一、Prepare files

1. input folder

```
INCAR_relax  # 结构弛豫用到的INCAR文件
vasp.new  # 子任务的提交脚本
```

2. origin folder

```
#被切面的bulk结构
structure1.vasp  #结构需要以.vasp结尾，且是poscar格式
structure2.vasp  #可以放置任意数量结构
...
```

3. calculate scripts

```
surslab.1.py    #切面主程序
calculate.py    #计算表面能程序
main.lsf    #提交主程序surslab.1.py的脚本
```

二、开始计算●

```
$ bsub<main.lsf
计算的顺序是：首先提交main.lsf至服务器，main.lsf会在服务器中运行surslab.1.py
上述计算结束后再自己运行calculate.py
$ ./calculate.py
```

三、脚本解释

1. main.lsf

```
#!/bin/bash
```

```
#BSUB -J slabs
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -q oldibm
#BSUB -n 14
##BSUB -R "span[host=1]"

# source /gpfs/compiler/parallel_studio_xe_2017.5.061/bin/psxevars.sh >/dev/null 2>&1
### 上述为提交任务通用语句
./surslab.1.py > result.log
### 在服务器端运行surslab.1.py，并将运行结果输出至 result.log文件中
find . -name cal* |xargs chmod 700
### 这个是为后期计算表面能做的准备，为cal_surface_energy.py（后面会提到）提供可执行权限（看到surslab.1.py会理解）
```

2. surslab.1.py

```
#!/gpfs/home/mncui/soft/anaconda3/bin/python3
# Import the neccesary tools to generate surfaces
from pymatgen.core.surface import SlabGenerator, generate_all_slabs, Structure, Lattice, get_d
from pymatgen.io.vasp.inputs import Incar, Kpoints, Kpoints_supported_modes, Poscar, Potcar, VaspInput
from pymatgen.io.vasp.outputs import Dynmat, Outcar, Oszicar
# Import the neccesary tools for making a Wulff shape

import os
import shutil
import re
import time
import subprocess
```

定义函数

```
# 写入POSCAR
def write_poscar(stru,hkl,path):
```

```
    os.makedirs(str(path)) # make directory for 'structure/miller/poscar'
    slab.to('poscar',str(path)+'/POSCAR')
# 写入KPOINTS文件
def write_kpoints(path):
    # Returns an automatic Kpoint object based on a structure and a kpoint density per inverse Angstrom^3 of reciprocal cell.
    # structure-Input sturcture
    # kppvol-Grid density per Angstrom^(-3) of reciprocal cell
    # force_gamma (bool)  - Force a gamma centered mesh
    kpoints = Kpoints.gamma_automatic()  # gamma 取点 111
    kpoints.write_file(str(path)+'/KPOINTS')  # 写入path路径，KPOINTS文件
    log.write('.  ')
'''
    kpoints = Kpoints.automatic_density_by_vol(structure,int(kppvol),force_gamma)
    kpoints.write_file(str(path)+'/KPOINTS')
    log.write('.  ')
'''
# 写入INCAR
def write_incar(caltype,path):
    # caltype - calculation type: 'relax', 'static','bader'or 'pdos'
    incar = Incar.from_file('input/INCAR'+'_'+str(caltype))
    incar = incar + {'LDIPOL':'.TRUE.','IDIPOL':'3'}
    incar.write_file(str(path)+'/INCAR')
    log.write('.  ')
# 写入POTCAR(原子数，元素，路径)
def write_potcar(num,element,path):
    pot = open(str(path) + '/POTCAR','w')
    for i in range(num):
        potcar = Potcar.from_file('/gpfs/home/mncui/bin/psudopotential/paw_pbe/' + str(psudo[element[i]]) + '/POTCAR')
        pot.write(str(potcar))
    pot.close()
    log.write('.\n')
# 运行vasp，当任务PENDING 的数量<2,则自动提交任务，如果PENDING 数量>2就停止提交任务，并等待
def vasp_run(path):
    shutil.copy(str(cwd)+'/input/vasp.new',str(path))
    out = subprocess.check_output('bjobs',shell=True)   # subprocess.check_output可以在后面运行linux命令且赋值给out，牛皮
```

```
    put = []
    put = re.findall(r"PEND",str(out))
    while put != []:# when jobs pending
        if len(put) < 2: # if num of pending less than 3,bsub continuum
            os.chdir(path)
            bsub = subprocess.check_output('bsub < vasp.new',shell=True)
            log.write('\n'+str(bsub)+'\n')
            log.write('Time: '+time.asctime( time.localtime(time.time()) )
+'\n')
            log.flush()
            os.chdir('../../../')
            return
        else:
#           log.write('Pending: >%s> ' %len(put)+'\n')
            time.sleep(30)
            out = subprocess.check_output('bjobs',shell=True)
            put = []
            put = re.findall(r"PEND",str(out))
#           log.write('         >%s> ' %len(put)+'\n')
            log.flush()
    os.chdir(path)
    bsub = subprocess.check_output('bsub < vasp.new',shell=True)
    print(bsub)
    log.write('\n'+str(bsub)+'\n')
    log.write('Time: '+time.asctime( time.localtime(time.time()) ) +'\n')
    log.flush()
    os.chdir('../../../')
# 定义赝势文件的选取，如加Ag_sv还是 Ag_pv的赝势，字典
psudo = {'Ac': 'Ac', 'Ag': 'Ag', 'Al': 'Al', 'Ar': 'Ar', 'As': 'As', 'Au':
'Au', 'B': 'B', 'Ba': 'Ba_sv', 'Be': 'Be_sv', 'Bi': 'Bi', 'Br': 'Br', 'C':
'C', 'Ca': 'Ca_sv', 'Cd': 'Cd', 'Ce': 'Ce', 'Cl': 'Cl', 'Co': 'Co', 'Cr':
'Cr_pv', 'Cs': 'Cs_sv', 'Cu': 'Cu_pv', 'Dy': 'Dy_3', 'Er': 'Er_3', 'Eu':
'Eu', 'F': 'F', 'Fe': 'Fe_pv', 'Ga': 'Ga_d', 'Gd': 'Gd', 'Ge': 'Ge_d',
'H': 'H', 'He': 'He', 'Hf': 'Hf_pv', 'Hg': 'Hg', 'Ho': 'Ho_3', 'I': 'I',
'In': 'In_d', 'Ir': 'Ir', 'K': 'K_sv', 'Kr': 'Kr', 'La': 'La', 'Li':
'Li_sv', 'Lu': 'Lu_3', 'Mg': 'Mg_pv', 'Mn': 'Mn_pv', 'Mo': 'Mo_pv', 'N':
'N', 'Na': 'Na_pv', 'Nb': 'Nb_pv', 'Nd': 'Nd_3', 'Ne': 'Ne', 'Ni':
'Ni_pv', 'Np': 'Np', 'O': 'O', 'Os': 'Os_pv', 'P': 'P', 'Pa': 'Pa', 'Pb':
'Pb_d', 'Pd': 'Pd', 'Pm': 'Pm_3', 'Pr': 'Pr_3', 'Pt': 'Pt', 'Pu': 'Pu',
'Rb': 'Rb_sv', 'Re': 'Re_pv', 'Rh': 'Rh_pv', 'Ru': 'Ru_pv', 'S': 'S',
'Sb': 'Sb', 'Sc': 'Sc_sv', 'Se': 'Se', 'Si': 'Si', 'Sm': 'Sm_3', 'Sn':
```

```
'Sn_d', 'Sr': 'Sr_sv', 'Ta': 'Ta_pv', 'Tb': 'Tb_3', 'Tc': 'Tc_pv', 'Te':
'Te', 'Th': 'Th', 'Ti': 'Ti_pv', 'Tl': 'Tl_d', 'Tm': 'Tm_3', 'U': 'U',
'V': 'V_pv', 'W': 'W_pv', 'Xe': 'Xe', 'Y': 'Y_sv', 'Yb': 'Yb_2', 'Zn':
'Zn', 'Zr': 'Zr_sv'}
# 这个是固定在什么坐标范围的原子
def addsd(structure, midcoor, path):
    # structure - POSCAR
    # midcoor - middle of coordinates for fix
    lines = open(structure,'r').readlines()
    with open(structure,'w') as f:
        if midcoor < 0.5: # if atom all  on down side of slabs
            for num in range(len(lines)):
                if num > 7:
                    splits = lines[num].split()
                    if float(splits[2])< float(midcoor): # if atom low
that midcoor, fixing
                        f.write(splits[0]+' '+splits[1]+' '+splits[2]+' F
F F ')
                    else: # if atom above midcoor , allow movement
                        f.write(splits[0]+' '+splits[1]+' '+splits[2]+' T
T T ')
                    if len(splits)>3: #If the atom had an identifer (ex:
'.5 .5 .5 Mg'), add the identifier
                        f.write(splits[3]+' \n')
                    else:
                        f.write(' \n')
                elif num < 7: # Write all lines prior to line 7 as is
                    f.write(lines[num])
                elif num==7:
                    f.write('Selective \n') # Add 'S' tag whil keeping
                    f.write(lines[num]) # Add previous identifier
        if midcoor > 0.5: # if atom all on top side of slabs
            for num in range(len(lines)):
                                if num > 7:
                                        splits = lines[num].split()
                                        if float(splits[2]) >
float(midcoor): # if atom low that midcoor, fixing
                                                f.write(splits[0]+'
'+splits[1]+' '+splits[2]+' F F F ')
                                        else: # if atom above midcoor ,
allow movement
```

```
                                                f.write(splits[0]+' '+splits[1]+' '+splits[2]+' T T T ')
                                        if len(splits)>3: #If the atom had an identifer (ex: '.5 .5 .5 Mg'), add the identifier
                                                f.write(splits[3]+' \n')
                                        else:
                                                f.write(' \n')
                                elif num < 7: # Write all lines prior to line 7 as is
                                        f.write(lines[num])
                                elif num==7:
                                        f.write('Selective \n') # Add 'S' tag whil keeping
                                        f.write(lines[num]) # Add previous identifier
# 计算表面能，也就是在每个产生的结构文件夹中写入一个文件"cal_surface_energy.py"，
# main.lsf中需要给他加上可执行权限 700
# 执行calculate.py的目的就是运行每个文件夹中的cal_surface_energy.py，然后提取输出的表面能
def surf_energy(structure, S_area, path):
    os.chdir(path)
    with open('cal_surface_energy.py','a') as f:
        f.write('#!/gpfs/home/mncui/soft/anaconda3/bin/python3.6 \n')
        f.write('import os \nimport shutil \nimport re\nimport time\nimport subprocess\n')
        f.write('from pymatgen.core.surface import SlabGenerator, generate_all_slabs, Structure, Lattice, get_d \nfrom pymatgen.io.vasp.inputs import Incar, Kpoints, Kpoints_supported_modes, Poscar, Potcar, VaspInput\nfrom pymatgen.io.vasp.outputs import Dynmat, Outcar, Oszicar\n')
        f.write('# writen by mncui 2019.02.06 for calculation Surface Energy\n')
        f.write("structure = '%s'\n" %structure)
        f.write("oszicar = Oszicar('../../bulk/'+str(structure)+'/OSZICAR')\n")
        f.write("E_bulk = oszicar.final_energy\n")
        f.write("natom = %f \n" %natom)
        f.write("S_area = %f \n" %S_area)
        f.write("oszicars = Oszicar('OSZICAR')\n")
        f.write("E_slab = oszicars.final_energy\n")
```

```
        f.write("pos = [line.strip(' ').split() for line in open('POSCAR')]\n")
        f.write("natoms = sum([item for item in [float(i) for i in pos[6][:]]])\n")
        f.write("with open('surface_energy','w') as f:\n")
        f.write("    f.write('E_bulk  = %.2f \\nnatom  = %s\\n' %(E_bulk, natom))\n")
        f.write("    f.write('E_slab = %.2f  \\nnatoms = %s\\n' %(E_slab, natoms))\n")
        f.write("    f.write('S_area = %.2f\\n' %S_area)\n")
        f.write("    f.write('E_surf = (E_slab - (natoms/natom) * E_bulk)/(S_area * 2)\\n')\n")
        f.write("    f.write('E_surf = %.3f' %((float(E_slab) - (float(natoms)/float(natom)) * float(E_bulk))/(S_area * 2)))")
    os.chdir('../../../')
```

主程序

Python | L1 (default) | ···

```
#----------------------------main.py_begin------------------------------


cwd = os.getcwd()  # 获取当前文件目录
log = open('log','w')  # 新建log文件，方便在程序运行的时候观察输出
dire = open('dir','w') # 新建dir文件，可以输出程序运行过程中新建的结构目录
log.write(cwd + '\n')
poscar = os.listdir('origin/')  # 将origin文件夹中的结构名赋值给poscar
# 第一层for循环，开始逐个结构计算了
for stru in poscar:
    log.write('----------------------'+str(stru)+'-----------------------\n')
    origin = 'origin/' + str(stru)  # stru are name of POSCAR file from origin directory
    struct = Structure.from_file(origin) # read POSCAR
    i = 0
    j = 0
    hkl = []
    posplit = [line.strip(' ').split() for line in open(origin)]
    element_line = posplit[5][:]
    num = len(element_line)      # atomic type number of bulk
```

```
    natom = sum([item for item in [float(i) for i in posplit[6][:]]])   # atomic number of bulk
    #----------calculate bulk energy-----------
    os.makedirs('slab/bulk/' + str(stru))
    struct.to('poscar', 'slab/bulk/'+ str(stru) + '/POSCAR')
    shutil.copy(str(cwd)+'/input/INCAR_relax', 'slab/bulk/' + str(stru)+'/INCAR')
    write_potcar(num, element_line, 'slab/bulk/' + str(stru))
    write_kpoints('slab/bulk/' + str(stru))
    vasp_run('slab/bulk/' + str(stru))

    #-------------Slab generator- --------------
    slabs = generate_all_slabs(struct,2,15,15)
    log.write("    %s unique slab structures have been found for a max Miller index of 2" %len(slabs)+ '\n')
    # 第二层for循环，开始对某结构，产生的所有slab，进行逐个操作
    # 包括新建文件夹，分别写入INCAR,KPOINTS,POSCAR,POTCAR,vasp.lsf,cal_surface_energy.py文件
    # 计算表面积
    for slab in slabs:
        i = i+1
        if str(slab.miller_index) != str(hkl):
            j += 1
            hkl = slab.miller_index
            log.write('  %s  '%j + str(hkl) + '\n')
            center = slab.center_of_mass
            surf_area = slab.surface_area  # Surface area of slab
            log.write(' surface_area is : ' + str(surf_area)+ '\n')
            n = 0
            path = 'slab/' + str(stru)+'/'+str(hkl[0])+str(hkl[1])+str(hkl[2])+'-'+str(n)
            dire.write(path+'\n')  # write directory file for calculate.py scripts
            write_poscar(stru, hkl, path)

            structure = Structure.from_file(str(path)+'/POSCAR')
            write_kpoints(path)
            write_incar('relax',path)
            # write potcar
            p = [line.strip(' ').split() for line in open(str(path)+'/POSCAR')]
```

```
            pelements = p[5][:]
            pn = len(pelements)
            write_potcar(pn, pelements, path)

            addsd(str(path)+'/POSCAR',center[2],path)
            surf_energy(stru,surf_area,path)
            vasp_run(path)
        else:
            j += 1
            log.write('euqal in miller_index but not in determination'+str(hkl)+ '\n')
            center = slab.center_of_mass
            surf_area = slab.surface_area
            log.write(' surface_area is : ' + str(surf_area)+ '\n')
            n = n+1
            path = 'slab/' + str(stru)+'/'+str(hkl[0])+str(hkl[1])+str(hkl[2])+'-'+str(n)
            dire.write(path+'\n')
            write_poscar(stru, hkl, path)
            write_kpoints(path)
            structure = Structure.from_file(str(path)+'/POSCAR')
            write_incar('relax',path)
            # write potcar
            p = [line.strip(' ').split() for line in open(str(path)+'/POSCAR')]
            pelements = p[5][:]
            pn = len(pelements)
            write_potcar(pn, pelements, path)

            addsd(str(path)+'/POSCAR',center[2],path)
            surf_energy(stru,surf_area,path)
            vasp_run(path)

```

3. calculate.py

Python L1 (default) ···

```
#!/gpfs/home/mncui/soft/anaconda3/bin/python3
# Import the neccesary tools to generate surfaces
from pymatgen.core.surface import SlabGenerator, generate_all_slabs,
```

```
Structure, Lattice, get_d
from pymatgen.io.vasp.inputs import Incar, Kpoints,
Kpoints_supported_modes, Poscar, Potcar, VaspInput
from pymatgen.io.vasp.outputs import Dynmat, Outcar, Oszicar
# Import the neccesary tools for making a Wulff shape

import os
import shutil
import re
import time
import subprocess
import pprint
# 读取dir文件中的目录信息
f = [line.strip('\n').split() for line in open('dir','r')]
for i in range(len(f)):
    path = str(f[i][0])
    print('path:'+path)
    os.chdir(path)
    os.system('chmod 700 cal_surface_energy.py')
    os.system('./cal_surface_energy.py')
    os.system('grep E_surf surface_energy')
    os.chdir('../../../')
    print('----')
```