TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

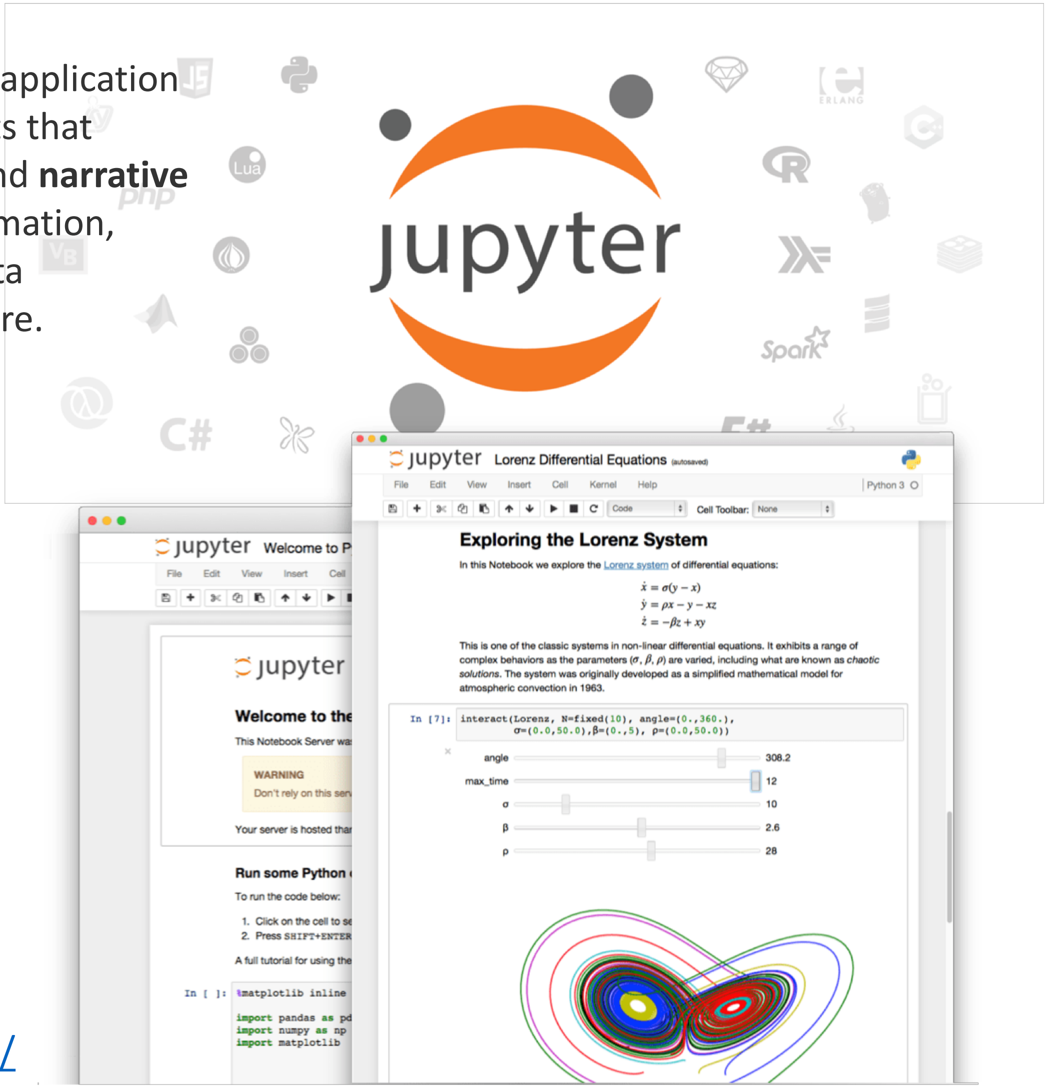
TEXAS
The University of Texas at Austin

A Notebook-based Platform for Computational Chemistry and Materials Science

Chun-Yaung Lu
alu@tacc.texas.edu

Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain **live code**, **equations**, **visualizations** and **narrative text**. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.



Welcome to the

This Notebook Server wa

WARNING

Don't rely on this serv

Your server is hosted thar

Run some Python

To run the code below:

1. Click on the cell to se
2. Press SHIFT+ENTER

A full tutorial for using the

```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

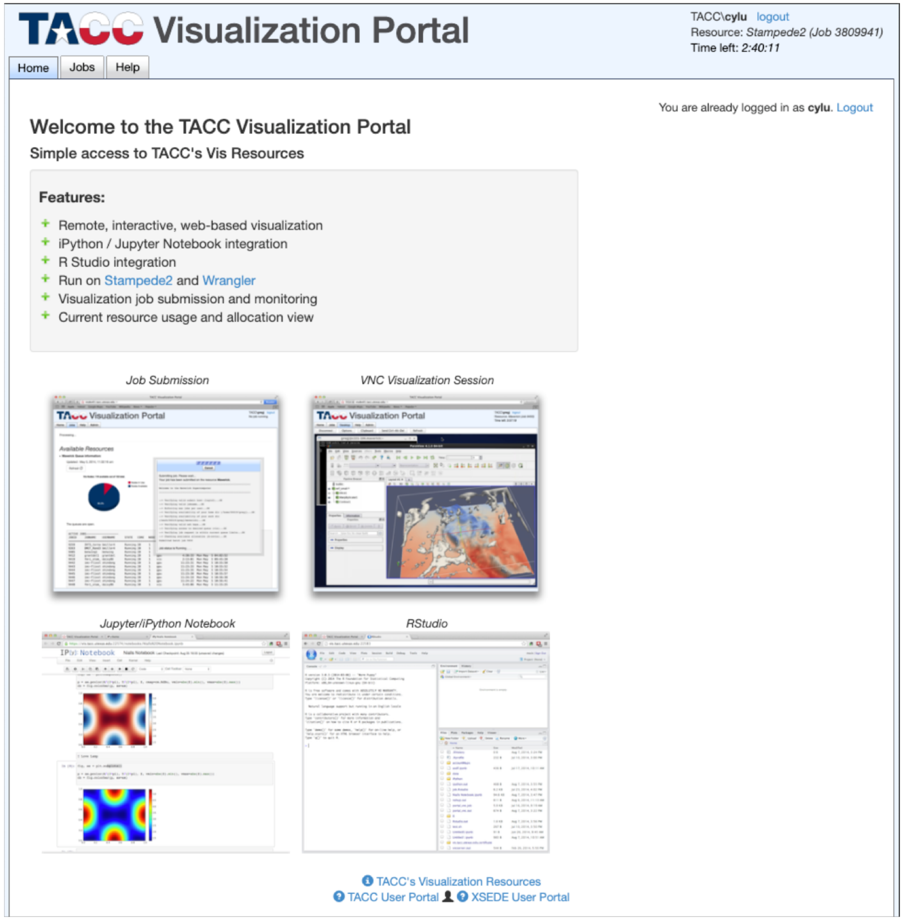
```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                 σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

angle 308.2
max_time 12
σ 10
β 2.6
ρ 28

https://jupyter.org/

TACC Visualization Portal

https://vis.tacc.utexas.edu



TACC\cylu logout
Resource: *Stampede2 (Job 3809941)*
Time left: *2:40:11*

You are already logged in as **cylu**. Logout

Welcome to the TACC Visualization Portal

Simple access to TACC's Vis Resources

Features:

- Remote, interactive, web-based visualization
- iPython / Jupyter Notebook integration
- R Studio integration
- Run on Stampede2 and Wrangler
- Visualization job submission and monitoring
- Current resource usage and allocation view

Job Submission

VNC Visualization Session

Jupyter/iPython Notebook

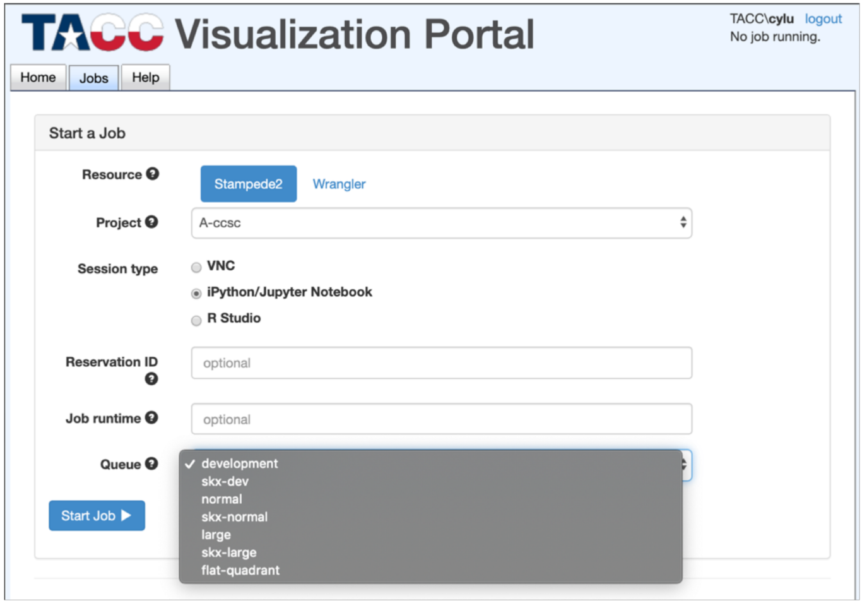
RStudio

TACC's Visualization Resources
TACC User Portal XSEDE User Portal

- Use your **TACC** or **XSEDE** User Portal username and password to log in
- Run VNC, iPython/Jupyter Notebook, R Studio on Stampede2 or Wrangler
- Available Stampede2 queues for iPython/Jupyter: *development, skx-dev, normal, skx-normal*, …
- One compute node each time
  KNL: 68 cores, SKX: 48 cores

TACC Visualization Portal

https://vis.tacc.utexas.edu



No job running.

Start a Job

Resource: Stampede2 Wrangler

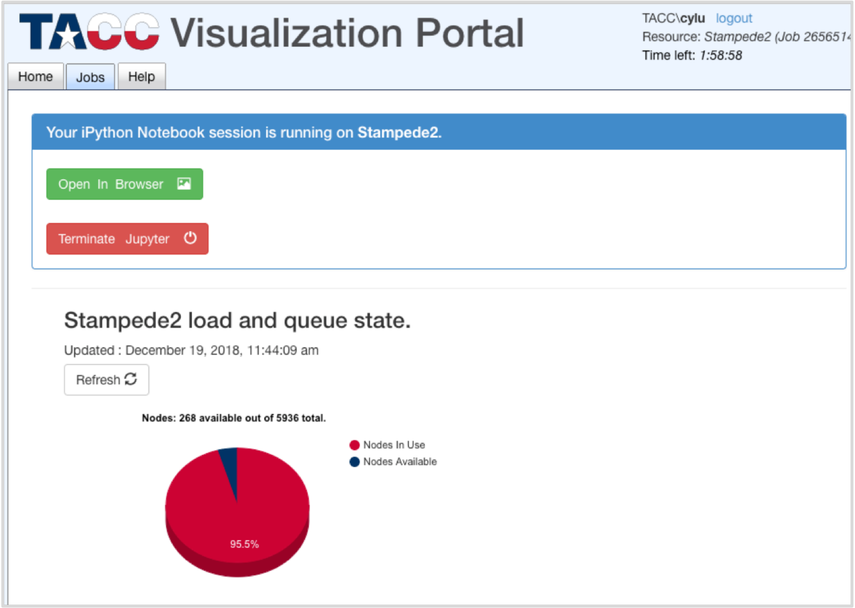
Project: A-ccsc

Session type:
○ VNC
● iPython/Jupyter Notebook
○ R Studio

Reservation ID: optional

Job runtime: optional

Queue:
✓ development
skx-dev
normal
skx-normal
large
skx-large
flat-quadrant



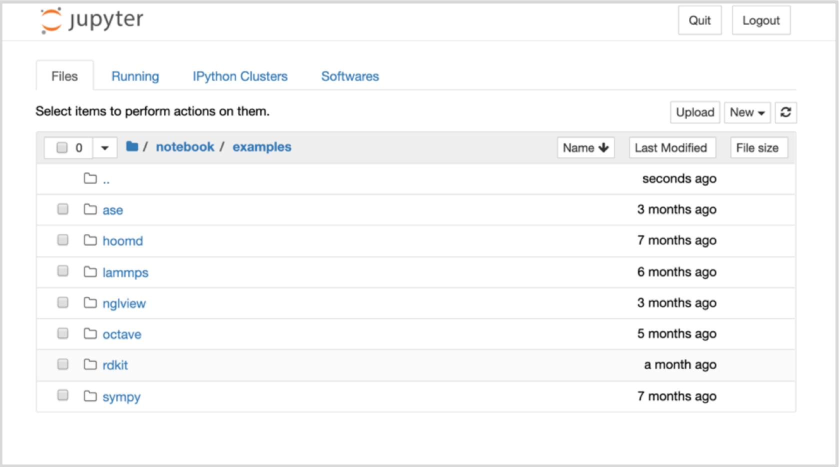
Resource: *Stampede2 (Job 265651*

Time left: *1:58:58*

Your iPython Notebook session is running on **Stampede2**.

Stampede2 load and queue state.

Updated : December 19, 2018, 11:44:09 am



Select items to perform actions on them.

□ 0 / notebook / examples	Name	Last Modified	File size
..		seconds ago	
□ ase		3 months ago	
□ hoomd		7 months ago	
□ lammps		6 months ago	
□ nglview		3 months ago	
□ octave		5 months ago	
□ rdkit		a month ago	
□ sympy		7 months ago	

Running LAMMPS

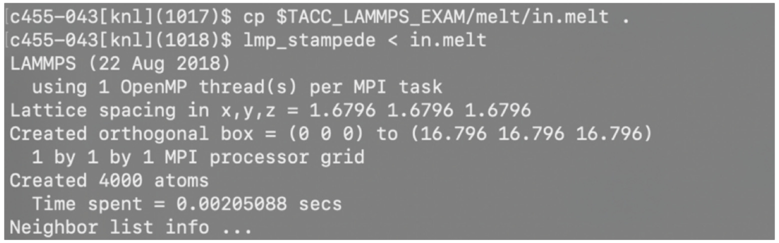
On Stampede2:

`$module load lammps`

will load default version 16Mar18.

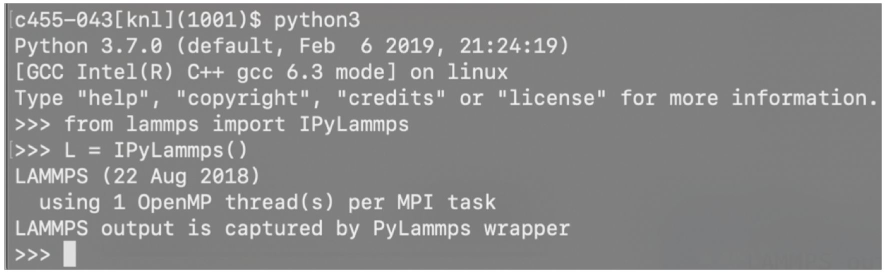
The latest version 5Jun19 is also available

Command line: lmp_stampede < input



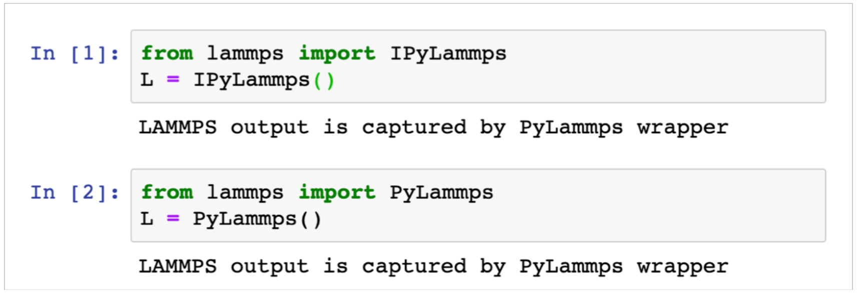
```
c455-043[knl](1017)$ cp $TACC_LAMMPS_EXAM/melt/in.melt .
c455-043[knl](1018)$ lmp_stampede < in.melt
LAMMPS (22 Aug 2018)
  using 1 OpenMP thread(s) per MPI task
Lattice spacing in x,y,z = 1.6796 1.6796 1.6796
Created orthogonal box = (0 0 0) to (16.796 16.796 16.796)
  1 by 1 by 1 MPI processor grid
Created 4000 atoms
  Time spent = 0.00205088 secs
Neighbor list info ...
```

In Python3



```
c455-043[knl](1001)$ python3
Python 3.7.0 (default, Feb  6 2019, 21:24:19)
[GCC Intel(R) C++ gcc 6.3 mode] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from lammps import IPyLammps
>>> L = IPyLammps()
LAMMPS (22 Aug 2018)
  using 1 OpenMP thread(s) per MPI task
LAMMPS output is captured by PyLammps wrapper
>>> 
```

In Jupyter



```
In [1]: from lammps import IPyLammps
        L = IPyLammps()

        LAMMPS output is captured by PyLammps wrapper

In [2]: from lammps import PyLammps
        L = PyLammps()

        LAMMPS output is captured by PyLammps wrapper
```

https://lammps.sandia.gov/doc/Python_head.html

More Packages

There are many many other packages that you can install and use in Jupyter

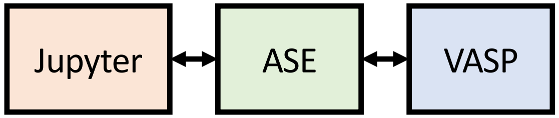
Many packages can be installed simply by using pip:

```
$pip install myPackage --user
```

Some jupyter-friendly chemistry packages that I installed and tested on Stampede2 (not a full list)

Name	Version	Function
gpaw	1.5.1	Quantum DFT
lammps	22Aug18	Classical MD
hoomd-blue	2.3.5	Classical MD, CGMD
ase	3.17.0	Simulation interface
tsase	master	Transition state library for ASE
rdkit	2018_03_4	Cheminformatics, ML
mdtraj	1.9.2	Analysis tool
pytraj	2.0.3	Analysis tool
cpptraj	18.00	Analysis tool
parmed	3.03	Analysis tool
OpenKIM	1.9.7	Force field database
libxc	4.2.3	XC library
libvdwxc	0.3.2	XC-VDW library
nglview	1.1.7	Visualizer

Atomic Simulation Environment (ASE) Supported Software



Jupyter ↔ ASE ↔ VASP

Name	Description	ST2
Asap	Highly efficient EMT code	
GPAW	Real-space/plane-wave/LCAO PAW code	YES
Hotbit	DFT based tight binding	
abinit	Plane-wave pseudopotential code	
amber	Classical molecular dynamics code	YES
castep	Plane-wave pseudopotential code	
cp2k	DFT and classical potentials	
demon	Gaussian based DFT code	
dftb	DFT based tight binding	
dmol	Atomic orbital DFT code	
QE	Plane-wave pseudopotential code	YES
exciting	Full Potential LAPW code	
aims	Numeric atomic orbital, full potential code	
fleur	Full Potential LAPW code	

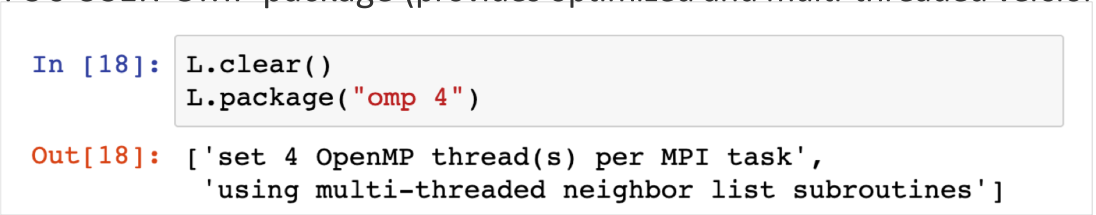
Name	Description	ST2
gaussian	Gaussian based electronic structure code	YES
gromacs	Classical molecular dynamics code	YES
gulp	Interatomic potential code	YES
jacapo	Plane-wave ultra-soft pseudopotential code	
lammps	Classical molecular dynamics code	YES
mopac	Semiempirical quantum chemistry code	
nwchem	Gaussian based electronic structure code	YES
octopus	Real-space pseudopotential code	
onetep	Linear-scaling pseudopotential code	
siesta	LCAO pseudopotential code	YES
turbomol	Fast atom orbital code	
VASP	Plane-wave PAW code	YES
dftd3	DFT-D3 dispersion correction calculator	

https://wiki.fysik.dtu.dk/ase/ase/calculators/calculators.html#module-ase.calculators

LAMMPS + Jupyter + HPC

TACC Vis Portal lets you run Jupyter Notebook on ONE compute node (KNL: 68 cores, SKX: 48 cores)

- LAMMPS’s USER-OMP package (provides optimized and multi-threaded version of many LAMMPS functions)



```
In [18]:  L.clear()
          L.package("omp 4")

Out[18]:  ['set 4 OpenMP thread(s) per MPI task',
           'using multi-threaded neighbor list subroutines']
```

- Run LAMMPS on multiple processors (use ipyparallel + mpi4py)

MPI4Py:

MPI for Python provides bindings of the Message Passing Interface (MPI) standard for the Python programming language, allowing any Python program to exploit multiple processors.

https://mpi4py.readthedocs.io/en/stable/

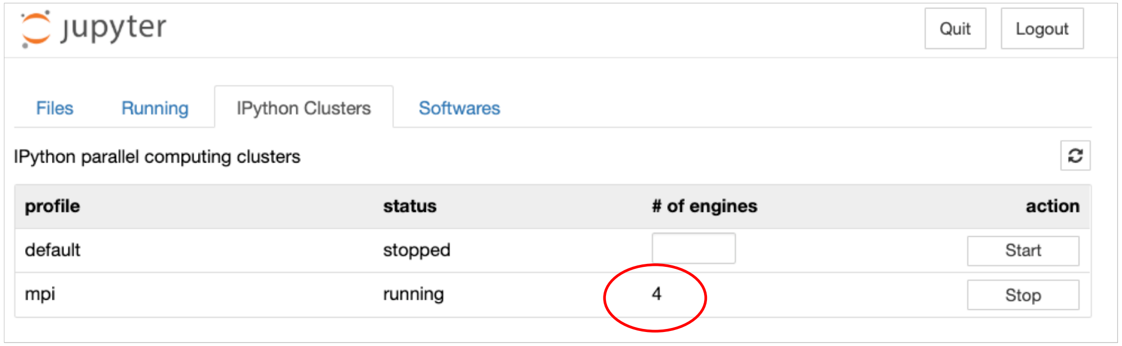
Ipyparalle:

Ipyparallel (formerly IPython parallel) enables all types of parallel applications to be developed, executed, debugged, and monitored interactively.

https://ipyparallel.readthedocs.io/en/latest/intro.html

Parallel Scheme (domain decomposition)

Step 1
Start ipcluster

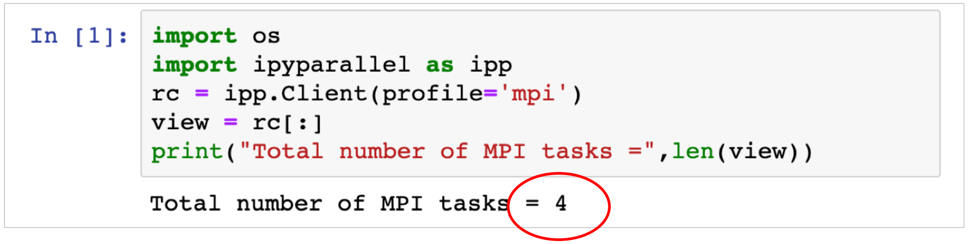


IPython parallel computing clusters

profile	status	# of engines	action
default	stopped		Start
mpi	running	4	Stop

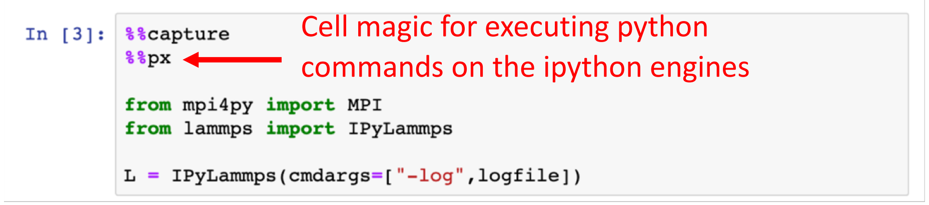
Start with 4 ipython engines

Step 2
Import ipyparalle



```
In [1]: import os
        import ipyparallel as ipp
        rc = ipp.Client(profile='mpi')
        view = rc[:]
        print("Total number of MPI tasks =",len(view))

        Total number of MPI tasks = 4
```

Step 3
Import mpi4py and start LAMMPS



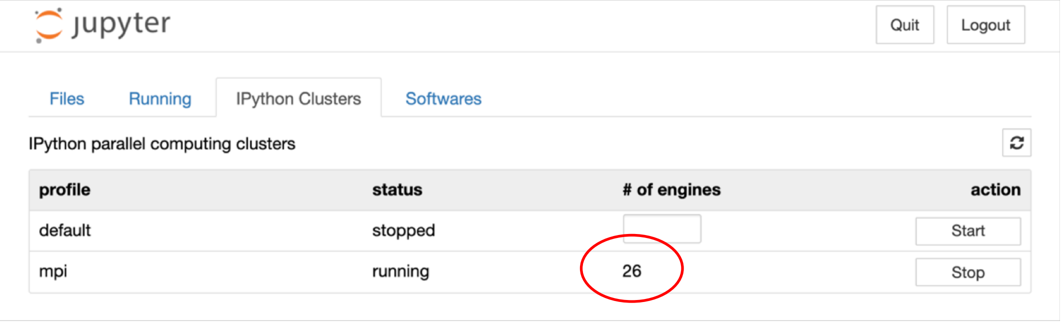
```
In [3]: %%capture
        %%px

        from mpi4py import MPI
        from lammps import IPyLammps

        L = IPyLammps(cmdargs=["-log",logfile])
```

Cell magic for executing python commands on the ipython engines

Parallel Scheme (multi-replica)

(Example: lammps_neb_hop1.ipynb)

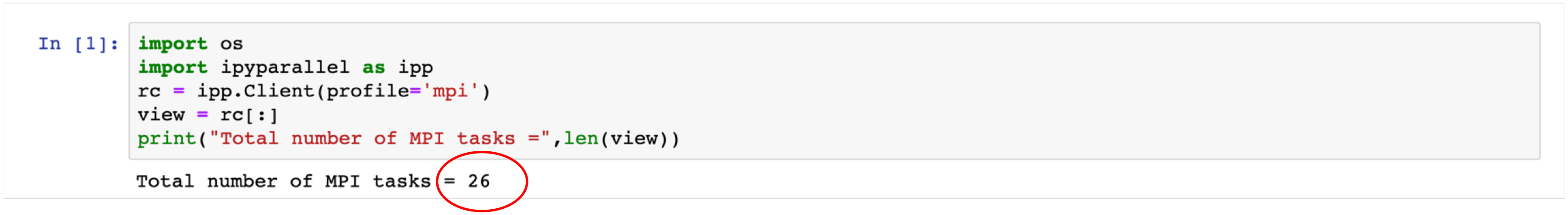
E.g. 13 replicas, 2 cpus/replica



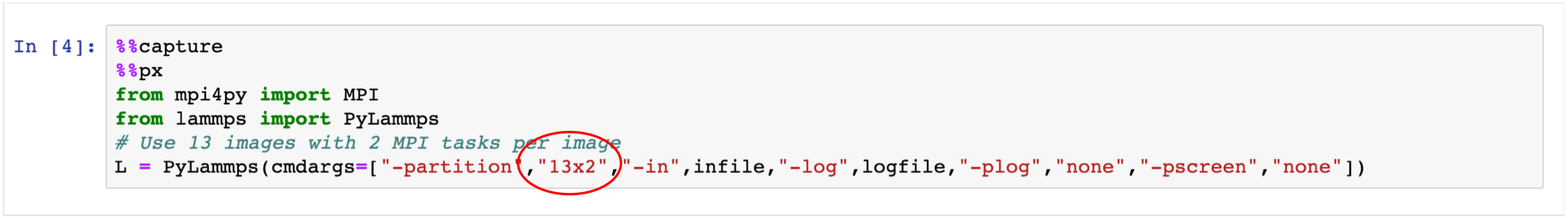
IPython parallel computing clusters

profile	status	# of engines	action
default	stopped		Start
mpi	running	26	Stop

Start with 26 iPython engines



```
In [1]: import os
        import ipyparallel as ipp
        rc = ipp.Client(profile='mpi')
        view = rc[:]
        print("Total number of MPI tasks =",len(view))

        Total number of MPI tasks = 26
```



```
In [4]: %%capture
        %%px
        from mpi4py import MPI
        from lammps import PyLammps
        # Use 13 images with 2 MPI tasks per image
        L = PyLammps(cmdargs=["-partition","13x2","-in",infile,"-log",logfile,"-plog","none","-pscreen","none"])
```

Start LAMMPS with 13 replicas with 2 cores/replica

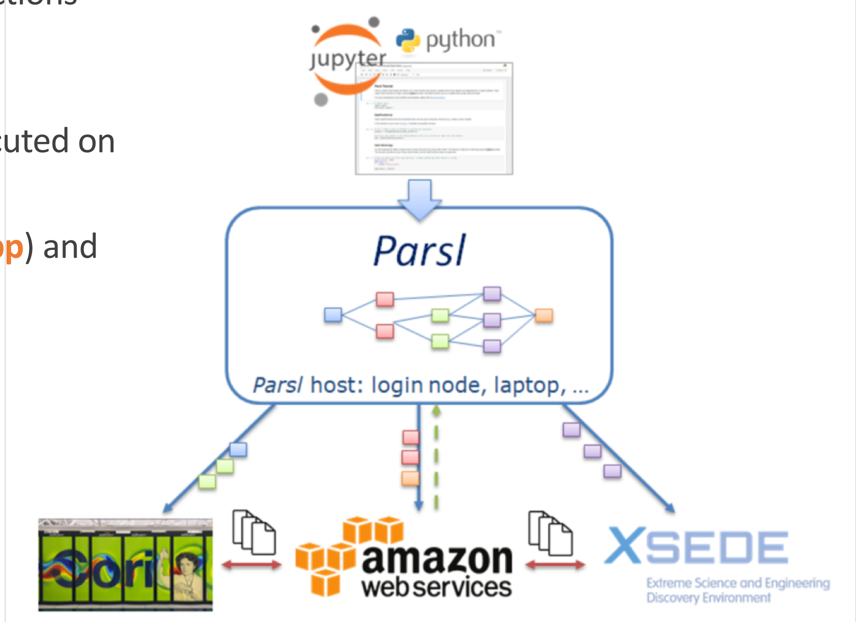
Making Workflow: PARSL

Parsl is a native Python library. It allows you to write functions that execute in parallel and tie them together with dependencies to create workflows.

“App” is a piece of code that can be asynchronously executed on an execution resource

Parsl provides support for pure Python apps (**python_app**) and also command-line apps executed via Bash (**bash_app**)

Parsl creates implicit workflows based on the passing of control or data between Apps.


jupyter
python
Parsl
Parsl host: login node, laptop, ...
Cori
amazon web services
XSEDE
Extreme Science and Engineering Discovery Environment

```
@python_app
def hello ():
   return 'Hello World!'
```

```
@bash_app
def echo_hello(stdout='echo-hello.stdout', stderr='echo-hello.stderr’):
   return 'echo "Hello World!"'
```

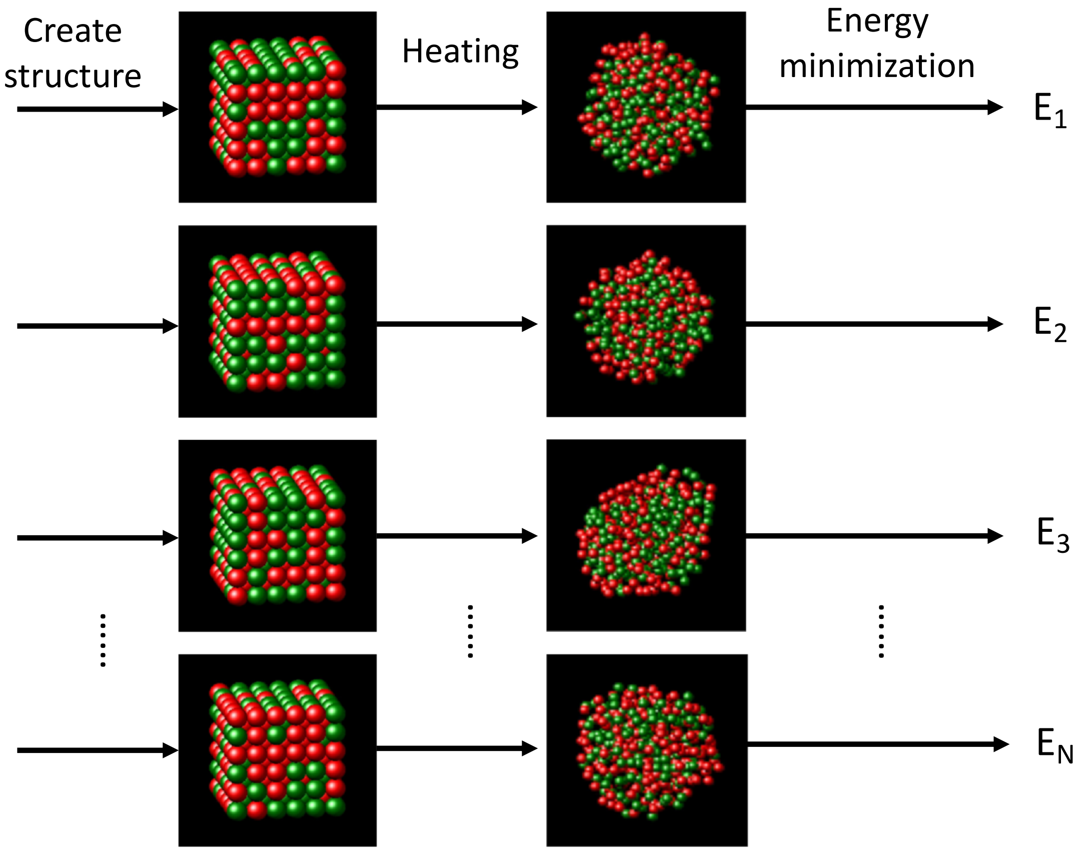
```
@bash_app
def run_lammps(stdout=‘stdout’, stderr=‘lmp.stderr’):
   return 'lmp_stampede < input'
```

http://parsl-project.org/

Example 1 (Fe/Cr random alloy)

Pure python code

M tasks
N workers

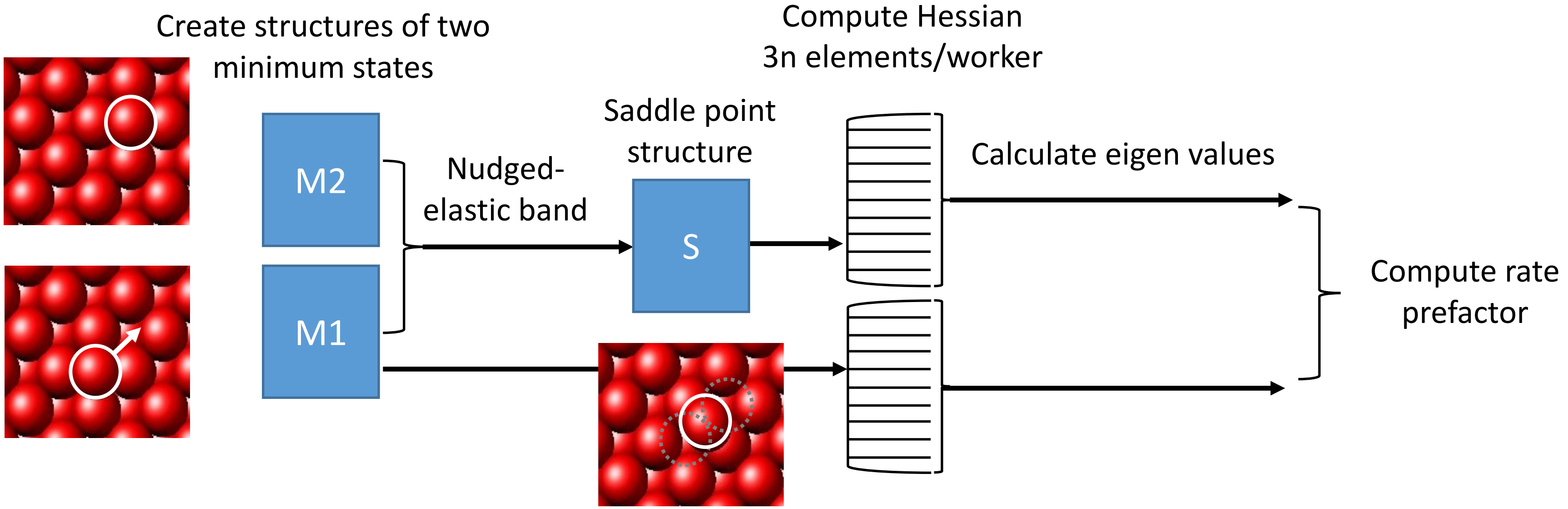


Parallel workflow

Example 2 (Al surface diffusion, Hessian matrix)

Pure python code

$$H_{3n\times 3n} = \{\partial V/\partial x_i \partial x_j\}$$



Create structures of two minimum states

M2

M1

Nudged-elastic band

Saddle point structure

S

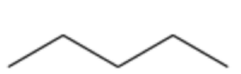
Compute Hessian
3n elements/worker

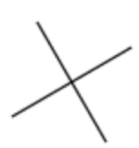
Calculate eigen values

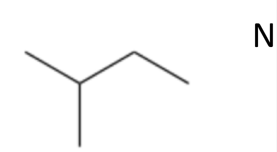
Compute rate prefactor

Example 3 (Alkane C-H Bond dissociation Energy)

Alkane isomers:
E.g. C_5H_{12}

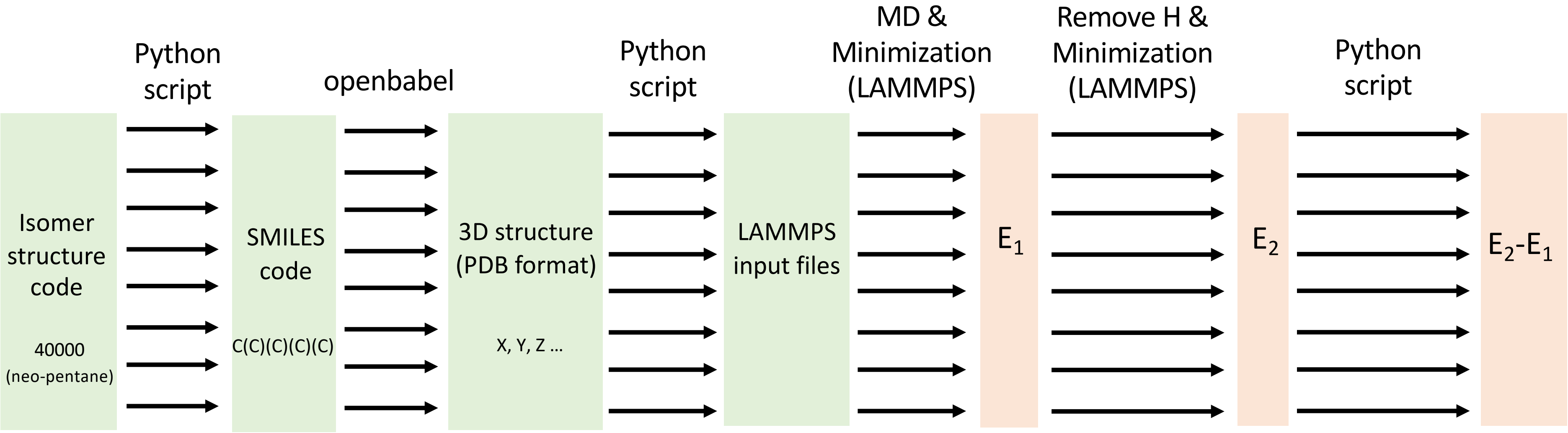

n-pentane


neo-pentane


i-pentane

Number of isomers:

$C_{10}H_{22}$	75
$C_{12}H_{26}$	355
$C_{15}H_{32}$	4,347
$C_{18}H_{38}$	60,523
$C_{20}H_{42}$	366,319
$C_{21}H_{44}$	910,726
$C_{24}H_{50}$	> 14.5M


Isomer structure code

40000
(neo-pentane)

Python script

SMILES code

C(C)(C)(C)(C)

openbabel

3D structure (PDB format)

X, Y, Z …

Python script

LAMMPS input files

MD & Minimization (LAMMPS)

E_1

Remove H & Minimization (LAMMPS)

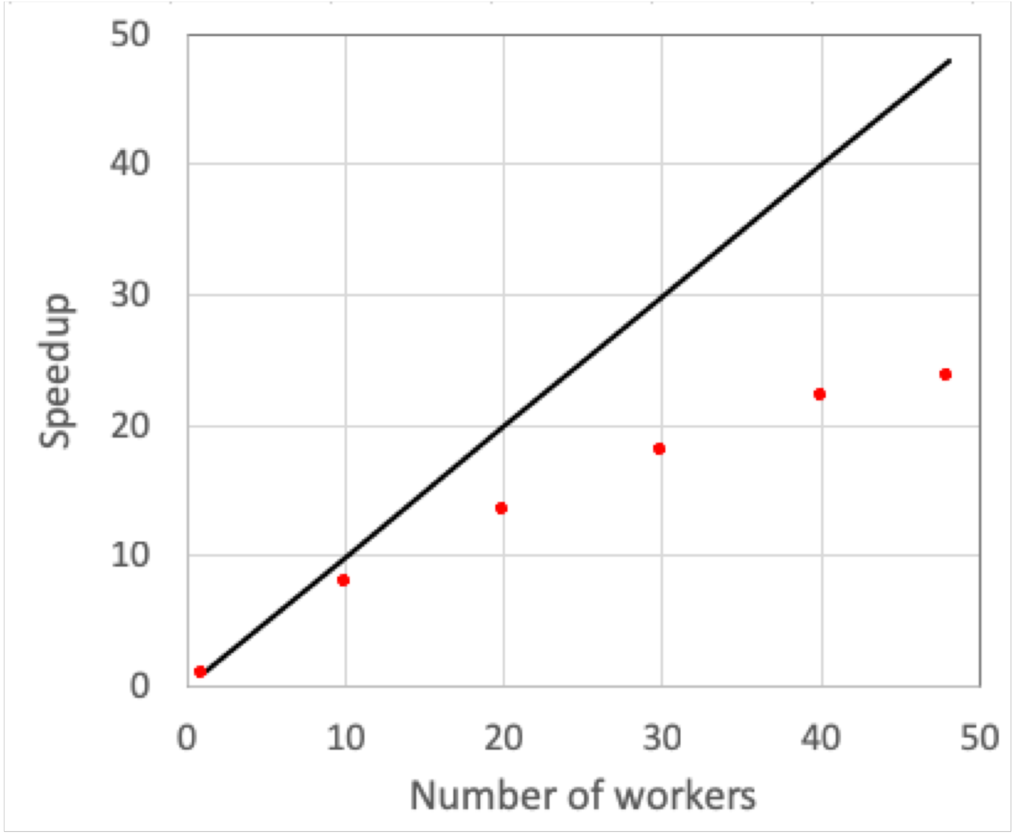
E_2

Python script

E_2-E_1

Example 3 (performance test)

Stampede2 SKX node (48 cores)

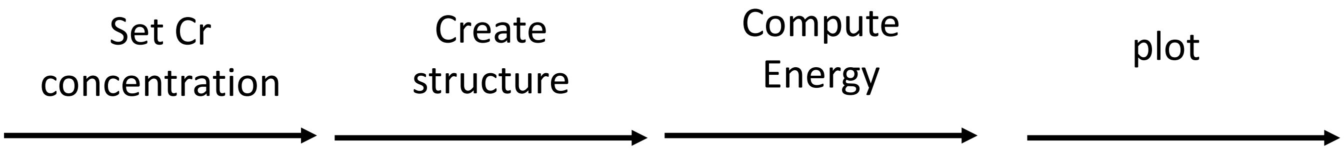


48 workers
Efficiency ~50%

96 mins → 4 mins

Run directly
$time python3 alkane.py

Example 4 (Fe/Cr random alloy, ipywidgets)

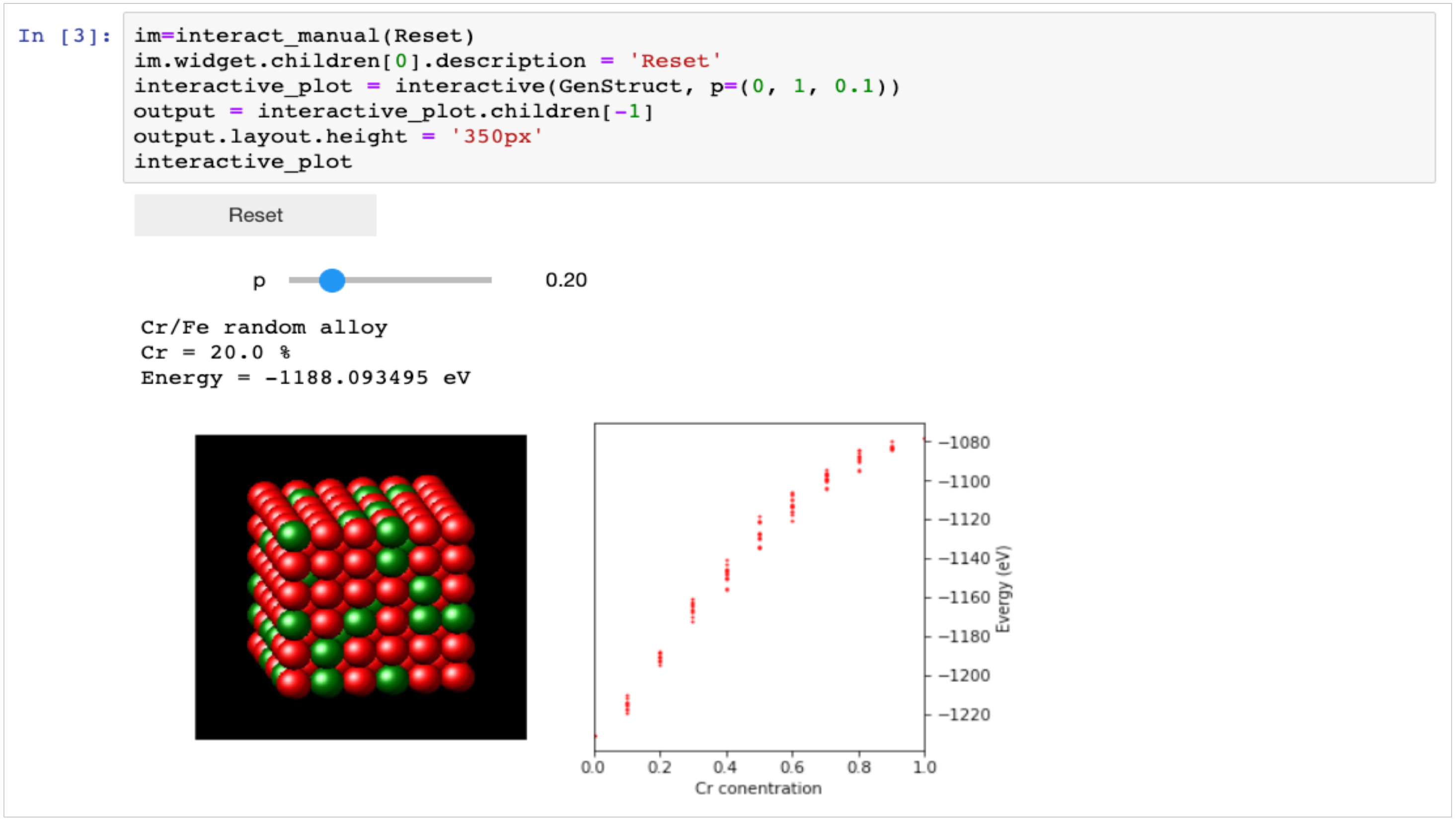


Set Cr concentration

Create structure

Compute Energy

plot



```
In [3]: im=interact_manual(Reset)
        im.widget.children[0].description = 'Reset'
        interactive_plot = interactive(GenStruct, p=(0, 1, 0.1))
        output = interactive_plot.children[-1]
        output.layout.height = '350px'
        interactive_plot
```

```
Cr/Fe random alloy
Cr = 20.0 %
Energy = -1188.093495 eV
```