EE3801 Data Engineering Principles

Optimizing Parallel Processing

Parallel Data Processing

Optimizing Parallel Processing

- Lab 6 techniques
- Fine-grained parallel processing
- Resource management
- Maximizing vCPUs
- Lab 7 techniques

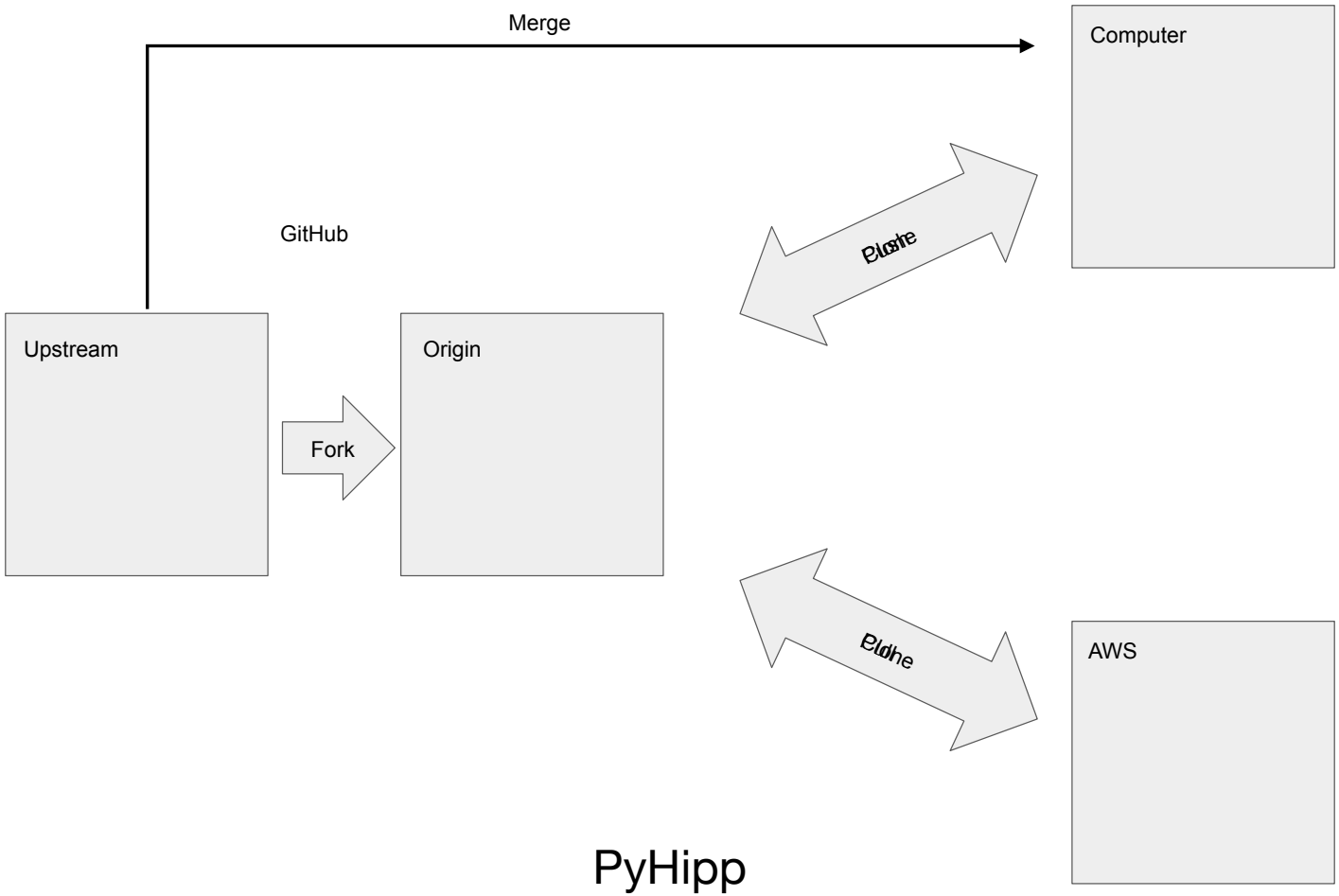
Optimizing Parallel Processing

Lab 6 Techniques

- Bash
  - GitHub command line (git pull, git push, git fetch upstream, etc.)

GitHub Repositories

Forked Repositories



PyHipp

Optimizing Parallel Processing

Lab 6 Techniques

- Bash
  - GitHub command line (git pull, git push, git fetch upstream, etc.)
  - Create shell scripts for frequently performed series of functions (checkfiles.sh)

GitHub Repositories

Lab 6 Techniques

Shell Script

```
#!/bin/bash

echo "Number of hkl files"
find . -name "*.hkl" | grep -v -e spiketrain -e mountains | wc -l

echo "Number of mda files"
find mountains -name "firings.mda" | wc -l

echo "Time taken (s)"
tail pipe-slurm*.out
```

To run the script:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ bash /data/src/PyHipp/checkfiles.sh
```

You can also make the script executable so you can run the script without first calling bash:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ chmod a+x /data/src/PyHipp/checkfiles.sh
(env1) [ec2-user@ip-10-0-5-43 20181105]$ /data/src/PyHipp/checkfiles.sh
```

If the “/data/src/PyHipp” directory is already in the $PATH variable in ~/.bash_profile, you can also also run the script by doing:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ checkfiles.sh
```

Optimizing Parallel Processing

Lab 6 Techniques

- Bash
  - GitHub command line (git pull, git push, git fetch upstream, etc.)
  - Create shell scripts for frequently performed series of functions (checkfiles.sh)
  - Run commands remotely via ssh, e.g.

```
pcluster ssh -i ~/MyKeyPair.pem --region ap-southeast-1 --
cluster-name MyCluster01 squeue
```

- Slurm
  - Use file dependencies to parallelize into 2 jobs (RPLParallel, RPLSplit)

Parallel Data Processing

Parallel Processing

Serial Pipeline

Dependencies	RPLParallel	RPLSplit	RPLLFP	RPLHighPass	Unity	EDFSplit
RPLParallel						
RPLSplit						
RPLLFP		√				
RPLHighPass		√				
Unity	√					
EDFSplit	√					
aligning_objects	√				√	√
raycast					√	√
mountain_batch				√		

Parallel Pipeline

RPLParallel	RPLSplit
Unity	RPLLFP
EDFSplit	RPLHighPass
aligning_objects	mountain_batch
raycast	

Optimizing Parallel Processing

Parallel Processing (2 jobs)

Time

RPLParallel	RPLSplit		
Unity	session01	channel001	rplraw_xxxx.hkl
EDFSplit	sessioneye	channel002	rplraw_xxxx.hkl
aligning_objects		…	…
raycast		channel124	rplraw_xxxx.hkl
	RPLLFP		
	session01	channel001	rpllfp_xxxx.hkl
	sessioneye	channel002	rpllfp_xxxx.hkl
		…	…
		channel124	rpllfp_xxxx.hkl
	RPLHighPass		
	session01	channel001	rplhighpass_xxxx.hkl
	sessioneye	channel002	rplhighpass_xxxx.hkl
		…	…
		channel124	rplhighpass_xxxx.hkl
	mountain_batch		
	session01	channel001	firings.mda
		channel002	firings.mda
		…	…
		channel124	firings.mda

Optimizing Parallel Processing

Lab 6 Techniques

- Bash
  - GitHub command line (git pull, git push, git fetch upstream, etc.)
  - Create shell scripts for frequently performed series of functions (checkfiles.sh)
  - Run commands remotely via ssh, e.g.

```
pcluster ssh -i ~/MyKeyPair.pem --region ap-southeast-1 --
cluster-name MyCluster01 squeue
```

- Slurm
  - Use file dependencies to parallelize into 2 jobs (RPLParallel, RPLSplit)
  - Use directory hierarchy to parallelize into 5 jobs (e.g. session01/array01)

Optimizing Parallel Processing

Coarse-Grained Parallel Processing (5 jobs)

Time

RPLParallel	RPLSplit		RPLSplit		RPLSplit		RPLSplit	
Unity	session01/array01	channel001	session01/array02	channel033	session01/array03	channel065	session01/array04	channel097
EDFSplit	sessioneye/array01	channel002	sessioneye/array02	channel034	sessioneye/array03	channel066	sessioneye/array04	channel098
aligning_objects		…		…		…		…
raycast		channel032		channel064		channel096		channel124
	RPLLFP		RPLLFP		RPLLFP		RPLLFP	
	session01/array01	channel001	session01/array02	channel033	session01/array03	channel065	session01/array04	channel097
	sessioneye/array01	channel002	sessioneye/array02	channel034	sessioneye/array03	channel066	sessioneye/array04	channel098
		…		…		…		…
		channel032		channel064		channel096		channel124
	RPLHighPass		RPLHighPass		RPLHighPass		RPLHighPass	
	session01/array01	channel001	session01/array02	channel033	session01/array03	channel065	session01/array04	channel097
	sessioneye/array01	channel002	sessioneye/array02	channel034	sessioneye/array03	channel066	sessioneye/array04	channel098
		…		…		…		…
		channel032		channel064		channel096		channel124
	mountain_batch		mountain_batch		mountain_batch		mountain_batch	
	session01	channel001	session01	channel033	session01	channel065	session01	channel097
		channel002		channel034		channel066		channel098
		…		…		…		…
		channel032		channel064		channel096		channel124

Optimizing Parallel Processing

Lab 6 Techniques

- Bash
  - GitHub command line (git pull, git push, git fetch upstream, etc.)
  - Create shell scripts for frequently performed series of functions (checkfiles.sh)
  - Run commands remotely via ssh, e.g.

```
pcluster ssh -i ~/MyKeyPair.pem --region ap-southeast-1 --
cluster-name MyCluster01 squeue
```

- Slurm
  - Use file dependencies to parallelize into 2 jobs (RPLParallel, RPLSplit)
  - Use directory hierarchy to parallelize into 5 jobs (e.g. session01/array01)
  - Use slurm parameter --cpus-per-task=5 to increase memory available to individual jobs

Optimizing Parallel Processing

Parallel Processing (5 jobs)

Increasing memory per job

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    2   queue1    rplpl ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
    3   queue1      rs1 ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
    4   queue1      rs2 ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
    5   queue1      rs3 ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
    6   queue1      rs4 ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
```

64 GB RAM

40 GB RAM

```
#SBATCH --time=24:00:00   # walltime
#SBATCH --ntasks=1   # number of processor cores (i.e. tasks)
#SBATCH --nodes=1   # number of nodes
#SBATCH --cpus-per-task=5            # number of processors per task
#SBATCH -J "rs1"   # job name
```

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    7   queue1    rplpl ec2-user  R       4:31      1 queue1-dy-r52xlarge-1
    8   queue1      rs1 ec2-user  R       4:31      1 queue1-dy-r52xlarge-1
    9   queue1      rs2 ec2-user  R       1:25      1 queue1-dy-r52xlarge-2
   10   queue1      rs3 ec2-user  R       1:25      1 queue1-dy-r52xlarge-3
   11   queue1      rs4 ec2-user  R       1:25      1 queue1-dy-r52xlarge-4
```

Optimizing Parallel Processing

Lab 6 Techniques

- Bash
  - GitHub command line (git pull, git push, git fetch upstream, etc.)
  - Create shell scripts for frequently performed series of functions (checkfiles.sh)
  - Run commands remotely via ssh, e.g.

```
pcluster ssh -i ~/MyKeyPair.pem --region ap-southeast-1 --
cluster-name MyCluster01 squeue
```

- Slurm
  - Use file dependencies to parallelize into 2 jobs (RPLParallel, RPLSplit)
  - Use directory hierarchy to parallelize into 5 jobs (e.g. session01/array01)
  - Use slurm parameter --cpus-per-task=5 to increase memory available to individual jobs
  - Use slurm job dependencies to run jobs in sequence

Data Processing on AWS

Cluster Deletion

consol_jobs.sh

```
#!/bin/sh
temp1=($(squeue))
cmd1="sbatch --dependency=afterok:"
counter1=0
for i in "${temp1[@]}"; do
       if [[ "$i" == "queue1" ]]; then
                      id1=${temp1[$counter1-1]}
                      cmd1="${cmd1}${id1}:"
       fi
       counter1=$((counter1+1))
done
cmd1=${cmd1::-1}
cmd1="${cmd1} /data/src/PyHipp/ec2snapshot.sh"

echo $($cmd1)
eval $cmd1
```

```
(env1) [ec2-user@ip-10-0-9-92 data]$ for i in "${temp1[@]}"; do echo $i; done
JOBID
PARTITION
NAME
USER
ST
TIME
NODES
NODELIST(REASON)
233
queue1
rplpl
ec2-user
CF
0:02
1
compute-dy-r5large-1
234
queue1
rse
ec2-user
CF
0:02
1
compute-dy-r5large-1
(env1) [ec2-user@ip-10-0-9-92 data]$
```

cmd1=“sbatch —dependency=afterok:

cmd1=“sbatch —dependency=afterok:233:

cmd1=“sbatch —dependency=afterok:233:234:

cmd1=“sbatch —dependency=afterok:233:234

cmd1=“sbatch —dependency=afterok:233:234 /data/src/PyHipp/ec2snapshot.sh

Optimizing Parallel Processing

Lab 6 Techniques

- Bash
  - GitHub command line (git pull, git push, git fetch upstream, etc.)
  - Create shell scripts for frequently performed series of functions (checkfiles.sh)
  - Run commands remotely via ssh, e.g.

    ```
    pcluster ssh -i ~/MyKeyPair.pem --region ap-southeast-1 --
    cluster-name MyCluster01 squeue
    ```
- Slurm
  - Use file dependencies to parallelize into 2 jobs (RPLParallel, RPLSplit)
  - Use directory hierarchy to parallelize into 5 jobs (e.g. session01/array01)
  - Use slurm parameter --cpus-per-task=5 to increase memory available to individual jobs
  - Use slurm job dependencies to run jobs in sequence
- AWS
  - Use AWS Lambda to run scripts without servers
  - Use AWS Lambda, EventBridge, and small EC2 instance to terminate head node automatically

Optimizing Parallel Processing

Lab 6 Techniques

EC2

```
[ec2-user@ip-172-31-47-12 ~]$ ./update_snapshot.sh data 2 MyCluster01

/home/ec2-user/cluster-config.yaml

    SnapshotId: snap-0a8bb6f9c13b437ed

[ec2-user@ip-172-31-47-12 ~]$ pcluster create-cluster --cluster-configuration
      ~/cluster-config.yaml --cluster-name MyCluster01
```

Computer

```
(aws) shihcheng@SC-M1-MBA ~ % update_snapshot.sh data 2 MyCluster01

/Users/shihcheng/cluster-config.yaml

    SnapshotId: snap-0e4cf11324a17c5e3

(aws) shihcheng@SC-M1-MBA ~ % pcluster create-cluster
      --cluster-configuration ~/cluster-config.yaml --cluster-name MyCluster01
```

Parallel Data Processing

Optimizing Parallel Processing

- Lab 6 techniques
- Fine-grained parallel processing
- Resource management
- Maximizing vCPUs
- Lab 7 techniques

Data Processing on AWS

Slurm Script (Called from /data/picasso/20181105)

```
#!/bin/bash

# Submit this script with: sbatch <this-filename>

#SBATCH —time=24:00:00   # walltime
#SBATCH --ntasks=1   # number of processor cores (i.e. tasks)
#SBATCH --nodes=1   # number of nodes
#SBATCH -J "pipe"   # job name

## /SBATCH -p general # partition (queue)
#SBATCH -o pipe-slurm.%N.%j.out # STDOUT
#SBATCH -e pipe-slurm.%N.%j.err # STDERR

# LOAD MODULES, INSERT CODE, AND RUN YOUR PROGRAMS HERE

python -u -c "import PyHipp as pyh; \
import DataProcessingTools as DPT; \
import os; \
import time; \
t0 = time.time(); \
print(time.localtime()); \
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLParallel, saveLevel=1); \
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLSplit, channel=[9, 31, 34, 56, 72, 93, 119, 120]); \
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLLFP, saveLevel=1); \
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLHighPass, saveLevel=1); \
DPT.objects.processDirs(dirs=None, objtype=pyh.Unity, saveLevel=1); \
pyh.EDFSplit(); \
os.chdir('session01'); \
pyh.aligning_objects(); \
pyh.raycast(1); \
DPT.objects.processDirs(level='channel', cmd='import PyHipp as pyh; from PyHipp import mountain_batch; mountain_batch.mountain_batch(); from PyHipp
import export_mountain_cells; export_mountain_cells.export_mountain_cells();'); \
print(time.localtime()); \
print(time.time()-t0);"

aws sns publish --topic-arn arn:aws:sns:ap-southeast-1:123456789012:awsnotify --message "JobDone"
```

Parallel Data Processing

Parallel Processing

Serial Pipeline

Dependencies	RPLParallel	RPLSplit	RPLLFP	RPLHighPass	Unity	EDFSplit
RPLParallel						
RPLSplit						
RPLLFP		√				
RPLHighPass		√				
Unity	√					
EDFSplit	√					
aligning_objects	√				√	√
raycast					√	√
mountain_batch				√		

Parallel Pipeline

RPLParallel	RPLSplit
Unity	RPLLFP
EDFSplit	RPLHighPass
aligning_objects	mountain_batch
raycast	

Optimizing Parallel Processing

Parallel Processing (2 jobs)

Time

RPLParallel	RPLSplit		
Unity	session01	channel001	rplraw_xxxx.hkl
EDFSplit	sessioneye	channel002	rplraw_xxxx.hkl
aligning_objects		…	…
raycast		channel124	rplraw_xxxx.hkl
	RPLLFP		
	session01	channel001	rpllfp_xxxx.hkl
	sessioneye	channel002	rpllfp_xxxx.hkl
		…	…
		channel124	rpllfp_xxxx.hkl
	RPLHighPass		
	session01	channel001	rplhighpass_xxxx.hkl
	sessioneye	channel002	rplhighpass_xxxx.hkl
		…	…
		channel124	rplhighpass_xxxx.hkl
	mountain_batch		
	session01	channel001	firings.mda
		channel002	firings.mda
		…	…
		channel124	firings.mda

Optimizing Parallel Processing

Coarse-Grained Parallel Processing (5 jobs)

Time ↓

RPLParallel	RPLSplit		RPLSplit		RPLSplit		RPLSplit	
Unity	session01	channel001	session01	channel033	session01	channel065	session01	channel097
EDFSplit	sessioneye	channel002	sessioneye	channel034	sessioneye	channel066	sessioneye	channel098
aligning_objects		…		…		…		…
raycast		channel032		channel064		channel096		channel124
	RPLLFP		RPLLFP		RPLLFP		RPLLFP	
	session01	channel001	session01	channel033	session01	channel065	session01	channel097
	sessioneye	channel002	sessioneye	channel034	sessioneye	channel066	sessioneye	channel098
		…		…		…		…
		channel032		channel064		channel096		channel124
	RPLHighPass		RPLHighPass		RPLHighPass		RPLHighPass	
	session01	channel001	session01	channel033	session01	channel065	session01	channel097
	sessioneye	channel002	sessioneye	channel034	sessioneye	channel066	sessioneye	channel098
		…		…		…		…
		channel032		channel064		channel096		channel124
	mountain_batch		mountain_batch		mountain_batch		mountain_batch	
	session01	channel001	session01	channel033	session01	channel065	session01	channel097
		channel002		channel034		channel066		channel098
		…		…		…		…
		channel032		channel064		channel096		channel124

Optimizing Parallel Processing

Fine-Grained Parallel Processing

Time

RPLSplit		
session01	channel001	rplraw_xxxx.hkl
	channel002	rplraw_xxxx.hkl
	...	...
	channel032	rplraw_xxxx.hkl

RPLLFP	RPLHighPass+mountain_batch
rpllfp_xxxx.hkl	rplhighpass_xxxx.hkl+firings.mda
rpllfp_xxxx.hkl	rplhighpass_xxxx.hkl+firings.mda
rpllfp_xxxx.hkl	rplhighpass_xxxx.hkl+firings.mda

Optimizing Parallel Processing

Fine-Grained Parallel Processing

Time

RPLParallel	RPLSplit	
Unity	sessioneye	channel001
EDFSplit		channel002
aligning_objects		…
raycast		channel124
	RPLLFP	
	sessioneye	channel001
		channel002
		…
		channel124
	RPLHighPass	
	sessioneye	channel001
		channel002
		…
		channel124

RPLSplit		RPLLFP	RPLHPS	RPLSplit		RPLLFP	RPLHPS	RPLSplit		RPLLFP	RPLHPS	RPLSplit		RPLLFP	RPLHPS
session01	channel001			session01	channel033			session01	channel065			session01	channel097		
	…				…				…				…		
	channel032				channel064				channel096				channel124		

Jobs: 6 + 110 + 110 = 226

Optimizing Parallel Processing

Fine-Grained Parallel Processing (226 jobs)

Just-in-time job submission

DPT.objects.processDirs(dirs=None, objtype=pyh.RPLSplit, channel=[*range(1,33)], SkipHPC=False, HPCScriptsDir = '/data/src/PyHipp/', SkipLFP=False, SkipHighPass=False, SkipSort=False);

rplsplit.py

~/Dropbox/Work/Python/PyHipp/PyHipp/rplsplit.py main (no function selected)

```python
            os.chdir(channelDir)
            print('Calling RPLRaw for channel {:03d}'.format(channelNumber))
            rplraw.RPLRaw(analogData = data, analogInfo = analogInfo, saveLevel = 1)
            if self.args['SkipHPC']:
                if not self.args['SkipLFP']:
                    print('Calling RPLLFP for channel {:03d}'.format(channelNumber))
                    rpllfp.RPLLFP(saveLevel = 1)
                if not self.args['SkipHighPass']:
                    print('Calling RPLHighPass for channel {:03d}'.format(channelNumber))
                    rplhighpass.RPLHighPass(saveLevel = 1)
                if DPT.levels.get_level_name('session', os.getcwd()) != 'sessioneye':
                    if not self.args['SkipSort']:
                        print('Calling Mountain Sort for channel {:03d}'.format(channelNumber))
                        # mountain_batch()
                        # export_mountain_cells()
            else:
                if 'HPCScriptsDir' not in kwargs.keys():
                    kwargs['HPCScriptsDir'] = ''
                if not self.args['SkipLFP']:
                    print('Adding RPLLFP slurm script for channel {:03d} to job queue'.format(channelNumber))
                    os.system('sbatch ' + kwargs['HPCScriptsDir'] + 'rpllfp-slurm.sh')
                if not self.args['SkipHighPass']:
                    if not self.args['SkipSort']:
                        print('Adding RPLHighPass and Mountain Sort slurm script for channel {:03d} to job queue'.format(channelNumber))
                        os.system('sbatch '+ kwargs['HPCScriptsDir'] + 'rplhighpass-sort-slurm.sh')
                    else:
                        print('Adding RPLHighPass slurm script for channel {:03d} to job queue'.format(channelNumber))
                        os.system('sbatch ' + kwargs['HPCScriptsDir'] + 'rplhighpass-slurm.sh')
```

L: 153 C: 1 Python Unicode (UTF-8) Unix (LF) Saved: 11/16/20, 5:13:37 PM 6,438 / 687 / 153 100%

Optimizing Parallel Processing

Fine-Grained Parallel Processing (226 jobs)

bin — ec2-user@ip-10-0-9-92:/data/src/PyHipp — -bash — 98×60

```
729   compute   rpllfp ec2-user PD       0:00      1 (Priority)
730   compute   rplhps ec2-user PD       0:00      1 (Priority)
731   compute   rpllfp ec2-user PD       0:00      1 (Priority)
732   compute   rplhps ec2-user PD       0:00      1 (Priority)
281   compute      rs1 ec2-user  R      29:10      1 compute-dy-z1d2xlarge-1
334   compute   rplhps ec2-user  R    1:16:55      1 compute-dy-z1d2xlarge-7
336   compute   rplhps ec2-user  R    1:16:43      1 compute-dy-z1d2xlarge-7
338   compute   rplhps ec2-user  R    1:16:19      1 compute-dy-z1d2xlarge-5
340   compute   rplhps ec2-user  R    1:15:37      1 compute-dy-z1d2xlarge-3
342   compute   rplhps ec2-user  R    1:15:32      1 compute-dy-z1d2xlarge-3
344   compute   rplhps ec2-user  R    1:14:42      1 compute-dy-z1d2xlarge-2
346   compute   rplhps ec2-user  R    1:14:42      1 compute-dy-z1d2xlarge-5
348   compute   rplhps ec2-user  R    1:13:59      1 compute-dy-z1d2xlarge-6
350   compute   rplhps ec2-user  R    1:13:49      1 compute-dy-z1d2xlarge-3
352   compute   rplhps ec2-user  R    1:13:02      1 compute-dy-z1d2xlarge-4
354   compute   rplhps ec2-user  R    1:12:50      1 compute-dy-z1d2xlarge-4
356   compute   rplhps ec2-user  R    1:09:21      1 compute-dy-z1dxlarge-1
359   compute   rplhps ec2-user  R    1:11:44      1 compute-dy-z1dxlarge-1
360   compute   rplhps ec2-user  R      29:13      1 compute-dy-z1d2xlarge-6
362   compute   rplhps ec2-user  R      50:40      1 compute-dy-z1d2xlarge-6
366   compute   rplhps ec2-user  R      47:38      1 compute-dy-z1d2xlarge-7
368   compute   rplhps ec2-user  R      47:11      1 compute-dy-z1d2xlarge-1
370   compute   rplhps ec2-user  R      55:25      1 compute-dy-z1d2xlarge-5
372   compute   rplhps ec2-user  R      55:25      1 compute-dy-z1d2xlarge-5
374   compute   rplhps ec2-user  R      55:25      1 compute-dy-z1d2xlarge-5
376   compute   rplhps ec2-user  R      49:20      1 compute-dy-z1d2xlarge-6
378   compute   rplhps ec2-user  R      49:20      1 compute-dy-z1d2xlarge-6
380   compute   rplhps ec2-user  R      49:00      1 compute-dy-z1d2xlarge-5
382   compute   rplhps ec2-user  R      13:59      1 compute-dy-z1d2xlarge-2
384   compute   rplhps ec2-user  R      11:03      1 compute-dy-z1d2xlarge-2
392   compute   rplhps ec2-user  R      28:54      1 compute-dy-z1d2xlarge-4
394   compute   rplhps ec2-user  R      34:28      1 compute-dy-z1dxlarge-1
396   compute   rplhps ec2-user  R      33:53      1 compute-dy-z1d2xlarge-2
398   compute   rplhps ec2-user  R      46:52      1 compute-dy-z1dxlarge-1
400   compute   rplhps ec2-user  R      33:53      1 compute-dy-z1d2xlarge-2
402   compute   rplhps ec2-user  R      29:55      1 compute-dy-z1d2xlarge-6
404   compute   rplhps ec2-user  R      27:03      1 compute-dy-z1d2xlarge-5
405   compute   rpllfp ec2-user  R       0:01      1 compute-dy-z1d2xlarge-2
412   compute   rplhps ec2-user  R      24:59      1 compute-dy-z1d2xlarge-1
416   compute   rplhps ec2-user  R       9:18      1 compute-dy-z1d2xlarge-6
418   compute   rplhps ec2-user  R       8:00      1 compute-dy-z1d2xlarge-6
419   compute   rpllfp ec2-user  R       7:56      1 compute-dy-z1d2xlarge-7
420   compute   rplhps ec2-user  R       7:56      1 compute-dy-z1d2xlarge-7
421   compute   rpllfp ec2-user  R       7:56      1 compute-dy-z1d2xlarge-7
422   compute   rpllfp ec2-user  R       7:56      1 compute-dy-z1d2xlarge-7
423   compute   rplhps ec2-user  R      21:36      1 compute-dy-z1d2xlarge-2
424   compute   rplhps ec2-user  R      20:48      1 compute-dy-z1d2xlarge-1
425   compute   rpllfp ec2-user  R       7:56      1 compute-dy-z1d2xlarge-7
426   compute   rplhps ec2-user  R      11:25      1 compute-dy-z1d2xlarge-2
427   compute   rpllfp ec2-user  R       7:13      1 compute-dy-z1d2xlarge-5
429   compute   rplhps ec2-user  R       4:21      1 compute-dy-z1d2xlarge-4
430   compute   rplhps ec2-user  R       4:21      1 compute-dy-z1d2xlarge-4
431   compute   rpllfp ec2-user  R       4:21      1 compute-dy-z1d2xlarge-4
432   compute   rplhps ec2-user  R       4:21      1 compute-dy-z1d2xlarge-4
433   compute   rpllfp ec2-user  R       4:21      1 compute-dy-z1d2xlarge-4
434   compute   rplhps ec2-user  R       3:02      1 compute-dy-z1d2xlarge-3
435   compute   rpllfp ec2-user  R       3:02      1 compute-dy-z1d2xlarge-3
436   compute   rpllfp ec2-user  R       3:02      1 compute-dy-z1d2xlarge-3
437   compute   rplhps ec2-user  R       3:02      1 compute-dy-z1d2xlarge-3
438   compute   rplhps ec2-user  R       3:02      1 compute-dy-z1d2xlarge-3
```

Cannot actually run 64 jobs as the Master Node uses 2 CPUs and the EC2 instance also uses 2 CPUs

Optimizing Parallel Processing

Fine-Grained Parallel Processing (226 jobs)

- Greatly increases network file system requests from the compute nodes
- Need more RAM in the Master Node to prevent bottlenecks
- EBS volume with general purpose SSD helps

Amazon EBS Volumes

With Amazon EBS, you pay only for what you use. The pricing for Amazon EBS volumes is listed below

General Purpose SSD (gp2) Volumes	$0.12 per GB-month of provisioned storage
Provisioned IOPS SSD (io2) Volumes	$0.138 per GB-month of provisioned storage AND $0.072 per provisioned IOPS-month
Provisioned IOPS SSD (io1) Volumes	$0.138 per GB-month of provisioned storage AND $0.072 per provisioned IOPS-month
Throughput Optimized HDD (st1) Volumes	$0.054 per GB-month of provisioned storage
Cold HDD (sc1) Volumes	$0.03 per GB-month of provisioned storage

Parallel Data Processing

Optimizing Parallel Processing

- Lab 6 techniques
- Fine-grained parallel processing
- Resource management
- Maximizing vCPUs
- Lab 7 techniques

Optimizing Parallel Processing

Resource Management

- Third party spike sorting software:
  - https://github.com/flatironinstitute/mountainsort
- Contains code to lock file:
  - /data/miniconda3/envs/env1/etc/mountainlab/database/processor_specs.json
- Only 1 job can run at a time
- Causes problems with parallel processing

Optimizing Parallel Processing

Resource Management

- Create separate conda environments with its own copy of processor_specs.json file
  - x=0; while [$x -le 63]; do echo $x; conda create --name cenv$x --clone env1 --copy; ((x++)); done
- Need to use “--copy” flag as default is to link to existing environment

Optimizing Parallel Processing

Resource Management

- RPLHighPass+mountain_batch slurm script:

```
#!/bin/bash

# Submit this script with: sbatch <this-filename>

#SBATCH --time=24:00:00   # walltime
#SBATCH --ntasks=1   # number of processor cores (i.e. tasks)
#SBATCH --nodes=1   # number of nodes
#SBATCH -J "rplhps"   # job name

## /SBATCH -p general # partition (queue)
#SBATCH -o rplhps-slurm.%N.%j.out # STDOUT
#SBATCH -e rplhps-slurm.%N.%j.err # STDERR

# LOAD MODULES, INSERT CODE, AND RUN YOUR PROGRAMS HERE
/data/miniconda3/bin/conda init
source ~/.bashrc
conda activate cenv0

python -u -c "import PyHipp as pyh; import time; pyh.RPLHighPass(saveLevel = 1);
 from PyHipp import mountain_batch; mountain_batch.mountain_batch(); from PyHipp
 import export_mountain_cells; export_mountain_cells.export_mountain_cells(); print(time.localtime());"

conda deactivate
```

Optimizing Parallel Processing

Resource Management

- Need way to make sure each job uses a different environment
- Want to avoid having multiple copies of highpass-sort-slurm.sh using different environments
- Create way to check out and check in different environments

Optimizing Parallel Processing

Resource Management

- Create Python program to manage environments: envlist.py
- Usage 1: envlist.py env_prefix number_of_environments
- Create list of environment names in Python
  - envlist.py cenv 64
  - [cenv0, cenv1, cenv2, … cenv63]
  - saved in /data/picasso/envlist.hkl

Optimizing Parallel Processing

Resource Management

- Usage 2: envlist.py
- Use file locking to make sure only 1 job can access list at a time to get environment name
  - removes first name from the list, and returns it
  - envarg=`envlist.py` Runs the command within `` and sets the output to the variable envarg
  - echo $envarg
  - cenv0
  - [cenv1, cenv2, … cenv63]

Optimizing Parallel Processing

Resource Management

- Usage 3: envlist.py env_name_to_return
- Use file locking to make sure only 1 job can access list at a time to return environment name
  - initial list: [cenv1, cenv2, … cenv63]
  - envlist.py cenv0
  - [cenv1, cenv2, … cenv63, cenv0]
  - appends name to the end

Optimizing Parallel Processing

Resource Management

```
#!/bin/bash

# Submit this script with: sbatch <this-filename>

#SBATCH --time=24:00:00   # walltime
#SBATCH --ntasks=1   # number of processor cores (i.e. tasks)
#SBATCH --nodes=1   # number of nodes
#SBATCH -J "rplhps"   # job name

## /SBATCH -p general # partition (queue)
#SBATCH -o rplhps-slurm.%N.%j.out # STDOUT
#SBATCH -e rplhps-slurm.%N.%j.err # STDERR

# LOAD MODULES, INSERT CODE, AND RUN YOUR PROGRAMS HERE
/data/miniconda3/bin/conda init
source ~/.bashrc
envarg=`/data/src/PyHipp/envlist.py`
conda activate $envarg

python -u -c "import PyHipp as pyh; import time; pyh.RPLHighPass(saveLevel = 1);
 from PyHipp import mountain_batch; mountain_batch.mountain_batch(); from PyHipp
 import export_mountain_cells; export_mountain_cells.export_mountain_cells(); print(time.localtime());"

conda deactivate
/data/src/PyHipp/envlist.py $envarg
```

Parallel Data Processing

Optimizing Parallel Processing

- Lab 6 techniques
- Fine-grained parallel processing
- Resource management
- Maximizing vCPUs
- Lab 7 techniques

Optimizing Parallel Processing

Maximizing vCPUs

vCPU Number	1 Compute Node Type	Mixed Compute Nodes
1	r5.2xlarge 1	r5.2xlarge 1
...		
8		
9	r5.2xlarge 2	r5.2xlarge 2
...		
16		
17	r5.2xlarge 3	r5.2xlarge 3
...		
24		
25	r5.2xlarge 4	r5.2xlarge 4
...		
32		
33	r5.2xlarge 5	r5.2xlarge 5
...		
40		
41	r5.2xlarge 6	r5.2xlarge 6
...		
48		
49	r5.2xlarge 7	r5.2xlarge 7
...		
56		
57		r5.xlarge 1
58		
59		
60		
61	Head Node	Head Node
62		
63	EC2 Instance	EC2 Instance
64		

2xlarge: 8 vCPUs
xlarge: 4 vCPUs

Optimizing Parallel Processing

Maximizing vCPUs

cluster-config.yaml

```
Scheduling:
  Scheduler: slurm
  SlurmQueues:
  - Name: queue1
    ComputeResources:
    - Name: r5-2xlarge
      InstanceType: r5.2xlarge
      MinCount: 0
      MaxCount: 10
    - Name: r5a-xlarge
      InstanceType: r5a.xlarge
      MinCount: 0
      MaxCount: 1
```

Parallel Data Processing

Optimizing Parallel Processing

- Lab 6 techniques
- Fine-grained parallel processing
- Resource management
- Maximizing vCPUs
- Lab 7 techniques

Optimizing Parallel Processing

Lab 7 Techniques

- Bash
  - Use multiple conda environments to work around software with file locking
  - Create and use Python program to manage conda environments
  - Use sort command to organize output
    - `find session01 -name "hps*out" -or -name "lfp*out" | xargs tail -n 1 | sort`
  - Use cut command to extract output
    - `echo $filename`
    - `./session01/array04/channel123/rplhighpass_b59f.hkl`
    - `echo $filename | cut -d "/" -f 1-4`
    - `./session01/array04/channel123`

Optimizing Parallel Processing

Lab 7 Techniques

- Bash
  - Use comm command to find missing output
    - `find . -name "channel*" | grep -v -e eye -e mountain | sort > chs.txt`
    - `find . -name "rplhighpass*hkl" | grep -v -e eye | sort | cut -d "/" -f 1-4 > hps.txt`
    - `comm -23 chs.txt hps.txt`
  - Using for-loops
    - `cwd=`pwd`; for i in `comm -23 chs.txt hps.txt`; do echo $i; cd $i; sbatch /data/src/PyHipp/rplhighpass-sort-slurm.sh; cd $cwd; done`
    - `for i in 2018110[12]; do echo $i; cd $i; bash /data/src/PyHipp/pipe2a.sh; cd ..; done`
    - `for i in 20180??? 201810??; do echo $i; cd $i; sbatch /data/src/PyHipp/rplparallel-slurm.sh; cd ..; done`

Optimizing Parallel Processing

Lab 7 Techniques

- Slurm
  - Just-in-time job submission to allow jobs to start when data is ready
  - Creating many jobs allows full use of all available vCPUs
- AWS
  - Using combination of instance types for compute nodes to maximize vCPUs

Optimizing Parallel Processing

Lab Instructions

- Lab 7 Instructions:
  - https://ee3801.github.io/Lab7/instruction.html
- Submit to Canvas (Lab 7->Lab 7A & Lab 7->Lab 7B)
- Submit in PDF format
- Name the files Lab7A_YourName.pdf and Lab7B_YourName.pdf
- Part A due on Monday (Oct 30) 2 pm
- Part B due on Wednesday (Nov 1) 9 pm

Questions?