=epsilon_xy, i.e. shear=0.001 is a possibl
deltat          = 0.150000  # Time step in 1.018E-14 seconds
```

```
# Initial, static strain.  Load is applied in the x-direction for mode I loading
# sx = 1+epsilon_xx
sx=1.06
sy=1.00
sz=1.00



############################## Below:  Simulation description

# Create an initial geometry (OBtot) with particular cell size (cell)
[OBtot, cell] = XTL (Nx, Ny, Nz,  0, 'Si111.bgf', 6.6626, 9.42235, 3.84666,  12)  # good

print "Cell parameter before straining ", cell

# Strain the initial geometry according to sx, sy, sz
ModMultiCPP.StrainOBMol(OBtot, sx, sy, sz)

# Move entire crystal a bit to the right and top
ModMulti.TranslateOBMol (OBtot, 2.0, 5.0 ,0.0)

cell[0]*=sx;
cell[1]*=sy;
cell[2]*=sz;

cellx=cell[0]
celly=cell[1]
cellz=cell[2]

# Cut out initial ellipsoidal crack
OBtot= ModMulti.CutEllipse (OBtot, cellx/2., celly/2., cellz/2., AA,BB)

print "Cell parameter after straining ", cell

# Determine number of atoms in the system
natoms=OBtot.NumAtoms ()

print "Have %d atoms total " % natoms


# Here, the BCs are set (which atoms are fixed etc.)
# Atoms at a boundary on the left and right (10 A wide) are held fixed.  They
# can only move due to the externally described displacemements
# and do not follow F=ma.  This allows to apply a displacement
# boundary condition
i=1
while i<=natoms:

       atom=OBtot.GetAtom (i)
       atom.SetGID (i)
       atom.SetFixed (0, 0, 0)

       # Fixed the boundary atoms
       if (atom.GetX()<10.):
                atom.SetFixed (1, 1, 1)

       if (atom.GetX()>(cellx-10.)):
                atom.SetFixed (1, 1, 1)
```

```
      i=i+1

ymov=5.
ModMulti.TranslateOBMol (OBtot, ymov , 0.,  0.)

# Initialize the temperature - initial velocities from a MB distribution
ModDynamics.InitTemp_MB (OBtot, CTemp)

# -------------------------------------
# Setup for the handshaking (not critical for running the code)
ModMulti.init_weights (OBtot, 2)
OB2=ModMulti.CopyOBMol (OBtot)
xtip=cellx/2.
ytip=celly/3
ztip=cellz/2.
cellxo=cellx
cellyo=celly
celly=celly+20.0
cellx=cellx+20.0
xssa    =[30., (cellxo-30.), 30., cellyo-40., -1, 100.]
print "Search region crack ", xssa
# -------------------------------------

# Set up force fields:  1) Tersoff and 2) ReaxFF
IMD_Tersoff.SetupFF_Tersoff (1, 1, 1, cellx, celly, cellz, 1,1,1, "TrTERSOFF")
REAXTOOLS.SetupFF ('ffield', 343434)
modreax.pyinit (cellx, celly+50, cellz, 90, 90, 90)

# Test the force fields by doing a single point energy evaluation
IMDTOOLS.OBToPyIMD (OB2, IMD_Tersoff)
IMD_Tersoff.GetForces ()
IMD_Tersoff.PrintEnergies ()
IMDTOOLS.IMDForcesToOBMol (OB2, IMD_Tersoff)

# -------------------------------------
# Determine reactive region based on atomic energy Ecrit
xtip=0
ytip=0
ztip=0
[xtip,ytip,ztip]=ModMulti.FindCrackTip (OB2, xssa[0], xssa[1], xssa[2], xssa[3], xssa[4], xssa[5], xtip, ytip,
ModMulti.CopyPE (OBtot, OB2)
ModMultiCPP.find_reg (OBtot, 30., cellxo-30, 30., cellyo-30, -1., 1000.,  RRR, Ecrit,a1, a2, 1, 2)
# -------------------------------------

# -------------------------------------
# Set up domain decomposition scheme
sys=ModMultiCPP.divideSingleEngine (OBtot,1)
OB1=sys.GetMol (0)
OB2=ModMulti.CopyOBMol (OBtot)
# -------------------------------------

# Dump files
ModMulti.dumpOBMol (OB1,        'reax')
ModMulti.dumpOBMol (OB2,        'tersoff')
ModMulti.dumpOBMol (OBtot,      'all')
```

```
# ------------------------------------
# Test ReaxFF module for initial domain
REAXTOOLS.REAXForces (OB1, modreax, cellx, celly, cellz)
# ------------------------------------


######################## now start dynamics ################
print "Start dynamics "
print "TIMING:  ITERATION INTEGR_PART_1    REAX    IMD    FORCE_COMB    INTEGR_PART_2     XYZ_UPDATE  IMD_UPDATE

# "BIG" MD loop over all time steps
while (i<isteps):

        t0=time.clock()

        ModDynamicsCPP.IntegrateNVE1 ( OBtot, deltat, cellx, celly, cellz, 1,1,1)
        t11=time.clock()

        sys = OBSystem()
        sys.AddMol(OB1)
        sys.AddMol(OB2)
        ModMultiCPP.xyzupdate ( OBtot, sys )

        t111=time.clock()

        IMD_Tersoff.UpdateIMD (OB2)

        t1=time.clock()

        REAXTOOLS.REAXForces (OB1, modreax, cellx, celly, cellz)
        t2=time.clock()

        IMD_Tersoff.GetForces ()
        IMD_Tersoff.ForcesToOBMol (OB2)

        t3=time.clock()

        sys = OBSystem()
        sys.AddMol(OB1)
        sys.AddMol(OB2)
        ModMultiCPP.forcecombine ( OBtot, sys, 1) # 1=delete old forces

        t4=time.clock()
        ModDynamicsCPP.IntegrateNVE2 ( OBtot, deltat)
        t5=time.clock()

        # Apply strain
        if ( (i%istr)==0):
                ModMultiCPP.StrainOBMol(OBtot, sx, sy, sz)
                ModMulti.ShearOBMol(OBtot, shear, cellxo/2.)

        if ( (i%itemp)==0):
                print "Temperature (K)=", ModDynamicsCPP.GetTemp (OBtot)
                ModDynamicsCPP.ScaleVelFac ( OBtot, CTemp, 0.3)  # Note: 0.3 is the Berendson parameter

        t6=time.clock()
```

```
        # Write output files
        if ( (i%ifreq)==0):
           ##
           char = "results/all.%5.5d.xyz" % (i/ifreq)
           ModMulti.dumpOBMolToXYZ (OBtot, char)

           char = "results/filter.%5.5d.xyz" % (i/ifreq)
           ModMulti.dumpOBMolToXYZFilter (OB2, char, Ecrit)

           char = "results/reax.%5.5d.xyz" % (i/ifreq)
           ModMulti.dumpOBMolToXYZ (OB1, char)

           char = "results/all.%5.5d" % (i/ifreq)
           ModMulti.dumpOBMol (OBtot, char)
           char = "results/reax.%5.5d" % (i/ifreq)
           ModMulti.dumpOBMol (OB1, char)
           char = "results/tersoff.%5.5d" % (i/ifreq)
           ModMulti.dumpOBMol (OB2, char)

           print "Temperature %7.3f " % ModDynamicsCPP.GetTemp (OBtot)


        if (( (i%iregionupdate)==0) & (i>=0)):
          t100=time.clock()
          ModMulti.init_weights (OBtot, 2)
          print "Update simulation regions (depending on crack tip position)"
          #[xtip,ytip,ztip]=ModMulti.FindCrackTip (OB2, xssa[0], xssa[1], xssa[2], xssa[3], xssa[4], xssa[5], x
          [xtip,ytip,ztip]=ModMulti.FindCrackTipSEnergy (OB2, xssa[0], xssa[1], xssa[2], xssa[3], xssa[4], xssa

          print "ITER  %5.5d  CRACK_TIP  %5.3f %5.3f %5.3f " % (i,  xtip,ytip,ztip)
          #ModMultiCPP.assignsphere_weights (OBtot, xtip, ytip, ztip, RRR, a1, a2, 1, 2)
          ModMulti.CopyPE (OBtot, OB2)

          ModMultiCPP.find_reg (OBtot, 30., cellxo-30, 30., cellyo-30, -1., 1000.,  RRR, Ecrit,a1, a2, 1, 2)
          ModMultiCPP.find_reg_atype_smooth (OBtot, 30., cellxo-10, 30, cellyo-30, -10, 10000, RRR, 8,  a1, a2,
          ModMultiCPP.find_reg_atype_smooth (OBtot, 30., cellxo-10, 30, cellyo-30, -10, 10000, RRR, 1,  a1, a2,

          sys.DeleteMol (sys.GetMol (0) )

          sys=ModMultiCPP.divideSingleEngine (OBtot,1)
          OB1=sys.GetMol (0)
          OB2=ModMulti.CopyOBMol (OBtot)
          t101=time.clock()
          print "*** Time to update the regions %5.3f" % ( t101-t100 )


        if (i%iprintff)==0:
           print "TIMING: %5.5d  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f %5.3f" % ( i, t11-t0, t

           IMD_Tersoff.PrintEnergies ()

           print "*** step ", i, "Temperature (K)=", ModDynamicsCPP.GetTemp (OBtot), " ReaxFF/Tersoff atom numb

        i=i+1

######################## end dynamics ################
```