Vulcan

Release 0.1

Thiago Trevizam Dorini

Nov 23, 2021

CONTENTS

Vulcan is a Python code to make life easier when working with VASP. It optimized all steps in the calculation process, from creating the structure to be studied to making a backup of the results on a SQL database. It make's heavy use of the Atomic Simulation Environment (ASE). You can look their documentation here: https://wiki.fysik.dtu.dk/ase/

Note: This project is under active development.

CHAPTER ONE

CONTENTS

1.1 Usage

1.1.1 Requirements

- Python 3.x;
- Numpy;
- Matplotlib;
- ASE (Atomic Simulation Environment).

1.1.2 Installation

It is only necessary to get two files to use this program for now: vulcan.py and vulcan.sh. They can be found at my GitHub page: https://github.com/DoriniTT/Vulcan

1.1.3 Initialization

Since we will be using the ASE calculators, you need to set your environment so that the program knows where to find the right executables and files. For this, you can go to:

https://wiki.fysik.dtu.dk/ase/ase/calculators/vasp.html?highlight=vasp#ase.calculators.vasp.Vasp

Note: The Vulcan program only supports VASP for now, but I'm working on adding more ab initio codes.

At its core, this program is really simple. It has the vulcan.sh file to manage the environment for the vulcan.py file. After the configuration step, all you have to do to launch the code is:

```
$ sh vulcan.sh
```

But first, we need to know how to configure these files for the type of calculation that we intend to do, which is what we will see in the next section.

1.2 Configuration - vulcan.py

This is the most important part. Let's do a step-by-step general configuration with a example structure.

I'll assume that you have access to a supercomputer to launch your calculations as a job (i.e. Slurm). I'll show later how to launch your calculations in your desktop machine later. To begin with, create a new directory in your $HOME (~) for this project and put both the vulcan.py and vulcan.sh there.

```
$ mkdir ~/tutorial-vulcan
$ cd ~/tutorial-vulcan
```

Assuming that these files are already in your home:

```
$ cp ~/vulcan.* ~/tutorial-vulcan
```

After this step, we can start configuring these files.

1.2.1 Structure generation

In general, the hole python script is centered around the `structures` function, which returns a list with all the structures that we want to study. All you have to do is create an `Atoms` ASE object and put it in the list. In the vulcan.py script, this needs to be done under the **DEFINE THE STRUCTURES HERE** commentary.

Note: If you are not familiar with the Atomic Simulation Environment, please take a look at their documentation on how to create an **Atoms** object.

We will begin creating an Cu-FCC structure:

```
#################### DEFINE THE STRUCTURES HERE ########################

cu = bulk('Cu', 'fcc', a=3.6)
struc = [cu]

#######################################################################
```

Here we will put only one structure to make things easier, but the program is made so that you can put as much structures as you want. That's all you need for this part. It becames more complex according to the structures that you want to create, but this part is entirely up to you.

1.2.2 Calculation setup

Now you need to focus on the `calculators` function. I have already put some default parameters for a DFT calculation with VASP, but you need to modify it according to what you need. For the functions implemented on this code, these default parameters usually works just fine. You can look at the VASP wiki for a description of all input parameters for VASP: https://www.vasp.at/wiki/index.php/The_VASP_Manual

1.2.3 Launch script setup

The last thing that you need to create is a `launch_{your-machine}` function (it is located right after the `run_step_relax` function). I have 4 functions created depending on the machine that I'm using. You need to create your own function based on a JOB script that you want. In my case, I automitized the number of processors that the JOB script will launch based on each structure's number of atoms. You can take advantage on the code's defined variables to adjust how you automitize your JOB.

Important: The only line that you cannot touch in the JOB script is the:

```
$ python run_$structure.py
```

Please leave this line as the last one in your job, otherwise the code will not work.

1.2.4 Optional setup

Although this is a optional, I highly recommend you to take a look on the `run_step_relax` function on the vulcan.py script. Depending on the type of calculation you intend to do, this is the function that you need to modify. Again, it has default parameters that works well for some cases.

Note: The **run_step_relax** function is the most versitile function in the code. Please try to understand its format if you want to add a new type of calculation.

1.3 Configuration - vulcan.sh

Now, this file's configuration will be devided into two parts: A **permanent** configuration, in which you need to configure only once for your machine, and the defining of the variables.

For the **permanent** configuration, the first thing you need is to define the `machine` bash variable with the same name as your supercomputer. Let's say you are in a cluster called **samba**:

```
machine="samba"
```

Now you also need to add a `elif` under the `Creating the vasp.slurm` commentary.

Note: Remember that you needed to create a function for the JOB in the vulcan.py script. In our example, this function should be called `launch_samba`.

Therefore, in our example, you should add the line:

```
elif [[ $machine == "samba" ]]; then
    echo -e "from main_launch import *\nx = Calculo('$here' ,'$work', '$database', '$xc',
↪ $encut)\nx.launch_samba($structure)" > launch_$structure.py
```

Next, under the `LAUNCH` commentary, you need to add another `elif`:

```
elif [[ $machine == "samba" ]]; then
    echo "Samba!"
    sbatch run_$structure.slurm > output_sbatch; awk '{ print $4 }' output_sbatch >
↪$here/jobid_$structure
```

Note: I'm considering that your machine have a single partition. If that's not the case, you can add a variable in the `launch_samba` function with the partition and call it using a bash variable in the vulcan.sh script with the name `partition`. Follow the `explor` and `occigen` examples if you want to add yours.

This is the command to submit your calculation to the queue. In my case, I launch with the **sbatch** command, but you modify it according to your machine. We're done with the permanent configuration! Let's move on to the last part (finally!).

1.3.1 Variables on the vulcan.sh

The last thing you need to configure are the variables at the top of the file. I'll put a standand configuration and explain it.

```
#-------Parameters------#
PROJECT="VULCAN_PROJECT"
namework="tutorial_vulcan"
machine="samba"
partition_explor="std" # On Explor --"std", "sky", "mysky"
partition_occigen="HSW24" # On Occigen -- "BDW28", "HSW24"
xc="pbe"

vasp="True"
qe="False"
#------Calculation------#
ncore_test="False"

slab="False"
bader="False"

#########
md="False"
gamma="False"
relax="True"
dos="False"
stm="False"
##
cohp="False"
nbands="500"
##

##
adsorption="False"
plane_of_separation="9.2" # In the z direction
calculate_chgdiff="False"
##

##########
#-------Parameters------#
```

The variables `PROJECT` and `namework` will define the directory's names in the $SCRATCH to go in and launch the calculation. We set `vasp="True"` (I'm working on adding Quantum Espresso to work with this code as well). Next, in the "Calculations" section, there are several really important keys for the calculation. Each of them will activate one part of the `run_step_relax` function in the vulcan.py script, so you can choose what type of calculation you want to do. This code can do multiple types of series calculations for each structure, following the order that the variables appears. In this example, I'm activating only the `relax` flag, meaning that I want to do a simple relaxation in the Cu-FCC structure.

Note: The `run_step_relax` function is faily optimized, because it will make backup files of the important files on each type of calculation. If you want to restart your calculation when it stops, it will verify what are the flags that finished well and continue exactly from where you stoped. This way you don't need to be afraid to re-launch all the calculations.

And that's it! Type `sh vulcan.sh` and watch it run.

1.4 Automated backup

At the end of the calculation, there are several functions to save the important files as a backup. The way the script is structured now can be a bit heavy on the backup (I'm tired of losing files), but if you need it is possible to modify it easily on the `run_step_relax` function.

The most important backup that the script does is to save the current state of each structure to a SQLite3 database (`.db format`), this way you'll have all your results in a single place. The second layer of the backup is done by copying several important files the each structure and saving it to a **CALC_STATES** folder in the same directory that you launched your calculations. In this example it will be in the "~/tutorial-vulcan" folder.

1.5 Useful scripts

I developed a few scripts that can help in the result's analisys of this code, specially when it comes to the final results on the SQLite3 database, so we'll begin by this script. It is a good idea to put the folder where you git cloned these files in your path. So in your `~/.bashrc` you should add:

```
export PATH=${PATH}:~/path_to_folder_with_scripts
```

This way you can use this script anywhere.

1.5.1 search_db.py

This script is used to get some simple information on the structures that a .db file has. A basic usage is with the command:

```
$ search_db -f {database_name}.db
```

For a sample database where I have the results of the H2O molecule, the output is:

```
There are 1 structures in this database!
ID | Formula | T. Energy (eV) | Max force (eV/A) | a (A) | b (A) | c (A)
1   H2O   -14.22402066   0.015   7.94 7.94 7.94


Only valuable for structures with the same composition:
The structure 1 is the most stable with the total energy of -14.22402066 eV
```

Another interesting thing that can be done is select and visualize the structures that you want by ID. Here I'll select ID = 1 (which is the only structure on the database):

```
search_db -f h2o.db -ids 1 -v True
```

It will display not only the information regarding this structure but also will use the `view` function of ASE to plot its structure.

1.5.2 forces.py

This is a simple script to output the forces according to the OUTCAR file. It is useful to have an idea of how far the structure is from convergence when doing a structure relaxation. You only need to specify the -c flag, in which the program will search for all atoms with forces higher than this (units in eV/Angs.).

```
forces -c 0.02
```

1.6 Types of calculations

There are already a few available routines that you can perform using this code. It is relativelly simple to implement new routines based on the ones already available.

Let's explain what each routine is doing.

TO BE DONE

1.6.1 Gamma-point

Considering that the structure is far from fully relaxed, the idea of this function is to get a rough relaxation with the $gamma$-point. Before each calculation, the only thing that it does is verify if there is the line `reached required` on the OUTCAR_gam file. If it does not, the calculation will begin either from the CONTCAR file (if it exists), or from a new POSCAR.

1.6.2 Relaxation

The same idea as on the $gamma$-point calculation but now using a higher number of k-points. This time it will verify the `reached required` phrase on the OUTCAR_relax file. The number of k-points is automatically set by using the following formula:

$$k = (precision * 2\pi/a, precision * 2\pi/b, precision * 2\pi/c)$$

where `precision = 6` by default. You can set this higher if needed. If the flag `slab` is set to True in the vulcan.sh file, `k[2] = 1` is used.

At the end of the calculation, if everything goes well, it will save the files defined in the `input` variable as `*_relax`. You can add or remove files in this variable as you please.

1.6.3 Density of States

It uses the same idea as the *relax* function but setting different inputs for the INCAR. It will double the number of *k*-points and set the following inputs: `icharg=11, ismear=-5, prec='Accurate', nsw=0`. If the flag `results` is True on the vulcan.sh file, it will also plot the total Density of States to a file called `dos_results` in the launch directory.

1.6.4 COHP

Similar idea to the Density of States calculation, but with different flags following a COHP calculation. As can be seen in the Lobster manual http://schmeling.ac.rwth-aachen.de/cohp/index.php?menuID=6, one very important parameter is the number of bands, which you need to set manually on the vulcan.sh file.

After a successful calculation, this routine will also calculate the COHP using the `lobster` program, so be sure to have the binary file on your path. (You can get the script for free on their website). I set a very simple `lobsterin` file in the program as using only the recommended basis functions from the POTCAR file, feel free to set your own file according to what you need. You only need to modify the *lobsterin* string on the vulcan.py script.

1.6.5 STM

1.6.6 Adsorption

1.6.7 Molecular dynamics