Parallel Computing with Python

Monte Lunacek

Research Computing, University of Colorado **Boulder**

Outline

- Parallel Computing
  - What is it?
  - Easy parallel example
  - Considerations for using parallelism
- High Throughput Computing IPYthon
  - Configuration
  - Schedules
  - Fault-tolerance
- Conclusion

Parallel Computing

What is Parallel Computing?

Size

- Solve problems that can't fit on a laptop
- Need more than a few GB of RAM
- Need more than a few hundred GB of Disk

Speed

- Same problem, faster
- Makes a bigger problem more feasible

What is Supercomputing?

- Definition changes daily!
- Cluster of computers linked together
- 100x bigger, faster, better than a PC

Janus macbook comparison

Macbook	Janus
2.4 GHz Intel (dual-core)	2.8 GHz Intel (hex-core) X 2 X 1360
3 M cache	12 M cache
8 GB RAM	24 GB RAM

Speed (cpu) ~8000X

Size (memory) ~4000X

Parallel Computing

Shared Memory

- Open MP
- Communication occurs through the shared memory
- Restricted to the number of cores on a node

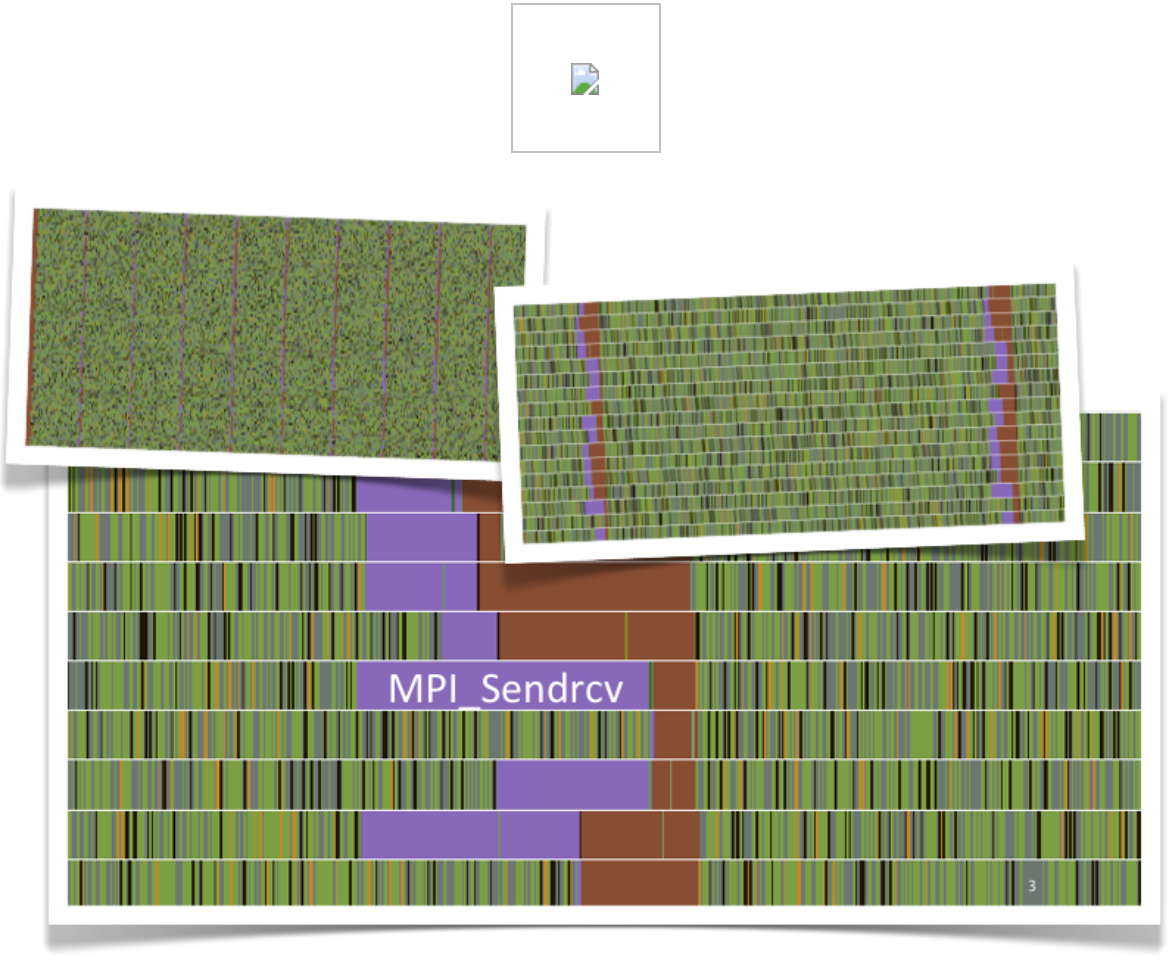
Distributed

- Message Passing Interface (MPI)
- Communication occurs through message passing
- Use thousands of cores

Hybrid

- OpenMP + MPI

Parallel Applications



High Throughout Computing

Completely independent tasks

- Simulations
  - Monte Carlo
  - Parameter scan
  - Uncertainty Quantification
  - Parameter Optimization
- Data Analysis (MapReduce)
- Parallel workflows

Multiprocessing example

Objectives

- Quick and easy parallelism
- Speedup
- Efficiency
- Karp-Flatt Metric
- Amdahl's law

IP[y]: Notebook

Notebooks | Clusters

To import a notebook, drag the file onto the listing below or **click here**.

/projects/molu8455/tutorials/python/notebooks
ipython
multiprocessing

High Throughput Computing with Python

Success Stories

~500,000 simulations on ~7,000 cores with mpi4py (http://mpi4py.scipy.org/)

Wrapped an engineering simulation with f2py (http://www.scipy.org/F2py) and IPython Parallel (http://ipython.org/ipython-doc/dev/parallel/)

David Folch: parameter optimization with Scoop (https://code.google.com/p/scoop/) and DEAP (https://code.google.com/p/deap/)

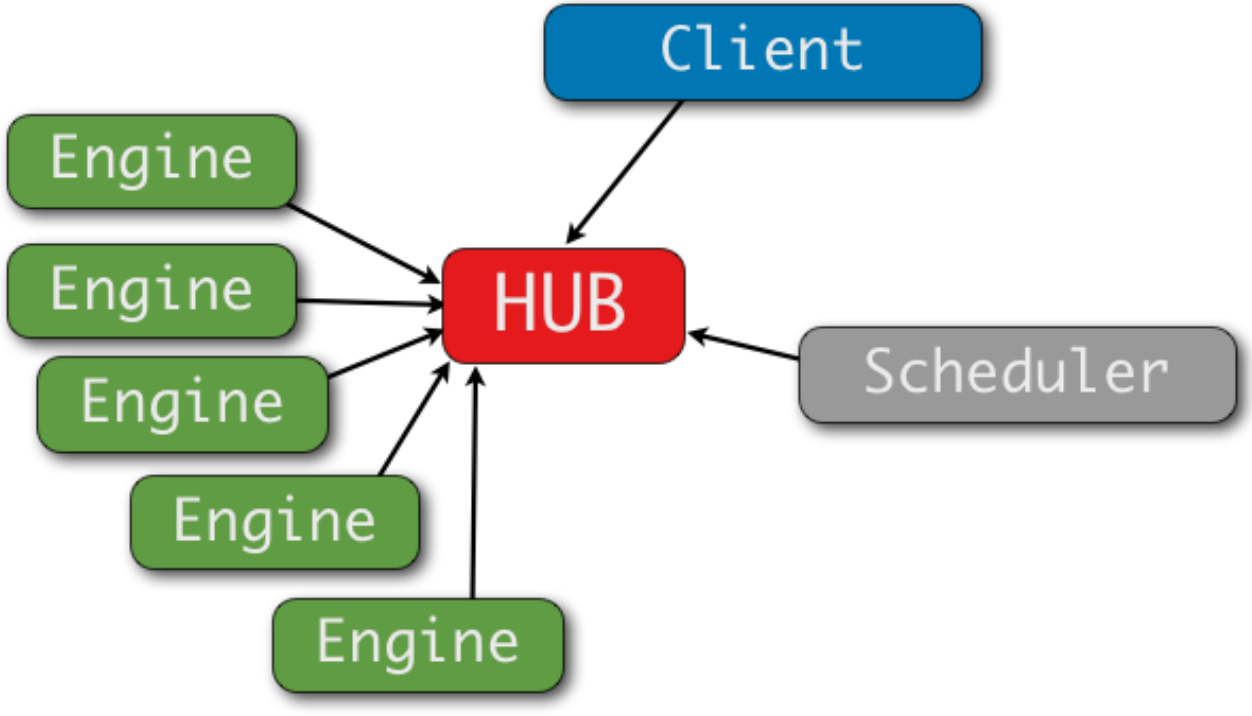
Jon Leff: QIIME IPython Parallel (http://ipython.org/ipython-doc/dev/parallel/)

Ann Deml: MPI tasks with Jinja2 (http://jinja.pocoo.org/docs/)

iPython parallel notebook

Interactive parallel computing on a cluster from a web browser

iPython parallel architecture



Goal: launch the engines and the controller client.

iPython parallel configuration

Launch with SSH and MPI

```
ipython profile create --parallel --profile=ssh
```

```
ipython profile create --parallel --profile=mpi
```

This creates two directories

- `.ipython/profile_ssh` and `.ipython/profile_mpi`

Each directory contains

- ipython_notebook_config.py
- ipcluster_config.py
- ipengine_config.py
- ipcontroller_config.py

iPython parallel configuration

ipython_notebook_config.py

```
c = get_config()
c.NotebookApp.ip = '*'
c.NotebookApp.port = 8888
c.NotebookApp.open_browser = False
```

iPython parallel configuration

profile_ssh/ipcluster_config.py

```
c = get_config()
c.LocalControllerLauncher.controller_args = ["--ip='*'"]
c.IPClusterEngines.engine_launcher_class = 'SSH'
c.SSHEngineLauncher.remote_profile_dir = u'.ipython/profile_ssh
'
c.SSHEngineLauncher.ssh_args = ['-t']
```

```
c.IPClusterStart.n = 12

c.SSHEngineSetLauncher.engines  = {}
filename = os.environ['PBS_NODEFILE']
with open(filename,'r') as file:
    for line in file:
        node = line.split()[0]
        if node not in c.SSHEngineSetLauncher.engines:
            c.SSHEngineSetLauncher.engines[node] = c.IPCluster
Start.n
```

Running iPython parallel

Start the engines

```
ipcluster start --profile='ssh' &
```

Stop the engines

```
ipcluster stop --profile='ssh'
```

Running the notebook on a cluster

Local

```
ssh -Y molu8455@login.rc.colorado.edu
```

Remote

```
qsub -I -X -q janus-admin -l nodes=2:ppn=12
```

```
ipcluster start --profile='ssh' &
```

```
ipython notebook --profile='ssh' --pylab=inline
```

Local

```
ssh -L 2000:node0379:8888 -f -N molu8455@login.rc.colorado.edu
```

IP[y]: Notebook

Notebooks | Clusters

To import a notebook, drag the file onto the listing below or **click here**.

/projects/molu8455/tutorials/python/notebooks
ipython
multiprocessing

IPython Parallel

```
from IPython.parallel import Client
```

Map the values

```
if __name__ == '__main__':

    data = range(200) # tasks
    rc = Client(profile='ssh')
    lview = rc.load_balanced_view()

    results = lview.map(simulation, data)
    results.wait()
```

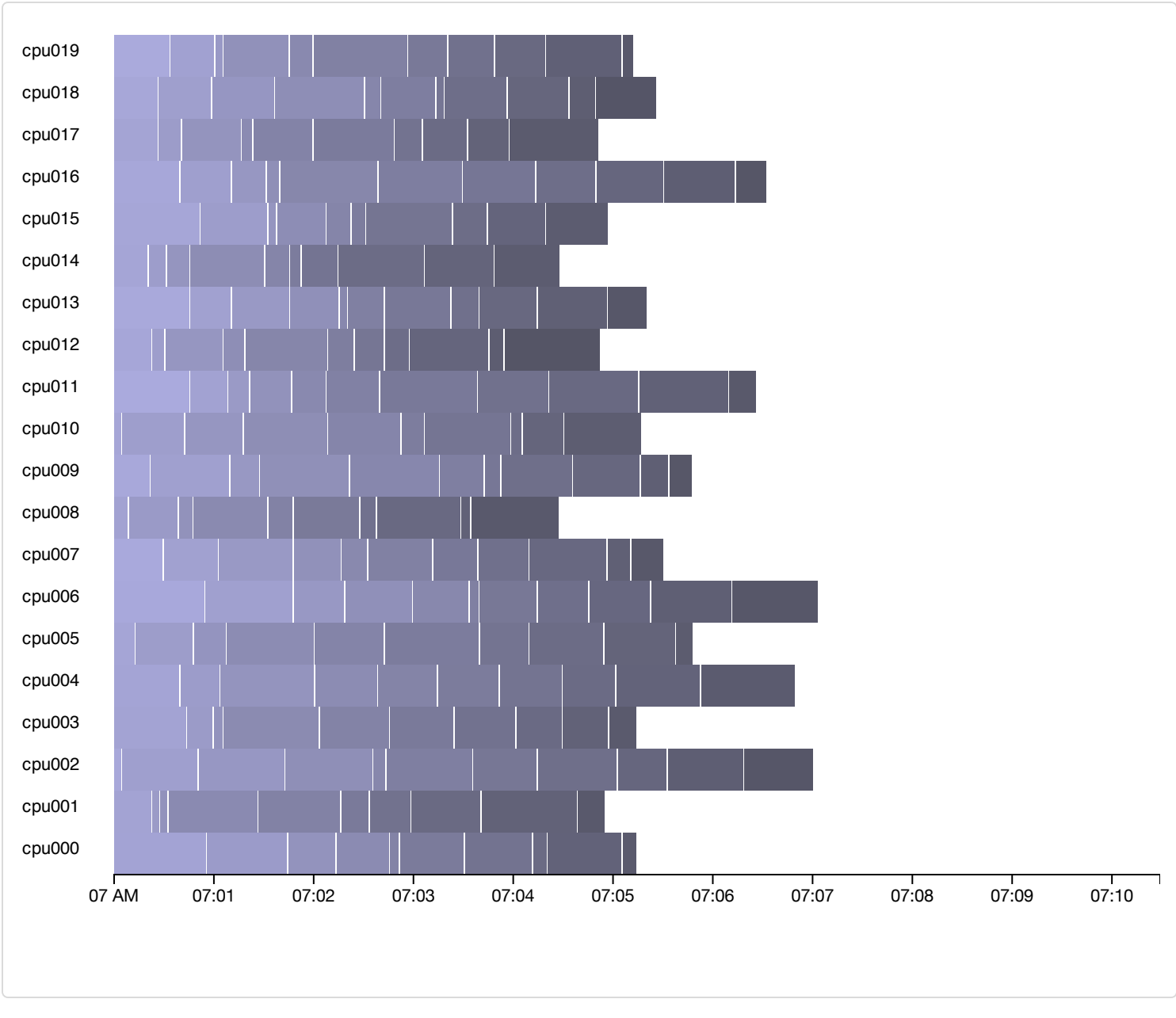
iPython parallel schedules

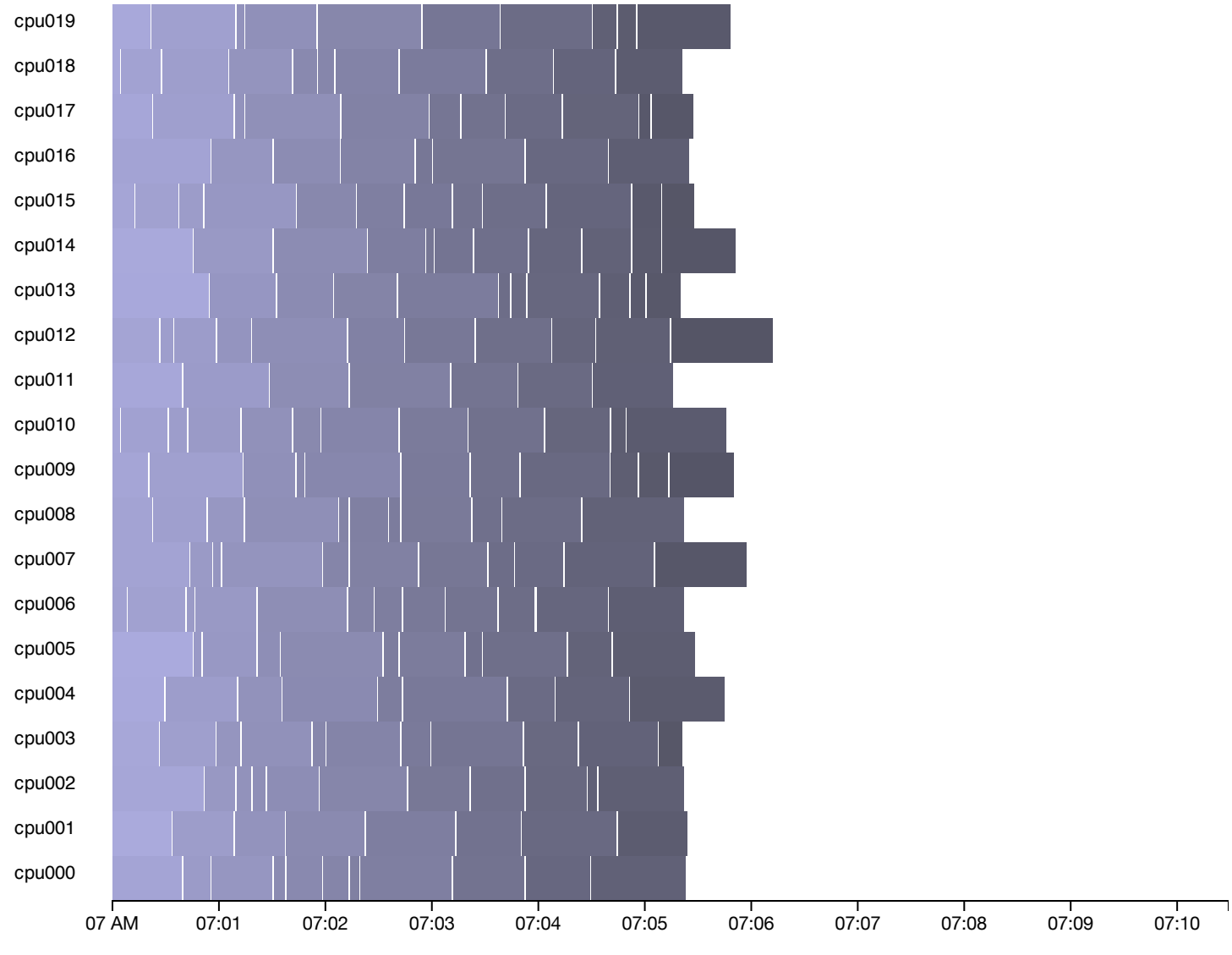
Config

```
# 'lru', 'weighted', 'pure','leastload'
c.TaskScheduler.scheme_name = 'leastload'
c.TaskScheduler.hwm = 1
```

`hwm`

- `0` Static schedule
- `1` Can have at most one waiting
- `2` Can have at most two waiting


cpu019
cpu018
cpu017
cpu016
cpu015
cpu014
cpu013
cpu012
cpu011
cpu010
cpu009
cpu008
cpu007
cpu006
cpu005
cpu004
cpu003
cpu002
cpu001
cpu000
07 AM
07:01
07:02
07:03
07:04
07:05
07:06
07:07
07:08
07:09
07:10


cpu019
cpu018
cpu017
cpu016
cpu015
cpu014
cpu013
cpu012
cpu011
cpu010
cpu009
cpu008
cpu007
cpu006
cpu005
cpu004
cpu003
cpu002
cpu001
cpu000
07 AM
07:01
07:02
07:03
07:04
07:05
07:06
07:07
07:08
07:09
07:10


1949

Tenacious Robustness Test

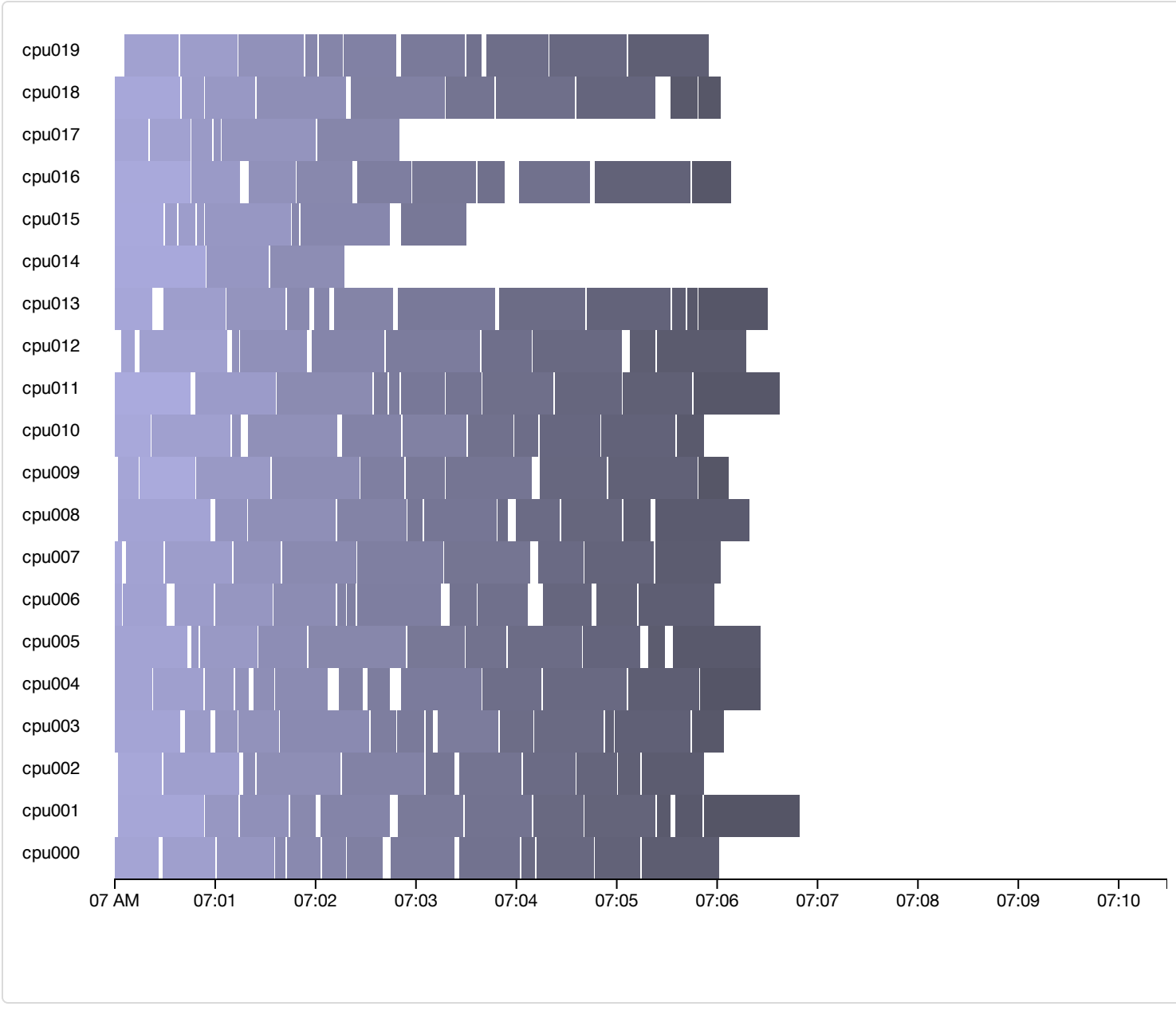
```
@require('time','socket','random',
'IPython.parallel.error.KernelError')
def simulation(x):

   time.sleep(5)
   if random.random() < 0.3:
      raise KernelError
   return {'task':x,
           'host' :socket.gethostname()}
```

Launch 10 nodes

Run **several** tasks

At some point, **kill a node**


cpu019
cpu018
cpu017
cpu016
cpu015
cpu014
cpu013
cpu012
cpu011
cpu010
cpu009
cpu008
cpu007
cpu006
cpu005
cpu004
cpu003
cpu002
cpu001
cpu000
07 AM
07:01
07:02
07:03
07:04
07:05
07:06
07:07
07:08
07:09
07:10

Conclusions

Python makes **supercomputing accessible**

Why parallel computing?

- It's the future!
- Puts you ahead in the game
- More efficient


Jobs
Bash
Static
Dynamic
Fault
cpu019
cpu018
cpu017
cpu016
cpu015
cpu014
cpu013
cpu012
cpu011
cpu010
cpu009
cpu008
cpu007
cpu006
cpu005
cpu004
cpu003
cpu002
cpu001
cpu000