HARVARD
Faculty of Arts and Sciences
RESEARCH COMPUTING

Running Biology Workflows on Odyssey

Bob Freeman, PhD
Dir. Research Technology Operations (HBS)
formerly RC Facilitator (FASRC)

robert_freeman@harvard.edu
@DevBizInfoGuy

With thanks to and contributions from:
Amir Karger & Kris Holton, HMS RITG

Goals

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Objectives

- Review the basics of our compute cluster Odyssey
- Use compute efficiently for typically inefficient programs or workflows
- Use Unix & programming techniques to do bioinformatics more efficiently
    - Bash: variables, for loops,
    - SLURM: job arrays, job dependencies
    - Parallelization approaches

Strategically

- "Work smarter, better, faster"
- … and to Think Differently
- Enable you to be successful with your research!

Slide deck avail as PDF after class

Overview

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

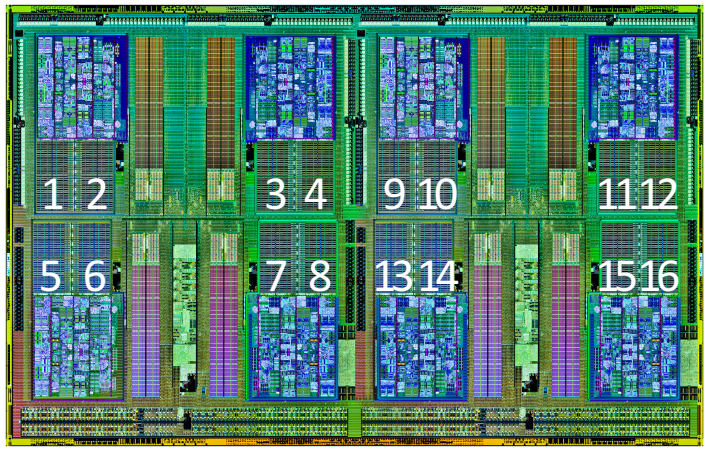
1. Cluster basics
   - Access, storage, software, partitions, resources
2. Using software
   - Module system, Java & Python, Updating local modules/packages
3. Running multicore (multiCPU) jobs & Scaling considerations
4. Pleasantly parallelizing tasks (e.g. getting BLAST results fast!)
5. Checkpointing to save (re)compute time
6. SLURM scripts for common programs & typical workflows
7. Troubleshooting & Getting help

General configuration...

HARVARD
Faculty of Arts and Sciences
RESEARCH COMPUTING

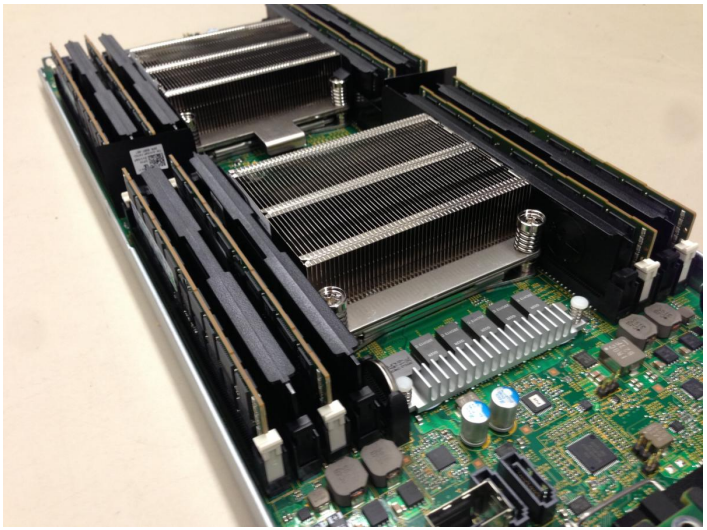
The basic computational unit in a cluster is a **CPU core**

- Each core runs one process, a *average* job
- Most compute nodes have 64 cores arranged on 4 CPUs (16 cores/CPU)
- Thus, most nodes run 64 batch job processes



A **typical compute node** is configured:

- 64 cores
- 256 GB RAM, or ~4 GB RAM/core
- 2 network cards: Infiniband (intraconnect) & xGb connections (interconnect)
- Small, local hard disk/SSD for boot and local /scratch



All cores on a node share all other resources of the node: memory, network bandwidth, etc.

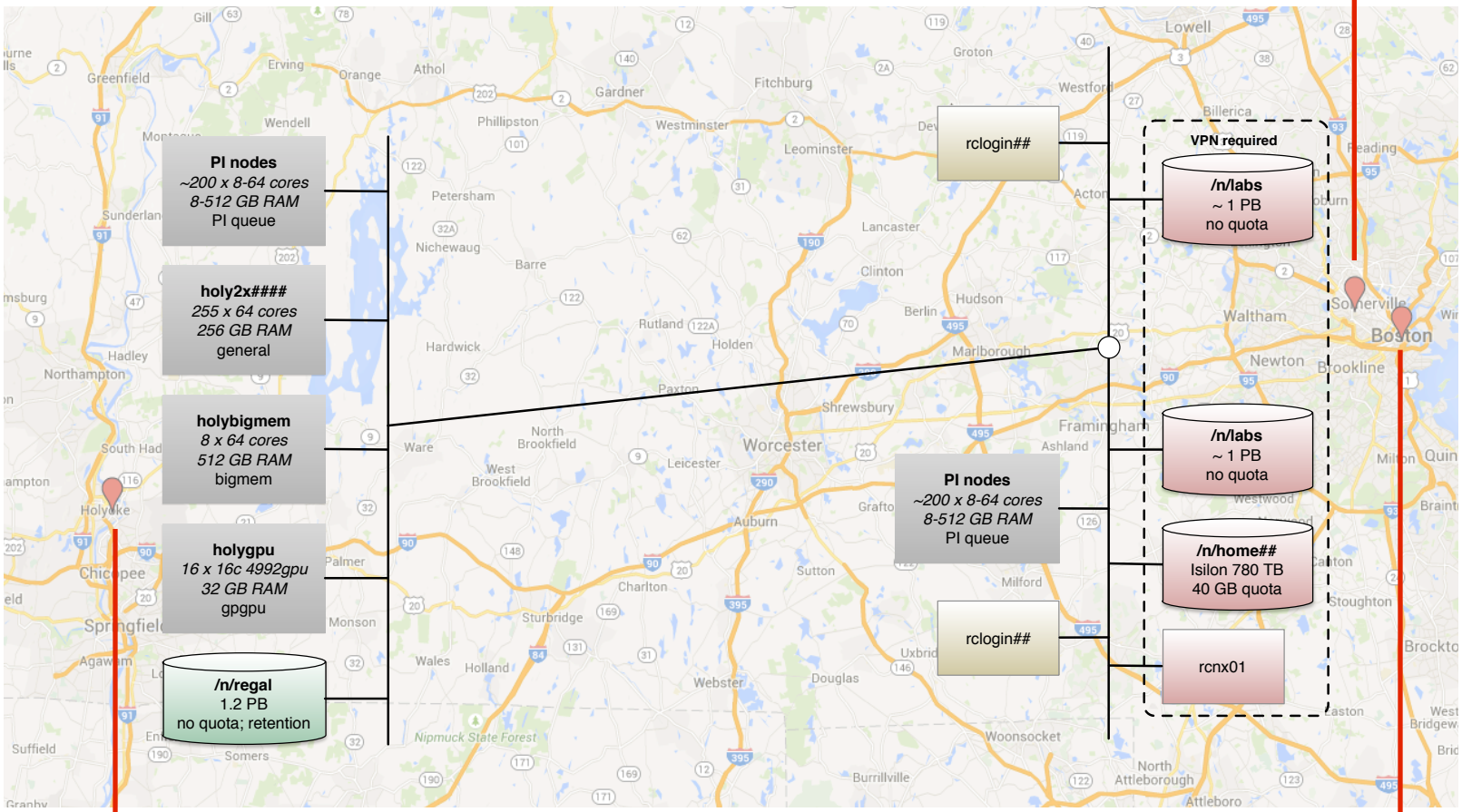
Thus, how you use these resources affects the other 63 jobs on that compute node

What is Odyssey?

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Compute nodes/disk are located in 3 data centers:

60 Oxford Street



MGHPCC, Holyoke MA

1 Summer Street

Topology may effect the efficiency of work

Typical Workflow

1. Login in to Odyssey
    a. Land on a login (head) node, appropriate for light work only
2. Copy/upload/download some files
3. Load appropriate software
4. Get interactive session
5. Test your program/script interactively to ensure it runs properly
6. Test run in batch: create batch file & submit to SLURM
    a. Continue working in the foreground while waiting for results
7. Scale up as necessary (10s, 100s, 1000s)
    a. With *caveats*: proper file placement, # cores, etc.

Data Security: Storage/Use of HRCI

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Confidential data is defined as

> "Information about a person or an entity that, if disclosed, could reasonably be expected to place the person or the entity at risk of criminal or civil liability, or to be damaging to financial standing, employability, reputation or other interests."

See HU's IT Security pages for methods for handling & network access: http://security.harvard.edu/

Under no circumstances should HRCI data be stored on RC storage without consultation. Storage must be specifically designed for HRCI data: http://fasrc.us/data_hrci

Working on restricted datasets (e.g. dbGAP or CMS)?

- Set up a follow-up appointment with RC
- Requires discussion and training on facilities you/PI can access under the University Data Usage Agreement (DUA) process
  - DUAs: Important documents signed by PI and Asst. Dean of RC to protect important datasets
  - Each dataset / DUA must be handled in a unique manner
- http://vpr.harvard.edu/pages/harvard-research-data-security-policy

Your PI must brief you on training for these datasets and how they need to be controlled

Transferring files to/from Odyssey

GUI client Filezilla for all platforms

- Configure according to http://fasrc.us/configfilezilla to avoid 2FA problems

Command line tools `scp` or `rsync`

- `rsync` is best for resuming transfers or transferring only changed file parts

Download data using `curl` or `wget`

- is available on all compute nodes, though web proxy needed for HRCI setups

Or by mountings disk shares. Please see http://fasrc.us/mountdisks

Examples:

```
# copy file in current dir to Odyssey home folder
scp somefile.txt rcuser@login.rc.fas.harvard.edu:~

# copy folder in current dir & contents to Odyssey home folder
rsync -av myfolder rcuser@login.rc.fas.harvard.edu:~

# download FASTA sequence from NCBI
wget "http://www.ncbi.nlm.nih.gov/nuccore/L03535.1?report=fasta&log$=seqview&format=text"
```

Common Filesystems

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

	Type	Size	Avail?	Mount Desktops?	Backup?[1]	Retention?	I/O profile
`/n/home##`	NFS	40 GB (hard limit)	all nodes	Y	Y	N	low
`/n/labfs#`	NFS	1 TB free (new labs) contact for costs	all nodes	Y	Y[2]	N	low
`/scratch`	local	250 GB/node (~4 GB/core)	all nodes	N	N	Y[3]	high
`/n/regal`	Lustre	1.2 PB	all nodes	N[4]	N	90-days[5]	high

[1]Backup methods differ. See http://fasrc.us/faqrecovery for more information.
[2]Lab disks shares are typically backed up unless noted.
[3]Files usually deleted when job finished. Please clean up your own mess, though.
[4]Can use file transfer methods to stage data.
[5]Retention is typically run at maintenance times. Areas can be exempted for common data (e.g. NCBI Genbank at `/n/regal/informatics_public`). Contacts us.

Common Filesystems: /scratch

Using local `/scratch`:

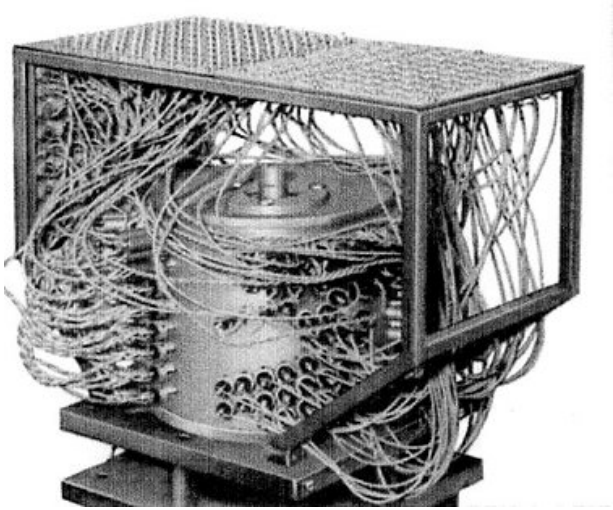
- 250 GB slice on each compute node, so there's about 4 GB disk space/job
- Is currently underutilized, so more space may be available (check `sbatch` options)
- Can see speedup of 2x – 3x, depending on pattern of file read/writes
- Since is local to each node, must use it *during* your job:

```
...
start_dir=$PWD
mkdir -p /scratch/$USER/$SLURM_JOBID
cd /scratch/$USER/$SLURM_JOBID

# do your work while writing temp files here
...

# copy files back and return from whence we came
cp –r results/ $start_dir/
cd $start_dir

# now cleanup
rm -rf /scratch/$USER/$SLURM_JOBID
```



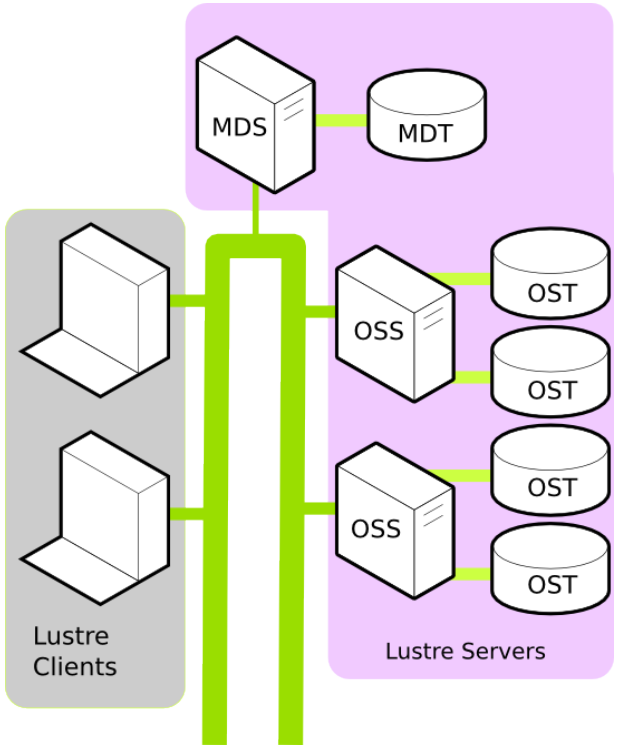
Common Filesystems: /n/regal

Using `/n/regal`:

- Most work should be done here, especially for ~≥ 10 simultaneous jobs
- No space restrictions, but files > 90 days old deleted (usually at maintenance)
- Can stage files prior to job by typical copy/rsync commands or Filezilla
- Remember to copy results back to home or lab shares for permanent storage

A couple more things to remember:

- Shared lab areas can be exempt from retention. Contact us.
- Public data sets can also be staged here – no need to keep your own copy
- NCBI, EMBL, UCSC data is stored at `/n/regal/informatics_public`:
  - FASTA data, BLAST databases, Bowtie2 indexes
- Contact us if you'd like to add more to this location



Login vs Interactive Nodes

Terminal sessions to `login.rc` puts you on one of several login nodes

- This gateway to the cluster has limited entry points, so..
- Only non-CPU-intensive work is appropriate: `cp`, `mv`, `nano`, `rsync`, etc.
- *Reminder*: rcnx01 and holynx01 are login nodes

Don't compute here, instead

- Submit a batch job (background task) to SLURM, or
- Request an interactive session (foreground task) on a compute node:

```
srun --pty --x11=first --mem 1000 -p interact -t 0-6:00 -n 1 -N 1 /bin/bash
```

srun: foreground
sbatch: background

Resources that you wish to request from SLURM

Script or program `/bin/bash` == shell

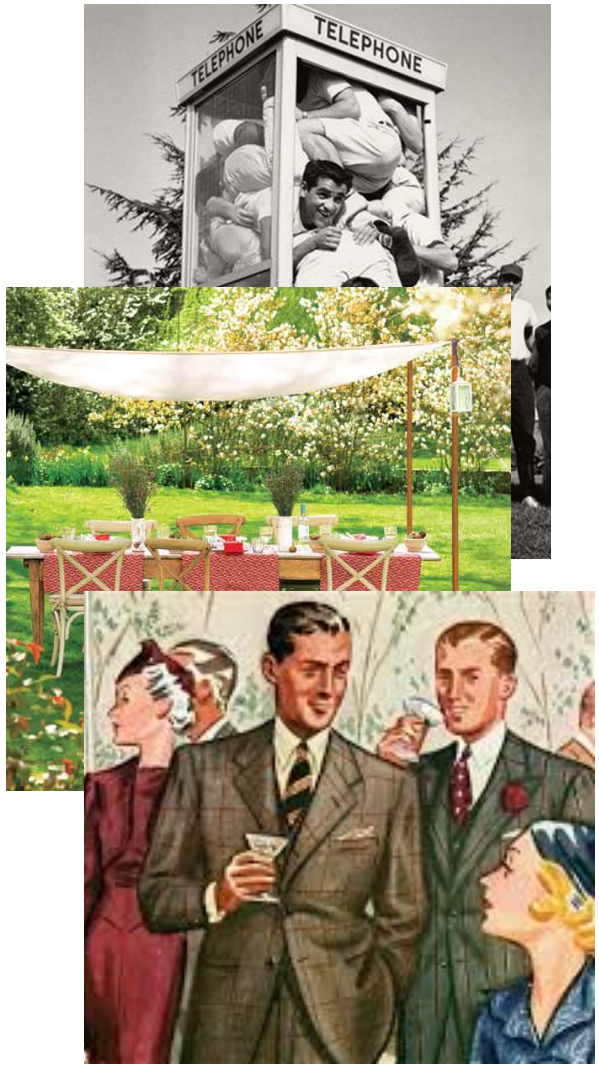
Choosing Resources: How?

Choosing resources is like attending a party:

- You need to RSVP the number of guests you intend to bring
  Request the resources you intend to use
- Extra guests: there's not enough food and drink for everyone
  CPU/disk overage: all jobs including your will run more slowly
  RAM/time overage: your job will be killed
- Too few: an unhappy host and wasted $$ / effort
  CPU/RAM: resources are wasted as they cannot be used by anyone else
  All: your job becomes harder to schedule

You also want to be polite:

- Stay the appropriate amount of time…
  Try to approximate your resource use with some padding for safety
- Don't slip in, drink & eat, and leave within minutes
  Try to avoid jobs that start and complete within minutes; especially in large numbers



Choosing Resources: Time & Memory

Time:

- Determined by your test runs during an interactive session
- Or if trying in batch, over-ask first, then reduce time on later runs
- Due to scheduler overhead, jobs should do at least 5 - 10 min of work

Memory:

- Check software docs for memory requirements
- If none stated, over-ask and do a trial run (via `srun` or `sbatch`)
- use `sacct` command to get post-run job info:

```
# RAM                                    requested/used!!
sacct -j JOBID --format=JobID,Elapsed,ReqMem,MaxRSS
```

"Never use a piece of bioinformatics software for the first time without looking to see what command-line options are available and what default parameters are being used"

-- acgt.me · by Keith Bradnam

Choosing Resources: Partitions

Name	Length	Size (cores)	Memory/node	Usage
interact	3 days	512 (8 nodes)	256 GB	all interactive work
serial_requeue	7 days*	30K+	varies (512 GB max)	best for single core jobs; or small numbers of cores for short durations; schedules best as hits all parts of the cluster
general	7 days	~14K	256 GB	large # of cores; MPI jobs; jobs sensitive to pre-emption (pipelines)
unrestricted	no limit	512	256 GB	all jobs with no time limit
bigmem	7 days	512	512 GB	jobs requiring >256 GB RAM (restricted access)
(private)	no limit	varies	256 GB typical	lab-specific partitions

Note: SLURM can schedule to quickest of two partitions with `-p partion1,partion2`

Choosing Resources: Partitions

interact

- Use for foreground, interactive sessions up to 2 days
- You can request multiple cores or large RAM
- Limit the number of active, interactive session to 5 or less

(private)

- PI-specific partitions, usually named after the lab
- Access is automatic, by group membership

bigmem

- For work where each job requires > 256 GB RAM
- Accessible only by request

general

- For all large core #, long, or MPI jobs, or jobs sensitive to pre-emption
- When busy, typically will take tens of minutes or hours to schedule
- Requesting full nodes may take >1 day for your job to schedule

Choosing Resources: Partitions

serial_requeue

- Recommended partition for single-core jobs; or jobs using up to 8 cores lasting up to approx. 6 – 12 hrs
- Most 'powerful' as hits every core on the cluster, including private compute
- Dispatches within seconds to minutes

But the downside…

- Jobs may be pre-empted (killed) if originally scheduled on a private node and the node owner submits work. **Your job is automatically rescheduled**

To mitigate this…

- Use the sbatch option `--open-mode=append` for your `-e` and `-o` log files
- Use `%N` (in addition to `%j`) in log file names to indicate what host your job ran on.
- If you append output, ensure that you zero your data files at the start of the job, to ensure that any files left over from a previous, partial run are removed.
- Structure your command flow so that you skip over any work already done. This allows your re-run job to pick up from where the pre-empted one left off.

Overview

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

1. Cluster basics
   - Access, storage, software, partitions, resources
2. Using software
   - Module system, Java & Python, Updating local modules/packages
3. Running multicore (multiCPU) jobs & Scaling considerations
4. Pleasantly parallelizing tasks (e.g. getting BLAST results fast!)
5. Checkpointing to save (re)compute time
6. SLURM scripts for common programs & typical workflows
7. Troubleshooting & Getting help

Lmod Module System

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Software is loaded incrementally using modules, to set up your shell environment (e.g. `PATH`, `BLASTDB`, and other environment variables)

Using the Harvard-modified, TACC module system `Lmod`:

- New system is still opt-in. Strongly suggested reading: http://fasrc.us/rclmod

```
source new-modules.sh                  # for opt-in folks
module load fastqc/1.0.0-fasrc01       # recommended!!

module load fastqc                     # most recent version

module-query fastqc                    # recommended!!
module-query --full-text fastqc        # gives more detail

module spider fastqc                   # find details on software
module avail 2>&1 | grep -i fastqc     # find titles/defaults
```

Software search capabilities similar to `module-query` are also available on the RC Portal!

Module loads best placed in SLURM batch scripts:

- Keeps your interactive working environment simple
- Is a record of your research workflow (reproducible research!)
- Keep `.bashrc` module loads sparse, lest you run into software and library conflicts

Java Programs

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

FASRC will no longer install Java apps and Python scripts

- they will probably not work correctly in read-only sections of the cluster
- Python scripts use `pip-install` or `python setup.py` & doesn't work with some setups

For Java…

- Download the `*.jar` files or the install files into a home or lab `apps/` or `bin/` directory
- Include the java `CLASSPATH` statement in your `.bashrc`, OR
- Set up a bash environment variable in your `.bashrc`
- Call the software using the `java` command, pointing to the appropriate routine

```
cd ~
mkdir -p apps; cd apps
wget http://…longURL…/Trimmomatic-0.36.zip
unzip Trimmomatic-0.36.zip
ln -s Trimmomatic-0.36 trimmomatic
echo "export TRIMMOMATIC=$HOME/apps/trimmomatic" >> ~/.bashrc

# in SLURM script or on command line…
module load java/1.8.0_45-fasrc01
cd ~/myFASTQdirectory; mkdir trimmed

# minHeap (-Xms) and maxHeap (-Xmx) options are optional but useful in some cases!!
java -Xms128m -Xmx4g -jar $TRIMMOMATIC/trimmomatic-0.32.jar SE -threads 1 \
  PSG177_TGACCA.fastq.gz trimmed/PSG177_TGACCA.fastq
  ILLUMINACLIP:TruSeq3-PE.fa:2:40:15 LEADING:3 TRAILING:3 \
  SLIDINGWINDOW:4:20 MINLEN:25
```

Python Programs

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

For Python we recommend:

- Use the standard `module load python/2.7.6-fasrc01` for pulling in default modules
- Use the Anaconda environment for customizing modules & versions
- Multiple custom environments can be set up for home or lab folders (e.g. development or production code). Check conda options for 'non-standard' locations

```
module load python/2.7.6-fasrc01                    # for any python version
conda create -n ENV_NAME --clone="$PYTHON_HOME"      # created at ~/env/ENV_NAME

# load in environment
source activate ENV_NAME                             # local environment is default

# load in new package
conda install MYPACKAGE                              # replace MYPACKAGE

# update a specific package
# 'remove' first if you hit update problems
conda update MYPACKAGE

# other methods after source activate
pip install MYPACKAGE
python setup.py install MYPACKAGE
```


R / Python / Perl Libraries

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

We've already covered Python libraries. R and Perl are much simpler!

R: Use the R_LIBS_USER environment variable…

```
# load R, default packages, & set local install dir (must already exist!)
module load R_packages/3.2.0-fasrc01
...
# can put this in .bashrc, but best to do this after R module load
export R_LIBS_USER=$HOME/apps/R:$R_LIBS_USER
#
R
install.packages('deplyr')                              # inside R
```

For Perl, we can do the same thing!

```
# load Perl, default modules, & set local install dir (must already exist!)
module load perl-modules/5.10.1-fasrc11

# can put these in your .bashrc
export LOCALPERL=$HOME/apps/perl
export PERL5LIB=$LOCALPERL:$LOCALPERL/lib/perl5:$PERL5LIB
export PERL_MM_OPT="INSTALL_BASE=$LOCALPERL"
export PERL_MB_OPT="--install_base $LOCALPERL"
export PATH="$LOCALPERL/bin:$PATH"

# and now do easy, local installs with cpan
cpan FASTAParse
```


Installing Your Own Software

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

- Users can compile software in their `/home` or `/groups` directories, where they have permission
- Binaries just require "unzipping" (ie `tar -zxvf *.tgz`)

- Common compiling libraries are found as modules:

```
gcc/4.8.2-fasrc01
intel/15.0.0-fasrc01
boost/1.59.0-fasrc01

gcc/4.8.2-fasrc01 llvm/3.5.1-fasrc02

gcc/4.8.2-fasrc01 openmpi/1.8.3-fasrc02
gcc/4.8.2-fasrc01  mvapich2/2.0-fasrc03
```

- Common math libraries are also present but may be compiler-specific

```
gcc/4.8.2-fasrc01  intel-mkl/11.0.0.079-fasrc01
```


Installing a Binary

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

- Users can place software in their /home or /groups directories, where they have permission
- Binaries just require "unzipping" (ie tar -zxvf .tgz)

```
cd ~
mkdir -p apps; cd apps
wget http://…longURL…/myBinarySoftware-1.0.tgz

tar -xvf myBinarySoftware-1.0.tgz
ls myBinarySoftware-1.0
ln -s myBinarySoftware-1.0 myBinarySoftware

# add to .bashrc; assumes all binaries are in top level of this folder
echo "export PATH=$HOME/apps/myBinarySoftware:$PATH" >> ~/.bashrc
export PATH=$HOME/apps/myBinarySoftware:$PATH

# and now run it!
myBinary
```


Installing from Source

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

- Users can compile software in their /home or /groups directories, where they have permission
- Source requires untaring and (often) selecting a compiler

```
cd ~
mkdir -p apps; cd apps
wget http://…longURL…/myBinarySoftware-1.0.tgz

tar -xvf myBinarySoftware-1.0.tgz
ls myBinarySoftware-1.0
ln -s myBinarySoftware-1.0 myBinarySoftware

cd myBinarySoftware
less README

module load gcc/4.8.2-fasrc01
./configure --prefix=$HOME/apps/myBinarySoftware
make
make install

# add to .bashrc; assumes all binaries are in top level of this folder
echo "export PATH=$HOME/apps/myBinarySoftware:$PATH" >> ~/.bashrc
export PATH=$HOME/apps/myBinarySoftware:$PATH

# and now run it!
myBinary
```


Overview

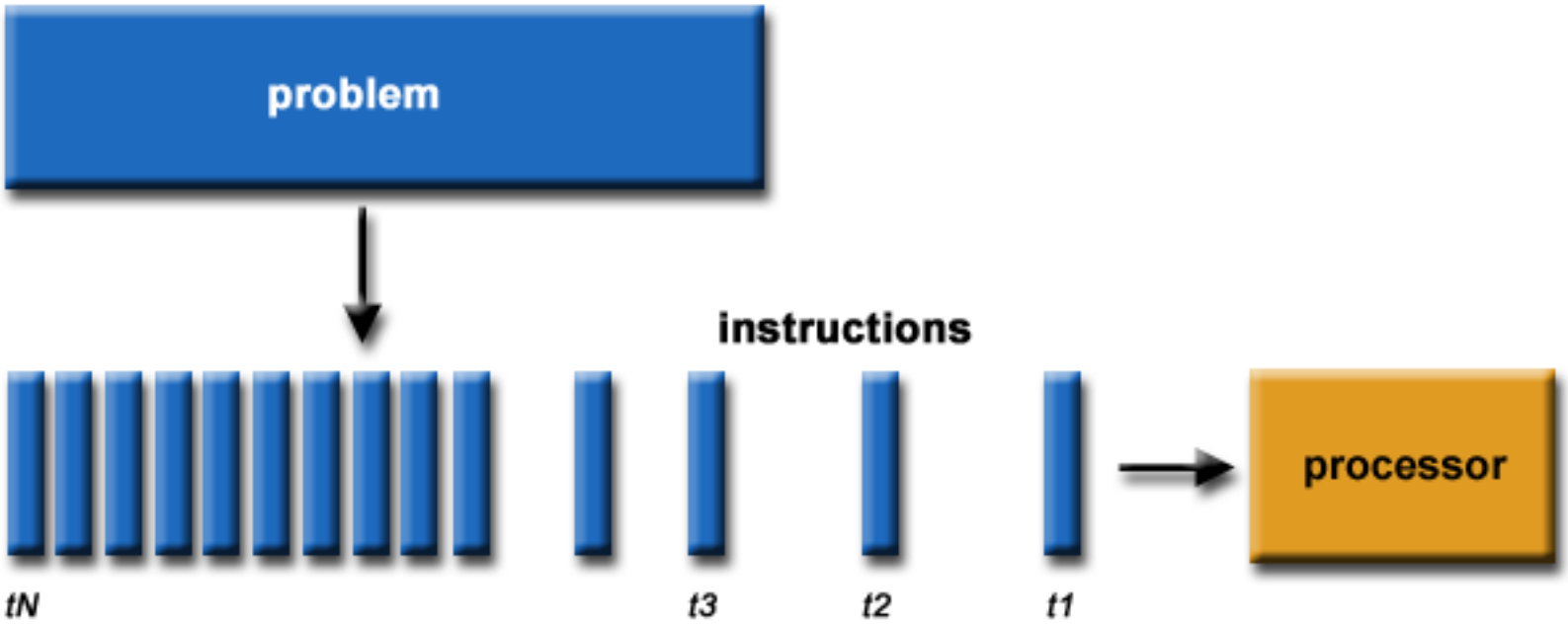
HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

1. Cluster basics
   - Access, storage, software, partitions, resources
2. Using software
   - Module system, Java & Python, Updating local modules/packages
3. Running multicore (multiCPU) jobs & Scaling considerations
4. Pleasantly parallelizing tasks (e.g. getting BLAST results fast!)
5. Checkpointing to save (re)compute time
6. SLURM scripts for common programs & typical workflows
7. Troubleshooting & Getting help

Serial vs Multicore Approaches

Traditionally, software has been written for serial computers

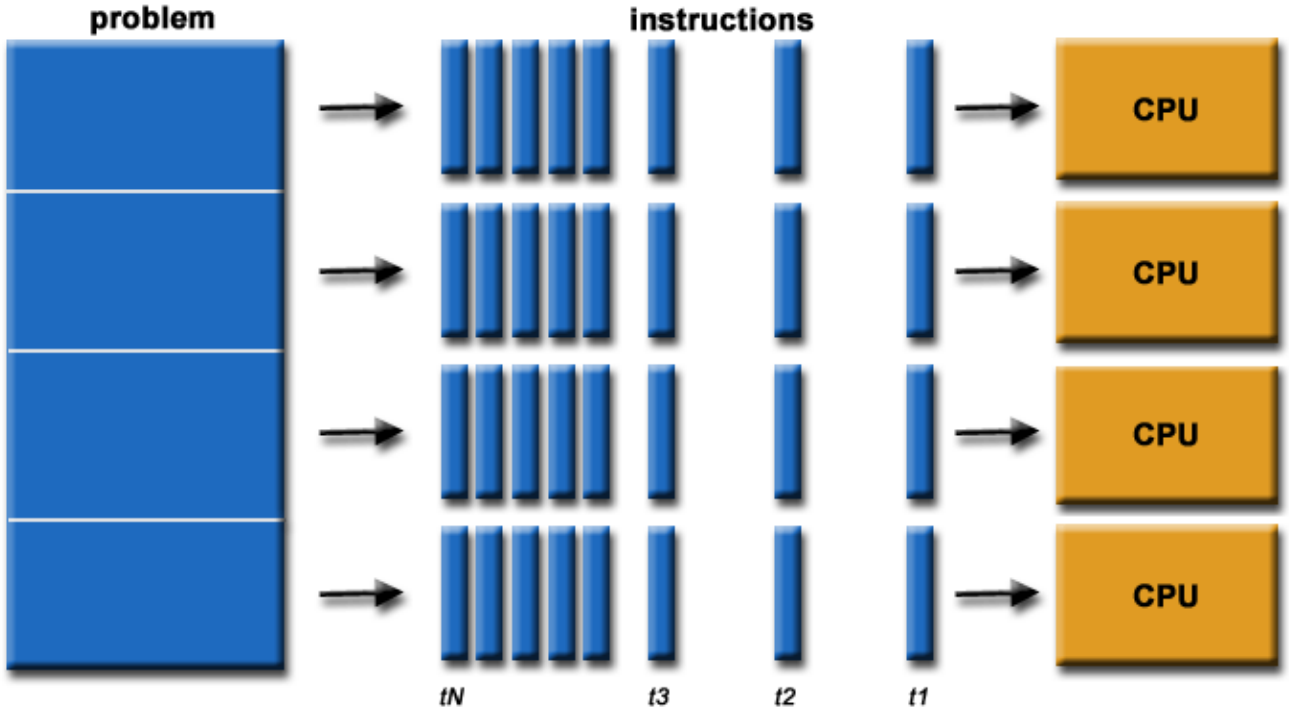
- To be run on a **single computer** having a single Central Processing Unit (**CPU**)
- Problem is broken into a discrete set of instructions
- Instructions are executed **one after the other**
- **One one instruction** can be executed at any moment in time



Serial vs Multicore Approaches

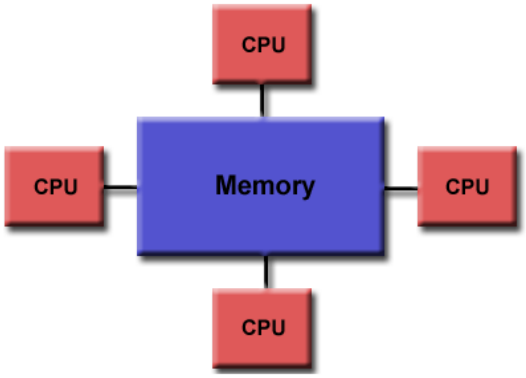
In the simplest sense, parallel computing is the **simultaneous use of multiple compute resources** to solve a computational problem:

- To be run using **multiple CPUs**
- A problem is broken into discrete parts that can be **solved concurrently**
- Each part is further broken down to a series of instructions
- Instructions from each part **execute simultaneously on different CPUs**

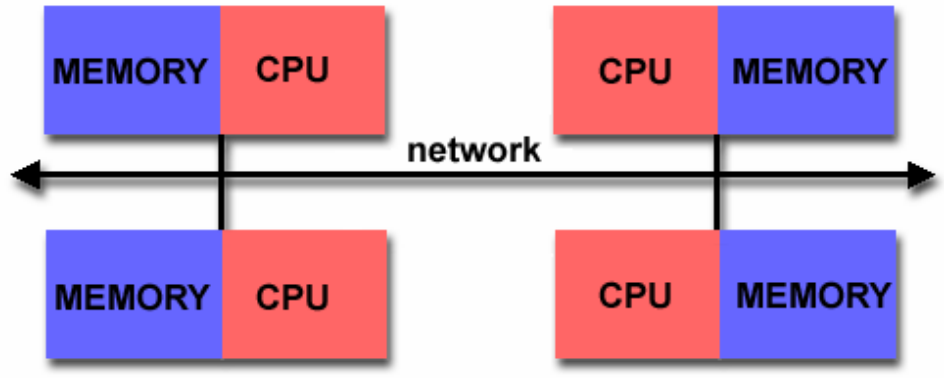


Serial vs Multicore Approaches

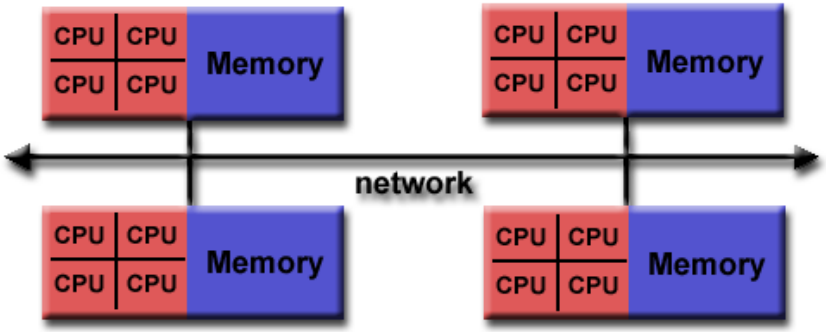
Many different parallelization approaches, which we won't discuss:

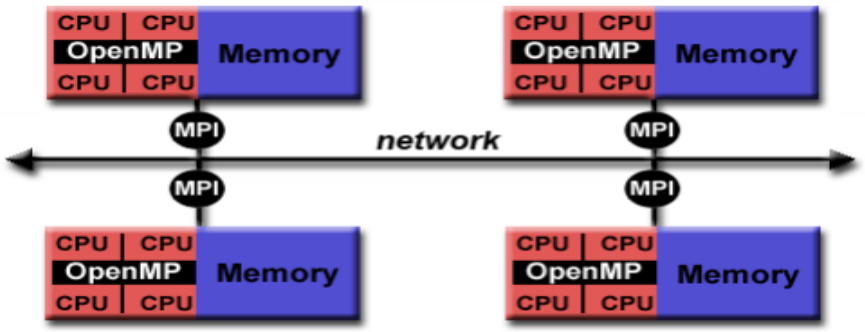


Shared memory



Distributed memory





Hybrid Distributed-Shared memory

Multicore Options in R, Python, & Perl

In order to run in parallel, programs (code) must be explicitly programmed to do so. Thus, requesting cores from the scheduler does not automagically parallelize your code.

```
#!/bin/bash
#
#SBATCH -p serial_requeue          # Partition to submit to (comma separated)
#SBATCH -J frog_blast              # Job name
#SBATCH -n 8                       # Number of cores
#SBATCH -N 1                       # Ensure that all cores are on one machine
...
module load ncbi-blast/2.2.31+-fasrc01
blastn -query seqs.fasta -db nt -out seqs.nt.blastn                                  # WRONG!!
blastn -query seqs.fasta -db nt -out seqs.nt.blastn -num_threads $SLURM_NTASKS # YES!!
```

By default, R, Python, and Perl are not multithreaded … so do not ask for >1 core.

- For R, you can use appropriate routines with Rparallel, Rforeach, RdoMC, or Rsnow
- For Python, you can use the multiprocessing library (or many others)
- For Perl, there's threads or Parallel::ForkManager

```
# R example
library(doMC)
mclapply(seq_len(), run2, mc.cores = Sys.getenv('SLURM_NTASKS'))
```

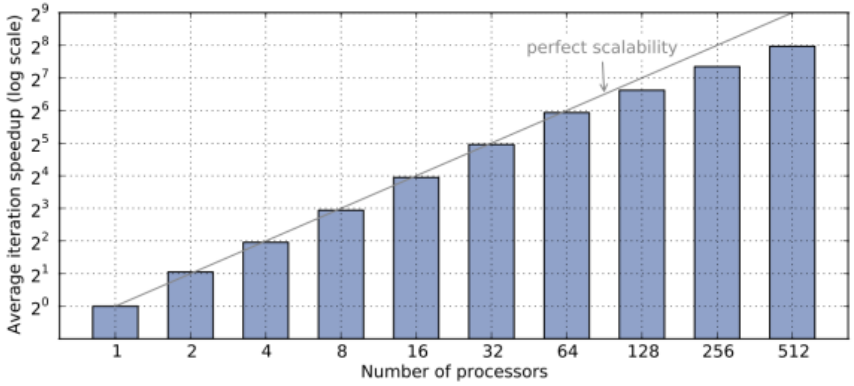
Scaling Tests Ensures Efficiency

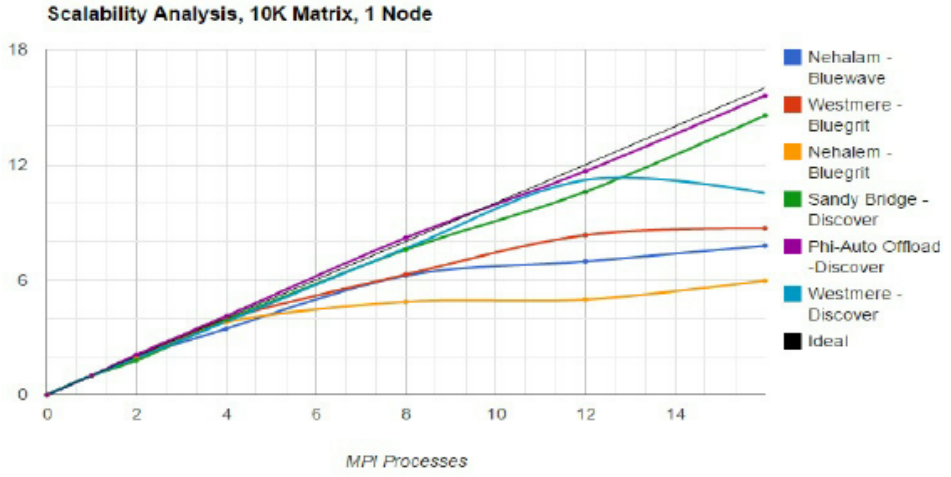
HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Not all programs can be scaled well. This is due to

- Overhead of program start
- Overhead of communication between processes (threads) within the program
- (worse:) Waiting to write to the network or disk (I/O)
- Other, serial parts of the program (parts that cannot be parallelized)

Scaling tests are important to help you determine the optimal # of cores to use!!





Your Own Scaling Tests!

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Create a SLURM script for an analysis that can be used for multiple CPU (core) values
# Input seqs.fa file has 350 FASTA sequences so we can get good parallelization values:

-- file: blast_scale_test.slurm ---
     #!/bin/bash
     #
     #SBATCH -p serial_requeue        # Partition to submit to (comma separated)
     #SBATCH -J blastx                # Job name
     #SBATCH -N 1                     # Ensure that all cores are on one machine
     #SBATCH -t 0-4:00                # Runtime in D-HH:MM (or use minutes)
     #SBATCH --mem 10000              # Memory pool in MB for all cores
     #SBATCH --mail-type=END,FAIL     # Type of email notification: BEGIN,END,FAIL,ALL

     source new-modules.sh; module load ncbi-blast/2.2.31+-fasrc01
     export BLASTDB=/n/regal/informatics_public/

     blastx -in seqs.fa -db $BLASTDB/custom/other/model_chordate_proteins \
            -out sk_shuffle_seqs.n${1}.modelchordate.blastx -num_threads $1
-----

# and now submit file multiple times with different core values
for i in 1 2 4 8 16; do
  echo $i
  # sbatch flags here will override those in the SLURM submission script
  sbatch -n $i -J blastx$i -o blastx_n$i.out -e blastx_n$i.err blast_scale_test.slurm $i
  sleep 1
done
```


Your Own Scaling Test Results!

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

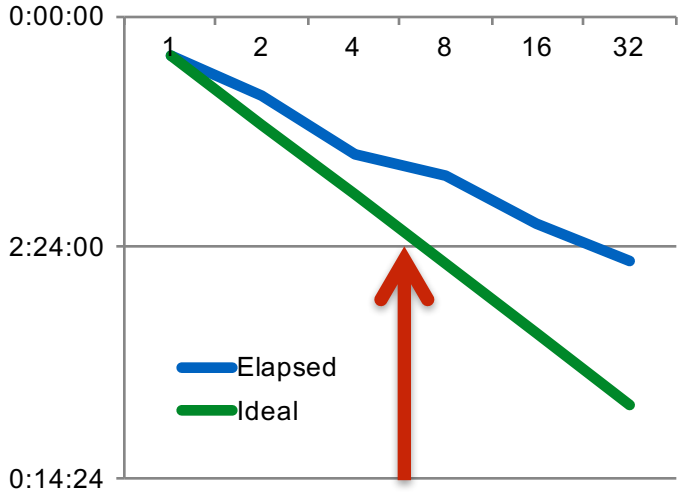
```
[bfreeman@rclogin04  ~]$ sacct -u bfreeman -S 4/6/16 --format=jobid,\
elapsed,alloccpus,cputime,totalcpu,state
       JobID    Elapsed  AllocCPUS     CPUTime   TotalCPU      State
------------ ---------- ---------- ---------- ---------- ----------
59817008       16:12:26          1   16:12:26   16:03:34  COMPLETED
59817008.ba+   16:12:26          1   16:12:26   16:03:34  COMPLETED
59817024       10:49:16          2   21:38:32   17:53:07  COMPLETED
59817024.ba+   10:49:16          2   21:38:32   17:53:07  COMPLETED
59817026       06:03:38          4 1-00:14:32   15:56:55  COMPLETED
59817026.ba+   06:03:38          4 1-00:14:32   15:56:55  COMPLETED
59817028       04:55:44          8 1-15:25:52   21:27:30  COMPLETED
59817028.ba+   04:55:44          8 1-15:25:52   21:27:30  COMPLETED
59817043       03:01:51         16 2-00:29:36 1-01:33:03  COMPLETED
59817043.ba+   03:01:51         16 2-00:29:36 1-01:33:03  COMPLETED
59847485       02:04:58         32 2-18:38:56 1-11:42:36  COMPLETED
59847485.ba+   02:04:58         32 2-18:38:56 1-11:42:36  COMPLETED
```

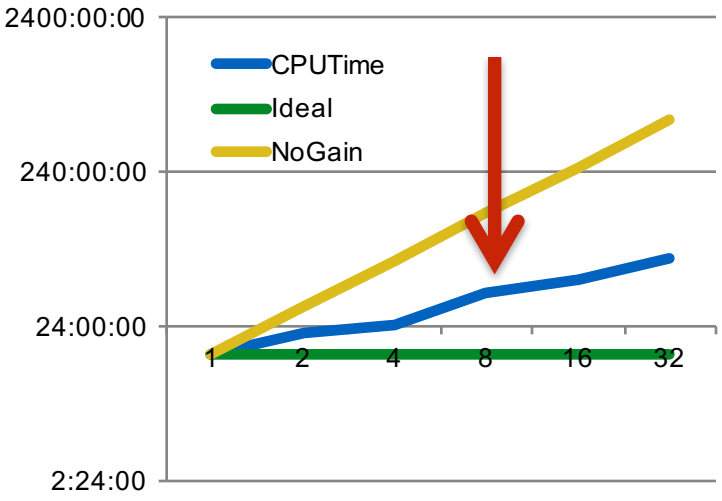
	1	*2*	*4*	*8*	*16*	*32*
Elapsed	**16:12:26**	**10:49:16**	**6:03:38**	**4:55:44**	**3:01:51**	**2:04:58**
Ideal	16:12:26	8:06:13	4:03:07	2:01:33	1:00:47	0:30:23
CPUTime	**16:12:26**	**21:38:32**	**24:14:32**	**39:25:52**	**48:29:36**	**66:38:56**
Ideal	16:12:26	16:12:26	16:12:26	16:12:26	16:12:26	16:12:26
NoGain	16:12:26	32:24:52	64:49:44	129:39:28	259:18:56	518:37:52
TotalCPU	**16:03:34**	**17:53:07**	**15:56:55**	**21:27:30**	**25:33:03**	**35:42:36**
Ideal	16:03:34	16:03:34	16:03:34	16:03:34	16:03:34	16:03:34
NoGain	16:03:34	32:07:08	64:14:16	128:28:32	256:57:04	513:54:08

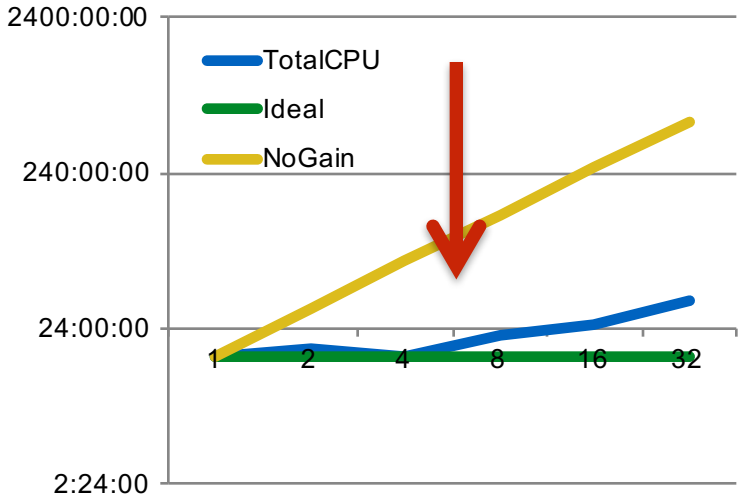
Your Own Scaling Test Results!

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

	1	2	4	8	16	32
Elapsed	**16:12:26**	**10:49:16**	**6:03:38**	**4:55:44**	**3:01:51**	**2:04:58**
Ideal	16:12:26	8:06:13	4:03:07	2:01:33	1:00:47	0:30:23
CPUTime	**16:12:26**	**21:38:32**	**24:14:32**	**39:25:52**	**48:29:36**	**66:38:56**
Ideal	16:12:26	16:12:26	16:12:26	16:12:26	16:12:26	16:12:26
NoGain	16:12:26	32:24:52	64:49:44	129:39:28	259:18:56	518:37:52
TotalCPU	**16:03:34**	**17:53:07**	**15:56:55**	**21:27:30**	**25:33:03**	**35:42:36**
Ideal	16:03:34	16:03:34	16:03:34	16:03:34	16:03:34	16:03:34
NoGain	16:03:34	32:07:08	64:14:16	128:28:32	256:57:04	513:54:08







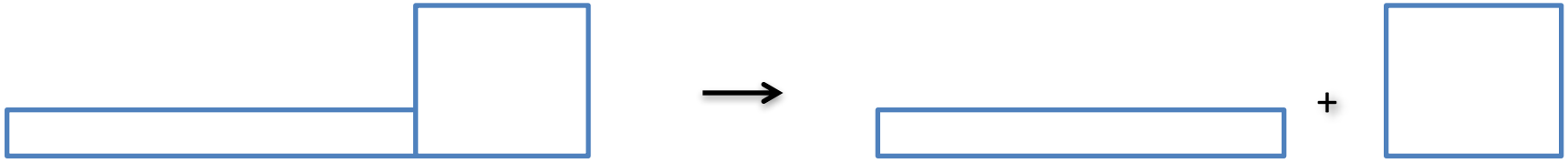
Mixed Multicore and Serial Workflows

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Choosing core count can be difficult, especially if there's a mix of serial and parallel steps….

- Think about how long your code will be in either modes
- Determine the fraction resource use across the whole job
- If < 20% in multicore use, then split up the tasks into two separate jobs
- Can use job dependencies to make submission easier

OK, can run as one long job



Overview

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

1. Cluster basics
   - Access, storage, software, partitions, resources
2. Using software
   - Module system, Java & Python, Updating local modules/packages
3. Running multicore (multiCPU) jobs & Scaling considerations
4. Pleasantly parallelizing tasks (e.g. getting BLAST results fast!)
5. Checkpointing to save (re)compute time
6. SLURM scripts for common programs & typical workflows
7. Troubleshooting & Getting help

Concept of Pleasant Parallelization

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Problem: How do I BLAST 200,000 transcripts against NR?

Solution: *Fake* a parallel BLAST. But how?

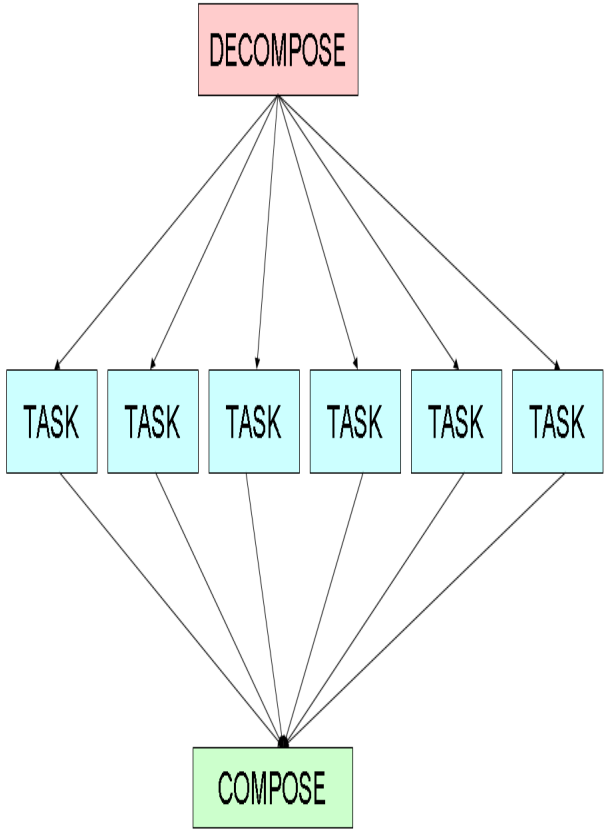
- Divide your input file into *n* separate files
- BLAST each smaller input file on a separate core
- Running on *n* cores will be almost exactly as *n* times faster!

Why?

- Each core doesn't need to talk to one another
- You could submit *n* jobs individually, but not recommended
- Use more sophisticated techniques:
  job arrays, gnu_parallel, GridRunner
- Shouldn't confuse this with *truly parallel* mpiBLAST

The efficiency of your work depends on how *parallelized* you make your task:

- You want to ensure that your jobs spend most of their time computing, and not in the queue or doing compute prep



X 100?? What would you choose?

Manual (Script) Approach

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Split input file into N files that run 1 to 6 hrs each
# can be done with perl or python script, scriptome, or fasta_tool
# create SLURM script for job array (nano blast_array.slurm)

-- file: blast_array.slurm ---
     #!/bin/bash
     #SBATCH --open-mode=append       # ensure output files are not overwritten
     #SBATCH -p serial_requeue        # Partition to submit to (comma separated)
     #SBATCH -J blastn_array          # Job name
     #SBATCH -n 1                     # Number of cores
     #SBATCH -N 1                     # All cores on one machine
     #SBATCH -t 0-1:00                # Runtime in D-HH:MM (or use minutes)
     #SBATCH --mem 2000               # Memory in MB
     #SBATCH -e blastn_%A_%a.err      # STDERR. %A is jobID, %a is 1 2 3 4 etc
     #SBATCH --mail-type=FAIL         # Type of email notification: BEGIN,END,FAIL,ALL
     #SBATCH --mail-user=rmf@123.com  # Email to which notifications will be sent
     #
     #SBATCH -o seqs_%a.nt.blastx     # STDOUT. %a is 1 2 3 4 etc

     source new-modules.sh; module load ncbi-blast/2.2.31+-fasrc01
     export BLASTDB=/n/regal/informatics_public/ref/

     # sleep random # seconds do that all jobs don't hit nt at once
     sleep $[ ( $RANDOM % $SLURM_ARRAY_TASK_MAX )  + 1 ]s
     blastn -query seqs_$SLURM_ARRAY_TASK_ID.fasta -db $BLASTDB/ncbi/nt/nt
-----

# and now submit file as job array
sbatch --array=1-N blast_array.slurm
```


GnuParallel Approach

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Split input file into N files that run 1 to 6 hrs each
# create list of commands to be executed in a text file
# create SLURM script for parallel execution on one machine (NB!)

-- file: blast_gnu_parallel.slurm ---
     #!/bin/bash
     #SBATCH --open-mode=append       # ensure output files are not overwritten
     #SBATCH -p serial_requeue        # Partition to submit to (comma separated)
     #SBATCH -J blastn_gnu_parallel   # Job name
     #SBATCH -n 16                    # Number of cores
     #SBATCH -N 1                     # All cores on one machine
     #SBATCH -t 0-6:00                # Runtime in D-HH:MM (or use minutes)
     #SBATCH --mem 64000              # Memory pool for all cores
     #SBATCH -o blast_gp_%j.out       # STDOUT. %j is jobID
     #SBATCH -e blast_gp_%j.err       # STDERR. %j is jobID
     #SBATCH --mail-type=FAIL         # Type of email notification: BEGIN,END,FAIL,ALL

     module load parallel/20160322-fasrc01
     log=`basename $1`

     parallel -joblog $log.log --outputasfiles -j$SLURM_NTASKS :::: $1
-----

# and now submit file to run on one machine
sbatch blast_gnu_parallel.slurm my_job_list.txt
```


GridRunner Approach

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Trinity-based grid scheduler to run FASTA file thru analyses by pleasant parallelization
# Install GridRunner in $HOME/apps and create grid.conf file that has configuration info

-- file: blast_gridrunner.slurm ---
     #!/bin/bash
     #SBATCH -p general                # Partition to submit to (comma separated)
     #SBATCH -J blastn_gridrunner      # Job name
     #SBATCH -n 1                      # Number of cores
     #SBATCH -N 1                      # All cores on one machine
     #SBATCH -t 0-6:00                 # Runtime in D-HH:MM (or use minutes)
     #SBATCH --mem 1000                # Memory in MB
     #SBATCH -o blast_gr_%j.out        # STDOUT. %j is jobID
     #SBATCH -e blast_gr_%j.err        # STDERR. %j is jobID
     #SBATCH --mail-type=FAIL          # Type of email notification: BEGIN,END,FAIL,ALL

     module load ncbi-blast/2.2.31+-fasrc01
     export BLASTDB=/n/regal/informatics_public/ref

     # this splits input FASTA file and starts watcher/dispatcher script
     $HOME/apps/BioIfx/hpc_FASTA_GridRunner.pl \
       --cmd_template "blastp -query __QUERY_FILE__ -db $BLASTDB/ncbi/swissprot/swissprot\
                       -max_target_seqs 1 -outfmt 6 -evalue 1e-5" \
     --query_fasta test.pep \
     -G odyssey.grid.conf \
     -N 10 -O test_blastp_search
-----
# and now submit file to run on one machine
sbatch blast_gridrunner.slurm
```


http://hpcgridrunner.github.io/

GridRunner Approach

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Trinity-based grid scheduler to run FASTA file thru analyses by pleasant parallelization
# Install GridRunner in $HOME/apps and create grid.conf file that has configuration info

-- file: odyssey_grid.conf ---
#-------------------------------------------------------------------------------
# grid type:
grid=SLURM

# template for a grid submission: YOU WILL NEED TO TEST AND SET THESE OPTIONS
#     TIME WILL DEPEND ON THE cmds_per_node
cmd=sbatch -p general --mem=4000 --time=02:00:00

# number of grid submissions maintained at steady state by the Trinity submission system
max_nodes=1000

# number of commands that are batched into a single grid submission job.
# YOU WILL NEED TO TEST AND SET THESE OPTIONS
cmds_per_node=60

#-------------------------------------------------------------------------------
```


http://hpcgridrunner.github.io/

Overview

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

1. Cluster basics
   - Access, storage, software, partitions, resources
2. Using software
   - Module system, Java & Python, Updating local modules/packages
3. Running multicore (multiCPU) jobs & Scaling considerations
4. Pleasantly parallelizing tasks (e.g. getting BLAST results fast!)
5. Checkpointing to save (re)compute time
6. SLURM scripts for common programs & typical workflows
7. Troubleshooting & Getting help

Checkpointing

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

- Saves the process of running application to a file, to be restarted later if necessary
- Safeguard long-running jobs, esp. on problematic systems or across cluster outages
- Functionality must be compiled into your source code

On SLURM, Use the Berkeley Lab Checkpoint/Restart (BLCR)

- Plug-in must be installed and enabled on cluster
- Features:
  1. Checkpoint of whole batch jobs in addition to job steps
  2. Periodic checkpoint of batch jobs and job steps
  3. Restart execution of batch jobs and job steps from checkpoint files
  4. Automatically requeue and restart the execution of batch jobs upon node failure

General mode of operation is to

1. Start the job step using the `srun_cr` command.
2. Create a checkpoint of `srun_cr` using BLCR's `cr_checkpoint` command and cancel the job. `srun_cr` will automatically checkpoint your job.
3. Restart `srun_cr` using BLCR's `cr_restart` command. Job will be restarted using a newly allocated jobid.

Checkpoint/blcr can create checkpoints for both interactive and batch steps, but only batch jobs can be restarted.

BLCR operation has been verified with MVAPICH2. Some other MPI implementations should also work.

See the SLURM docs at http://www.schedmd.com for more details

Checkpointing

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

For those not writing C or Fortran, there are other & 'poor man' approaches…

- Some applications already have checkpointing built in:
  1. Guassian
  2. Abaqus
  3. Molpro
- DMTCP tool can checkpoint state of programs, including multi-threaded & distributed apps

- Save the program state in a 'state' file on a periodic basis
  - Use serialization tools to periodically save all program variables on periodic basis
  - Can use same tools to load state back in after restart
- Trap Unix signals (e.g. SIGKILL, SIGSTOP) to get notification from the OS
  - Can be written into scripting languages R, Perl, Python, bash, etc.
- Ask SLURM to send signals to your running code before killed

- Can use 'file completion breadcrumbs' to restart work after parts have been completed

For more information, see http://www.cism.ucl.ac.be/Services/Formations/checkpointing.pdf

Checkpointing: 'poor man' Approach

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
#!/bin/bash
... #SBATCH stuff goes here ...
... module stuff goes here ...

# do some work, ensuring that we are 'checkpointing' along the way
SCRATCH=/n/regal/freeman_lab/bfreeman
startdir=$PWD
cd $SCRATCH/run1/

if [ ! -e part1.finished ]; then
    # if our part1.finished doesn't exist, then do this work…
    cp $HOME/run1/myfile.fastq.gz .
    gunzip myfile.fastq.gz

    touch part1.finished
fi

if [ ! -e part2.finished ]; then
    # if our part2.finished doesn't exist, then do this work…
    fastqc --outdir fastqc_data -threads 1 myfile.fastq

    touch part2.finished
fi

...

# let's go back from whence we started...
cd $startdir
```


Overview

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

1. Cluster basics
   - Access, storage, software, partitions, resources
2. Using software
   - Module system, Java & Python, Updating local modules/packages
3. Running multicore (multiCPU) jobs & Scaling considerations
4. Pleasantly parallelizing tasks (e.g. getting BLAST results fast!)
5. Checkpointing to save (re)compute time
6. SLURM scripts for common programs & typical workflows
7. Troubleshooting & Getting help

Example SLURM Scripts

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

- Demonstrate the various ways of using SLURM for bioinformatics workflows
- Can be used as templates for the various types of work that you wish to do

Program	***Techniques***
Multicore BLAST	Basic multicore application; using bash variables for file paths
Bowtie + Samtools	Piping commands with appropriate core selection; appication with dual single- and multi-core options
FASTQC	Purging module environment; truncating filenames; output redirection
Trimmomatic	Calling a java application with options; setting run path
Trinity	Job dependency; 2nd part is watcher (metascheduler) script with highly parallelized child jobs
RaxML	Running an MPI application; complex program with hybrid OpenMP/MPI modes
OMA	1st part is imple (built-in) single-core, low-memory job array functionality ; 2nd part is single-core, high-memory job-dependency submission

- Will be available online at our Github site

See our Github repository: https://github.com/fasrc/slurm_utils

Multicore BLAST

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Create SLURM script for multicore job (nano blast_multicore.slurm)

-- file: blast_multicore.slurm ---
     #!/bin/bash
     #SBATCH --open-mode=append       # ensure output files are not overwritten
     #SBATCH -p serial_requeue        # Partition to submit to (comma separated)
     #SBATCH -J blastn                # Job name
     #SBATCH -n 4                     # Number of cores
     #SBATCH -N 1                     # All cores on one machine
     #SBATCH -t 0-12:00               # Runtime in D-HH:MM (or use minutes)
     #SBATCH --mem 12000              # Memory in MB
     #SBATCH -o blastn_&j.out         # STDEOUT
     #SBATCH -e blastn_&j.err         # STDERR
     #SBATCH --mail-type=END,FAIL     # Type of email notification: BEGIN,END,FAIL,ALL

     source new-modules.sh; module load ncbi-blast/2.2.31+-fasrc01
     export BLASTDB=/n/regal/informatics_public/ref/

     blastn -query seqs.fasta \
       -db $BLASTDB/ncbi/nt/nt \
       -num_threads $SLURM_NTASKS \
       -out seqs.nt.blastn

-----

# and now submit job
sbatch blast_multicore.slurm
```


Bowtie + Samtools

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Map a FASTQ file against your genome and then sort it appropriately

-- file: map_n_sort.slurm ---
     #!/bin/bash
     #SBATCH -p serial_requeue       # Partition to submit to
     #SBATCH -n 8                    # Number of cores
     #SBATCH -N 1                    # Ensure that all cores are on one machine
     #SBATCH -t 0-6:00               # Runtime in days-hours:minutes
     #SBATCH --mem 8000              # Memory in MB
     #SBATCH -J mapNsort             # job name
     #SBATCH -o mapNsort_%j.out      # File to which standard out will be written
     #SBATCH -e mapNsort_%j.err      # File to which standard err will be written
     #SBATCH --mail-type=ALL         # Type of email notification- BEGIN,END,FAIL,ALL

     module load bowtie2/2.2.4-fasrc01 samtools/1.2-fasrc01

     bowtie2 -x genome -p $SLURM_NTASKS -1 seq.R1.fastq -2 seq.R2.fastq \
        | samtools view -b output.bam
     samtools sort -@ $SLURM_NTASKS -O bam -o output.sorted.bam
-----

# and now submit job
sbatch map_n_sort.slurm
```


FASTQC

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Generic batch file that will allow you to process lots of FASTQ files

-- file: fastqc.slurm ---
     #!/bin/bash
     #SBATCH -p serial_requeue       # Partition to submit to
     #SBATCH -n 8                    # Number of cores
     #SBATCH -N 1                    # Ensure that all cores are on one machine
     #SBATCH -t 0-3:00               # Runtime in days-hours:minutes
     #SBATCH --mem 2000              # Memory in MB
     #SBATCH -J FastQC               # job name
     #SBATCH -o FastQC.%j.out        # File to which standard out will be written
     #SBATCH -e FastQC.%j.err        # File to which standard err will be written
     #SBATCH --mail-type=ALL         # Type of email notification- BEGIN,END,FAIL,ALL

     module purge ## Why? Clear out .bashrc /.bash_profile settings that might interfere
     module load fastqc/0.11.5-fasrc01

     # grab filename base and create output directory
     j=`basename $1`
     mkdir -p fastqc_$j

     fastqc --outdir fastqc_$j -threads $SLURM_NTASKS $1  2>&1 > $j.fastqc.sbatch.out
-----

# and now submit job (can also loop to submit files - remember to sleep 1 between submits)
sbatch fastqc.slurm my_input_file.fastq
```


Trimmomatic

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Generic batch file that will allow you to process lots of FASTQ files

-- file: trimmomatic.slurm ---
     #!/bin/bash
     #SBATCH -p serial_requeue           # Partition to submit to
     #SBATCH -n 4                        # Number of cores
     #SBATCH -N 1                        # Ensure that all cores are on one machine
     #SBATCH -t 0-6:00                   # Runtime in days-hours:minutes
     #SBATCH --mem 2000                  # Memory in MB
     #SBATCH -o PSG177_trim.out          # File to which standard out will be written
     #SBATCH -e PSG177_trim.err          # File to which standard err will be written
     #SBATCH --mail-type=ALL             # Type of email notification- BEGIN,END,FAIL,ALL

     module load java/1.8.0_45-fasrc01
     export TRIMMOMATIC=$HOME/apps/trimmomatic
     mkdir trimmed

     java -jar $TRIMMOMATIC/trimmomatic-0.32.jar PE \
       -threads $SLURM_NTASKS \
       PSG177_TGACCA.R1.fastq.gz PSG177_TGACCA.R2.fastq.gz \
       trimmed/PSG177_TGACCA.R1.pair.fastq trimmed/PSG177_TGACCA.R1.single.fastq \
       trimmed/PSG177_TGACCA.R2.pair.fastq trimmed/PSG177_TGACCA.R2.single.fastq \
       ILLUMINACLIP:illuminaClipping_main.fa:2:40:15 \
       LEADING:3 TRAILING:3 \
       SLIDINGWINDOW:4:20 MINLEN:25
-----

# and now submit job
sbatch trimmomatic.slurm
```


Trinity (Gridrunner)

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Trinity: Inchworm + Chrysalis

create file trinity_ic.slurm in your favorite **text** editor
you may need to run this on the bigmem partition

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

#SBATCH  -p general                        # Partition  to submit  to
#SBATCH  –n 16                             # Number  of cores
#SBATCH  -N 1                              # Ensure  that  all  cores  are  on  one  machine
#SBATCH  –t 3-0:00                         # Runtime  in days-hours:minutes
#SBATCH  --mem  155000                     # Memory  in MB
#SBATCH  –J trinity_ic                     # job  name
#SBATCH  -o A1_trinity_ic.out              # File  to  which  standard  out will  be written
#SBATCH  –e A1_trinity_ic.err              # File  to  which  standard  err will  be written
#SBATCH  --mail-type=ALL                   # Type  of  email  notification-  BEGIN,END,FAIL,ALL
#SBATCH  --mail-user=name@harvard.edu      # Email  to  which  notifications  will  be sent

source  new-modules.sh;  module  load  trinityrnaseq

# cat R1 singles onto L pair; and R2 singles onto R pair
gunzip trimmed/A1_R1.single.fastq.gz; gunzip trimmed/A1_R2.single.fastq.gz;
cat trinity/R1_normalized.fq trimmed/A1_R1.single.fastq > trinity/A1_R1.p+s.clean.norm.fastq
cat trinity/R2_normalized.fq trimmed/A1_R2.single.fastq > trinity/A1_R2.p+s.clean.norm.fastq

Trinity --seqType fq \
   --JM 150G \
   --left trinity/A1_R1.p+s.clean.norm.fastq --right trinity/A1_R2.p+s.clean.norm.fastq \
   --SS_lib_type FR \
   --output trinity_output \
   --CPU 16 \
   --min_kmer_cov 2 \
   --max_reads_per_loop 5000000 \
   --group_pairs_distance 800 \
   --no_butterfly
-------

sbatch trininty_ic.slurm

Submitted batch job 22855027
```

Trinity (Gridrunner)

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Trinity: Butterfly

```
create file trinity_SLURM_conf.txt in your favorite text editor
-----
#-------------------------------------------------------------------------------------------
# grid type:
grid=SLURM
# template for a grid submission
cmd=sbatch -p serial_requeue --mem=10000 --time=02:00:00
# number of grid submissions to be maintained at steady state by the Trinity submission system
max_nodes=1000
# number of commands that are batched into a single grid submission job.
cmds_per_node=60
#-------------------------------------------------------------------------------------------
-----

create file trinity_b.slurm in your favorite text editor
-----
#!/bin/bash

#SBATCH -p general                          # Partition to submit to
#SBATCH –n 1                                # Number of cores
#SBATCH -N 1                                # Ensure that all cores are on one machine
#SBATCH –t 1-0:00                           # Runtime in days-hours:minutes
#SBATCH --mem 4000                          # Memory in MB
#SBATCH –J trinity_b                        # job name
#SBATCH -o A1_trinity_b.out                 # File to which standard out will be written
#SBATCH –e A1_trinity_b.err                 # File to which standard err will be written
#SBATCH --mail-type=ALL                     # Type of email notification- BEGIN,END,FAIL,ALL
#SBATCH --mail-user=name@harvard.edu        # Email to which notifications will be sent

source new-modules.sh; module load trinityrnaseq

Trinity --seqType fq \
  --left trinity/A1_R1.p+s.clean.norm.fastq --right trinity/A1_R2.p+s.clean.norm.fastq \
  --SS_lib_type FR \
  --output trinity_output \
  --grid_conf trinity_SLURM_conf.txt

-------

sbatch --dependency=afterok:22855027 trininty_b.slurm
```


RaxML (Hybrid OpenMP/MPI)

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# RaxML is a complex program that will scale depending on input # taxa & patterns
# Usage depends on how this has been compiled. Need to look in bin/ directory of software
# SORRY, NO EASY SOLUTIONS, but many good clues!!
#   PLEASE READ http://sco.h-its.org/exelixis/pubs/Exelixis-RRDR-2010-3.pdf AND
#      http://sco.h-its.org/exelixis/resource/doc/Phylo100225.pdf
# Can run on one node or across multiple nodes; depends on how you want to scale
# See good docs at http://wiki.hpc.ufl.edu/doc/RAxML
# To get help: mpirun -np 1 raxml -h

-- file: raxml-hybrid.slurm ---
     #!/bin/bash
     #SBATCH -p general              # Partition to submit to
     #SBATCH -n 16                   # Number of cores
     #SBATCH -N 1                    # 1 if under 32 cores; >= 2 machines otherwise
     #SBATCH -t 0-6:00               # Runtime in days-hours:minutes
     #SBATCH --mem-per-cpu 4000      # Memory in MB; see raxML docs (good info there!)
     #SBATCH -J raxml_test           # job name
     #SBATCH -o raxml_test_%j.out    # File to which standard out will be written
     #SBATCH -e raxml_test_%j.err    # File to which standard err will be written
     #SBATCH --mail-type=END,FAIL    # Type of email notification- BEGIN,END,FAIL,ALL

     module load gcc/4.8.2-fasrc01 openmpi/1.10.0-fasrc01 raxml/8.1.5-fasrc02
     ls -al $RAXML_HOME/bin

     # do not use -T option unless for raxml-PTHREADS or raxml-HYBRID
     mpirun -np $SLURM_NTASKS raxmlHPC-MPI-SSE3 [options]
-----

# and now submit job
sbatch raxml-hybrid.slurm
```


OMA

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

```
# Generic 1st OMA file to run as job array

-- file: oma1.slurm ---
     #!/bin/bash
     #SBATCH -p serial_requeue          # Partition to submit to
     #SBATCH -n 1                       # Number of cores
     #SBATCH -t 0-6:00                  # Runtime in days-hours:minutes
     #SBATCH --mem 2000                 # Memory in MB
     #SBATCH -o oma1_%A_%a.out          # STDOUT
     #SBATCH -e oma1_%A_%a.out          # STDERR
     #SBATCH --mail-type=FAIL           # Type of email notification- BEGIN,END,FAIL,ALL

     module load OMA/1.0.3-fasrc01
     OMA -s                             # works on data in current directory
-----

sbatch -array=1-100 oma1.slurm          # submit 1st all x all as job array

Submitted jobid 59634571

-- file: oma2.slurm ---
     #!/bin/bash
     #SBATCH -p general                 # Partition to submit to
     #SBATCH -n 1                       # Number of cores
     #SBATCH -t 0-6:00                  # Runtime in days-hours:minutes
     #SBATCH --mem 20000                # Memory in MB
     #SBATCH -o oma2_%j.out             # STDOUT
     #SBATCH -e oma2_%j.out             # STDERR
     #SBATCH --mail-type=FAIL           # Type of email notification- BEGIN,END,FAIL,ALL

     module load OMA/1.0.3-fasrc01
     OMA                                # works on data in current directory
-----

sbatch --dependency=afterok:59634751 oma2.slurm   # process AllxAll data; job dependency
```


Overview

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

1. Cluster basics
   - Access, storage, software, partitions, resources
2. Using software
   - Module system, Java & Python, Updating local modules/packages
3. Running multicore (multiCPU) jobs & Scaling considerations
4. Pleasantly parallelizing tasks (e.g. getting BLAST results fast!)
5. Checkpointing to save (re)compute time
6. SLURM scripts for common programs & typical workflows
7. Troubleshooting & Getting help

Basic Troubleshooting

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

Before seeking help, take some basic steps to ascertain what is going on with your job:

- Use `squeue` and `sacct` with `--format=` option to query details from SLURM
  - Are you having Fairshare issues (`Priority`)?
  - Is your job waiting for space (`Resources`)?
  - Will your job ever run (`Dependency`)?
  - Is there an error code or message
- Check your log files
  - You did specify both `-o` and `-e`, yes?
  - No log files? Does the path to your log files exist before the job start?
  - Message about `Pre-emption`, `Timeout`, or `Failure`?
  - The last error in the log is usually not the problem. The first one is!
- Did you request e-mail messages for your jobs with `--mail-type=`?
- Is your SLURM script formatted properly?
- Are you loading legacy modules? Possible software/library conflicts?

Check out Tips@12 presentation http://fasrc.us/fasrcmaterials

Problems, Pitfalls, and Prevention

HARVARD Faculty of Arts and Sciences RESEARCH COMPUTING

This is a shared resource, so everyone has skin in the game. And you can help us and yourself…

- Node and cluster problems are not unusual, esp. as large as system as Odyssey: I/O errors, node failures, memory errors, etc. Let us know if you see these.
- Review our Usage & Responsibilities guidelines: http://fasrc.us/hpccustoms
- Review our Common Pitfalls, lest you fall victim: http://fasrc.us/hpcpitfalls

Don't use multiple cores for R and Python scripts
These interpreters/runtime environments are can one use 1 core. Don't waste please.

PEND for >48 hrs
Asking for very large resource requests (cores/memory);very low Fairshare score

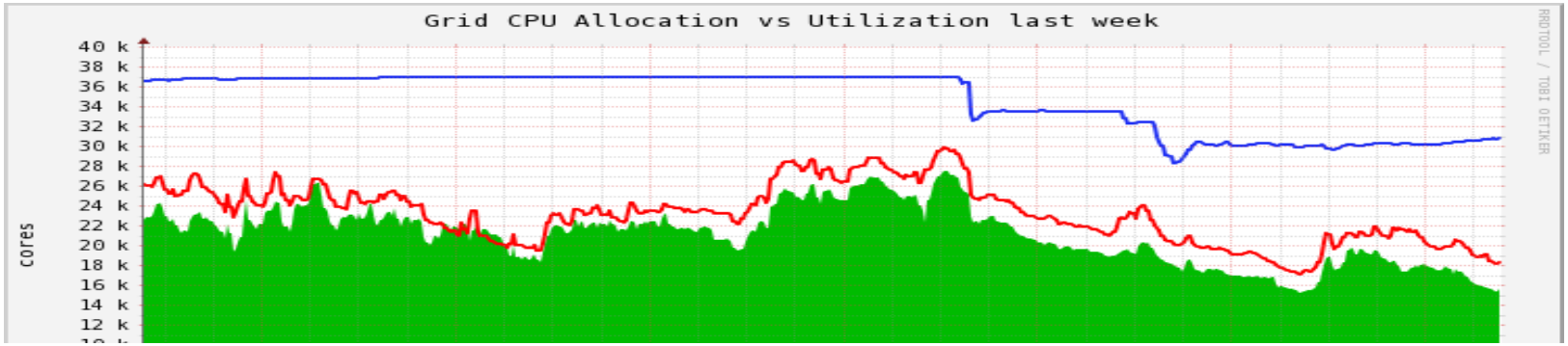
Quick run and FAIL…Not including -t parameter
no -t means shortest possible in all partitions == 10 min

Asking for multiple cores but forgetting to specify one node
-n 4 -N 1 is very different from -n 4

Not specifying enough cores
prog1 | prog2 | prog3 > outfile should run with 3 cores

Causing massive disk I/O on home folders/lab disk shares
your work & others on the same filesystem slows to a crawl; simple commands like ls take forever

Job efficiency



RC regularly reviews jobs based on their effective usage of their SLURM reservations (cores, memory, time, disk, ...) to promote maximum utilization of these resources.

- Over-requesting resources negatively effects the scheduling priority of your own jobs and blocks other users from these resources, which further lowers the overall research output for all HU users.
- Under-requesting resources negatively effects your job and those running on the same nodes; and potentially other jobs on the same filesystem

You may be contacted if you are regularly are having issues with your job efficiency and we will work with you to improve your performance.

Can calculate the efficiencies with the following formula:

```
sacct -u RCUSERNAME --format=user,state,jobid,alloccpus,elapsed,cputime
EffCPUs = CPUTime / Elapsed
%Eff = CPUTime / (AllocCPUs * Elapsed)
```

Getting Help

RC Website & Documentation -- only authoritative source

https://rc.fas.harvard.edu/

Submit a ticket on the portal https://portal.rc.fas.harvard.edu/

Best way to help us to help you? Give us…

Description of problem
Additional info (login/batch? partition? JobIDs?)
Steps to Reproduce (1., 2., 3...)
Actual results
Expected results

OdyBot, for quick-fix problems http://odybot.org/

Research Computing

HARVARD
Faculty of Arts and Sciences
RESEARCH COMPUTING

Please talk to your peers, and …
We wish you success in your research!

http://rc.fas.harvard.edu
https://portal.rc.fas.harvard.edu
@fasrc

Harvard Informatics
@harvardifx