Compute Cluster Workshop

Mike Smith & **Toby Hodges**

Introduction

https://git.embl.de/grp-bio-it/embl_hpc

What is a compute cluster?

- A bunch of individual machines (**nodes**) tied together
    - Nodes are often heterogeneous
        - No. of CPU cores, Memory, Disk space, ...
- Special software is used to represent those machines as a pool of shared resources
- This software gives you ability to ask for a chunk of this pool to run your software

What is a compute cluster?

- Tailored to batch processing (=jobs)
- Interactive use possible
- You don't care on which machine your job is running
- If you do, you can ask for specific resources to be allocated to you

High **P**erformance **C**omputing (HPC): the (effective) use of multiple computers to do things you couldn't do on a single machine.

When is HPC useful?

- When you want to get results faster than what your laptop can offer
    - Compute Intensive: Task requiring a large amount of computation
        - e.g. more rigorous sequence alignment
    - Memory Intensive: Task requiring a large amount of memory
        - e.g. scaling up from bacterial to human genome
    - Data Intensive: Task involved operating on a large amount of data
        - e.g. 50 human genomes

Where to find help

- Training like this one and the one tomorrow
- Wiki: https://wiki.embl.de/cluster/
- chat.embl.org #cluster
- itsupport@embl.de
- clusterNG mailing list
- Meetings as needed
  - When there are new things to announce and explain
- Bio-IT drop-in sessions and meetings, Coding Club

Jobs & Scheduling

How do I work with a cluster?

- Typically interact with a frontend (head) node
- A **job scheduler** manages where and when tasks are run
    - There are many options available e.g. LSF, Torque, Slurm, Condor, Univa Grid Engine
- Matches job requirements with available resources
- If no slots are available a job will wait until resources are available

Slurm

- “**S**imple **L**inux **U**tility for **R**esource **M**anagement”
- One of the most popular HPC schedulers
- All fancy features are first developed for Slurm
- Currently running 18.08
- Regular updates for bug fixes and new features

How do I connect to the cluster?

- Connect to the cluster frontend node via `ssh`

```
ssh <username>@login.cluster.embl.de
```

This is the frontend node

Obtaining example program

- Use git to download

```
git clone https://git.embl.de/grp-bio-it/embl_hpc.git
```

How do I run a program on the cluster?

- Never run anything on the frontend node! (except this one time...)

```
hostname
```

- Our first **job**

```
srun hostname
```

job: a resource allocation & the **steps** run within it (just one in above)
step: single task run by scheduler
`srun` submits a job step to the cluster

Training reservation

- You only need to use this during our session today

```
srun --reservation=training hostname
```

- Isolates us from the rest of EMBL

Reservation: collection of resources reserved for particular users/groups/time period

Example program

- Program should be present in the *'exercises'* directory
- Takes two arguments
    - `-t` Time to wait in seconds
    - `-m` Amount of memory to use in MB

```
./hpc_example -t 10 -m 100
```

(Remember not to run on the login node!)

- Prints arguments to screen -> creates list -> waits -> prints memory usage -> exits

Submit example program

```
srun --reservation=training \
    ./hpc_example -t 10 -m 100
```

Submitting Example program

- **`srun`** is not convenient, use **`sbatch`** to run in background
- We need to use a script - `batch_job.sh`

```
sbatch --reservation=training \
    batch_job.sh
```

`sbatch` submits a **job script** to the cluster
job script: simple script that combines resource requests and job steps

Viewing jobs

```
squeue
```

- We can filter the list to be more specific

```
squeue --user=<username>
squeue --reservation=training
```

`squeue` lists current jobs

Examining output

- Default output is a file based on the JobID e.g slurm-15273607.out
- You can change this
- Use `srun/sbatch --output=output.txt`

```
sbatch --output=<outputfile> \
    --reservation=training \
    ./batch_job.sh
```

- Append to a file with --open-mode=append

Options in the batch script

- All options can also go in the script itself
- Start option lines with `#SBATCH`
- Note: unless you specify otherwise (using the **`--export=NONE`** option), the current working environment is inherited by your job.
    - Where possible, try to include absolute paths to executables, files, scripts, etc in your job script

Quick recap

- Don’t run things on the head node!
- Submit jobs using `sbatch` (and `srun`)
- View status of jobs with `squeue`
- Edit the location of output with `--output=<filename>`
- Options can be at command line or in script with `#SBATCH`

Questions?

Experiment with settings

- We modify our script to accept arguments
- Submit several jobs, try using more memory

```
sbatch --reservation=training \
    ./batch_job.sh 20 ???
```

reminder: the second option controls the maximum memory that the job will use

Our node has 256GB or 256,000MB

Resource management

Reserving additional resources

- Sharing resources between users is a key function of the job scheduler
- Jobs may be killed or slow down if they try to use more than their allocated share
- Use `scontrol` to view the cluster configuration & default values

```
scontrol show partition
```

`scontrol` show configuration of the cluster
partition: collection of resources with common attributes (also known as a queue)

Requesting additional resources

- Sharing resources between users is a key function of the job scheduler
- Jobs may be killed or slow down if they try to use more than their allocated share
- Try reserving an appropriate amount of memory

`#SBATCH –mem=<XXX>` *(alternative)*

```
sbatch --mem=8200 \
    --reservation=training \
    ./batch_job.sh 30 8000
```

Requesting additional resources

- Try reserving a LARGE amount of memory

```
sbatch --mem=100gb \
    --reservation=training \
    ./batch_job.sh 300 5000
```

- Look at the waiting jobs with `squeue -t PENDING`
- Only a small number of jobs will be allowed to run simultaneously

Requesting appropriate resources

- Understanding the compute requirements of your task is key to effective use of an HPC cluster
- Ask for too much
    - Your job will wait for a long time unnecessarily
    - Reserve resources you don’t need, keeping others from using them
- Ask for too little
    - Job may be killed without finishing
    - You start using resources you haven’t asked for, potentially slowing things down for everyone

Canceling unwanted jobs

- Cancel a single job

```
scancel <jobID>
```

- Cancel all jobs for a user

```
scancel –u <username>
```

Number of cores

- Many programs offer ‘multi-threading’ or ‘multi-core’
- Make sure you request this with:

```
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
```
(other integers are available)

- Be aware of the default behavior of the application!

Setting a time limit

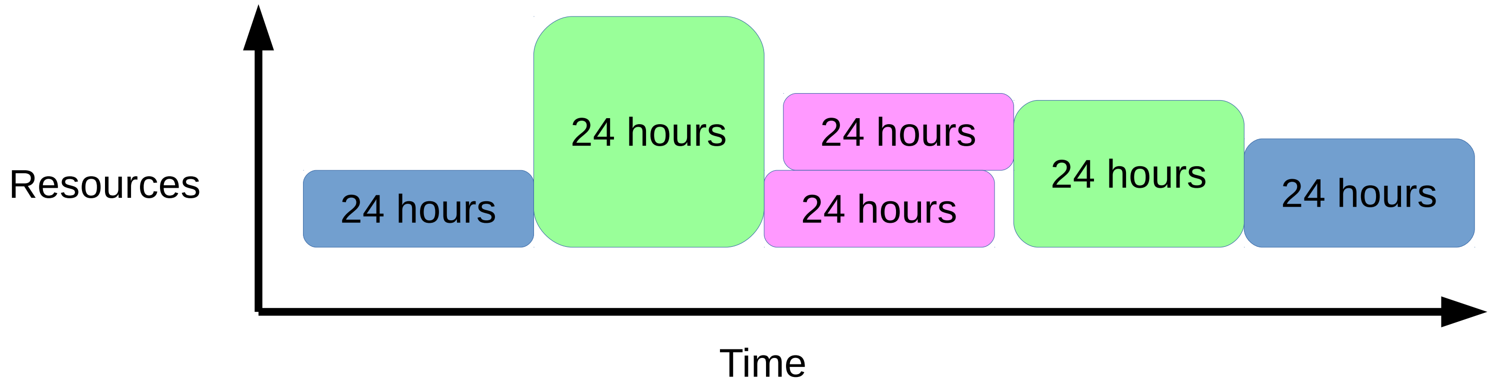
- Default time limit is 5 minutes
- Define a time limit with:

#SBATCH --time=*<DD-HH:MM:SS>*

```
sbatch --time=00-00:00:30 \
    --reservation=training \
    batch_job.sh 60 500
```

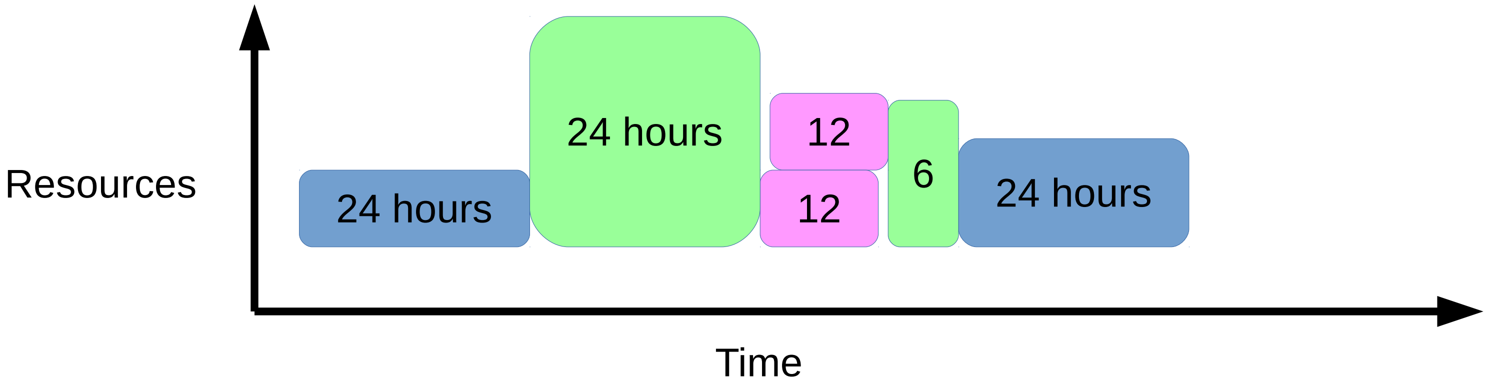
Time limits

- Providing a run time matters –
  - SLURM tries to slot short jobs into gaps
  - If every request has the same time, it can't do this



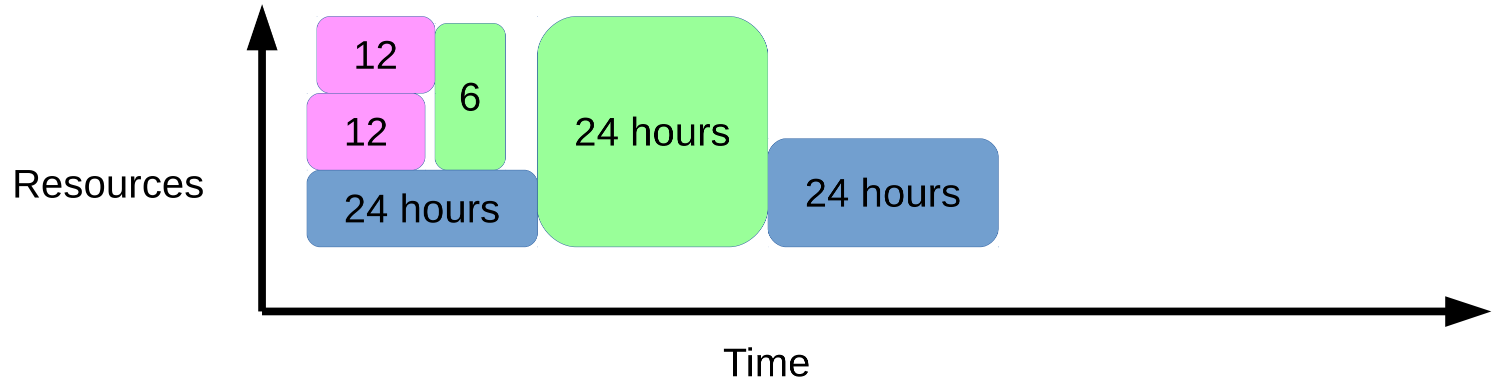
Time limits

- Default time limit is 20 minutes (will be 5 minutes soon)
- Providing a run time matters –
    - SLURM tries to slot short jobs into gaps
    - If every request has the same time, it can't do this



Time limits

- Default time limit is 20 minutes (will be 5 minutes soon)
- Providing a run time matters –
    - SLURM tries to slot short jobs into gaps
    - If every request has the same time, it can’t do this



Resources summary

- Balance between asking for enough to run your job, but not too much
- Unfortunately, determining the right amount is hard
    - Try running a few **realistic** tests
    - Read manuals – often they have some guidelines
    - If it's your software, maybe you can work from the code
- Use `seff` to report efficiency of a finished job

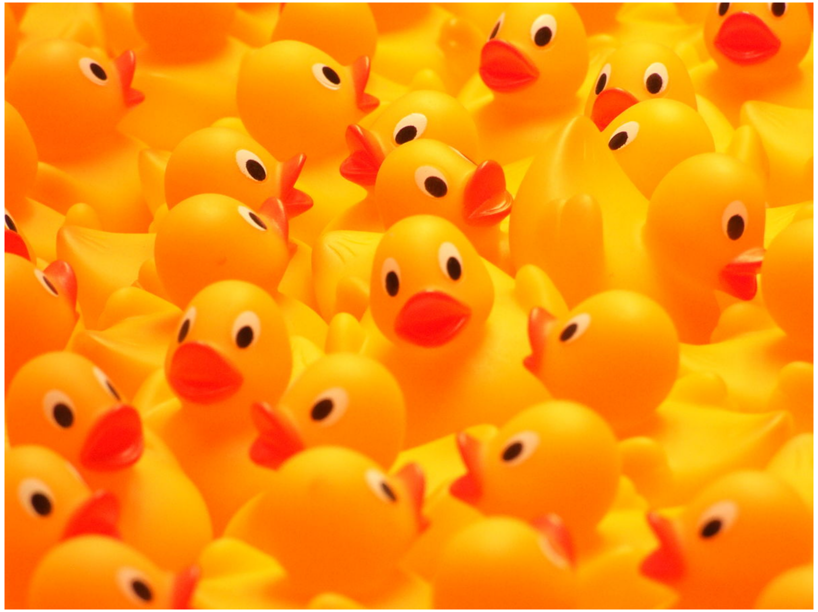
```
seff <jobid>
```

Resources summary emails

- In June 2018, usage summary emails were introduced for cluster users
- Every month, users receive a summary of their usage from slurm@embl.de
- This message includes information on the efficiency of the user’s jobs, in terms of CPU and memory used vs requested

Troubleshooting

CC-BY 2.0 https://www.flickr.com/photos/gaetanlee/298160434/



Job reporting

- You can get email notification of jobs finishing & details about their execution
- Use the **--mail-user=user@mail.com** option

```
sbatch --mail-user=<first.last>@embl.de \
    --mail-type=ALL \
    --reservation=training \
    batch_job.sh 20 500
```

- Report emails contain a lot of information
    - resource usage
    - efficiency of this usage vs what you requested

Why is my job not running?

- Slurm can tell you a reason:

```
scontrol show job <jobid>
```

- Many possible reasons:
  - Resources
  - Priority
  - Various limits

Why did my job fail?

- Use the `sacct` command to see information about recently-finished jobs

```
sacct
sacct -j <jobid>
```

- Many possible exit codes:
    - Completed is the expected one
    - Failed
    - Timeout
    - Cancelled
    - ...

More complex jobs

Batch scripts

- Batch scripts can have more than one step

- Try modifying `batch_job.sh` to run the example program twice, with different parameters

Using software

- Most commonly-used software is provided centrally, as **modules**
- To use this software, you first need to load the corresponding module

```
module load BWA
bwa index genome.fasta
```

`module load` add a specific software module to your working environment
module: package of pre-installed software, dependency-aware, optimized for hardware and environment

Using software

- Look at what modules are available with `module avail`, and search for something specific with `module spider <software>`

```
module avail
module spider samtools
```

`module avail` lists all modules (software & versions) available on the system
`module spider` search for all available modules (versions) for a particular program

Data Movement

- Always try to move data as close to compute as possible
- Nodes have >250GB of local `$TMPDIR`, use it:
    - `--tmp=50gb` (select only nodes with at least 50GB of free space)
    - `--gres=tmp:50gb` (declare your job will use 50GB of $TMPDIR)
- Copy your data to `$TMPDIR` as first step in your job
- Copy your results from `$TMPDIR` as last step of your job
- If you need more, copy your data to `/scratch`
    - Visible from all nodes
    - Each job gets a dedicated `$SCRATCHDIR`

Real world example

E.coli sequence alignment

- Look at *exercises/bwa/bwa_batch.sh*
- Multi-step job with data movement, software loading and resource requirements

Conclusions

- Head node is for job submission only
- Remember the cluster is shared between all EMBL users
- Understanding the requirements of your jobs is key
    - This can be hard :(
    - Doesn’t need to be super precise, reasonable estimates are fine

Where to find help

- Training like this one and the one tomorrow
- Wiki: https://wiki.embl.de/cluster/
- chat.embl.org #cluster
- itsupport@embl.de
- clusterNG mailing list
- Meetings as needed
    - When there are new things to announce and explain
- Bio-IT drop-in sessions and meetings, Coding Club

Some More Advanced Things

Parallelisation/GPU/job dependencies

- If we have time to cover this stuff…

```
sacct
sacct -u username
```

Backup slides

Slurm commands

- `srun` – run a single job step
- `sbatch` – submit a job script
- `scancel` – kill a running job
- `squeue` – reports the state of jobs in the queue
- `sinfo` – reports the state of queues and nodes
- `sacct` – query accounting database for info on finished jobs

Software environments

- Base OS: CentOS 7.4
- Environment modules used to enable specific software in your shell
- Software organized around toolchains
- Toolchains based on free, open source components: **foss**
- Two toolchains per year, we use components from H2 each year:
- foss/2015b (gcc 4.9)
- foss/2016b (gcc 5.4, OpenBLAS 0.2.18, FFTW 3.3.5)
- foss/2017b (gcc 6.4, OpenBLAS 0.2.20, FFTW 3.3.6)

Environment Modules

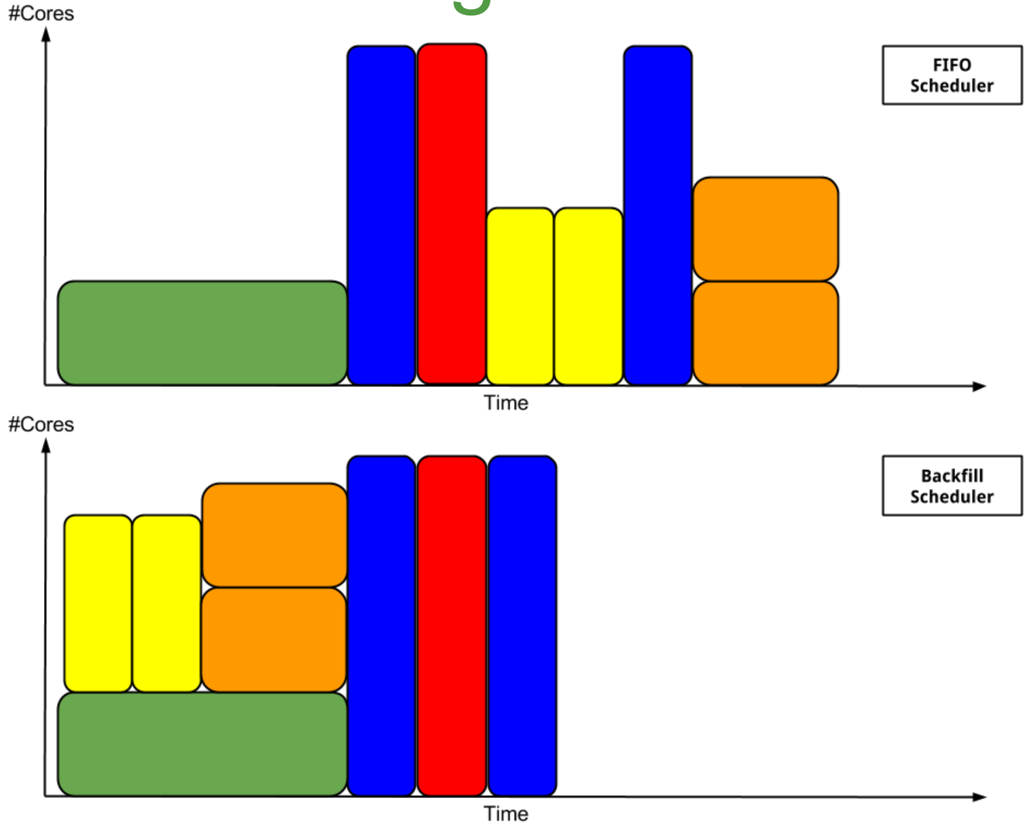
- Used with Lmod
- Provided by EasyBuild
- Repeatable software builds
- Hardware optimized builds
- Currently building for Nehalem,
- SandyBridge, Haswell and Skylake
- Large community
- Road map towards containers

Queues

- Default queue: htc
- Default run time 5 min, max runtime 20 days
- Default: 1 cpu, 2GB of memory
- Be sure to ask slurm for resources you need
- cpu, memory, time

- Hw specific:
- gpu

Backfill scheduling



For more information

- www.vi-hps.org
- www.prace-ri.eu

Exercise: login

- Use ssh to login to `login.cluster.embl.de`

Exercise: slurm resources

- View partitions: `sinfo -l`
- View node info: `sinfo -Nl`
- View node features: `sinfo -No “%N %f”`
- View reservations: `sinfo -T`

Slurm node states

- Idle
- Mixed
- Allocated
- Draining
- Drained
- Down
- Unknown
-

Exercise: modules

- List available modules: `module avail`
-
- Search available modules: `module spider <modulename>`
-
- Detailed description of a module: `module whatis <modulename>`
-
- Help for a specific module: `module help <modulename>`

Exercise: toolchains

- Run `gcc -v` and observe the version
- `module spider foss`
- `module load foss`
- Run `gcc -v` again and observe the version
- `module list`
- `module purge`
- `module list`

Exercise: dependencies

- `module load snakemake`
- `module list`
- `module load matplotlib/2.0.0-foss-2016b-Python-2.7.12`
- `module list`
- `snakemake -h`
- What happens?

How to handle that

Merit by Markus Fritz

Exercise: job environment

- `module purge`
- `module load foss`
- `srun -t 01:00 gcc -v`

Exercise: default resources

- `srun -t 05:00 --pty -E $SHELL`
-
- grep Cpus.*list /proc/self/status
-
- cat /sys/fs/cgroup/memory/slurm/uid_$(id -u)/job_$SLURM_JOBID/memory.limit_in_bytes
-
- exit

Exercise: asking for resources

- `srun -t 05:00 -N 1 -n 1 -c 4 --mem=500 --pty -E $SHELL`
-
- srun grep Cpus.*list /proc/self/status
-
- srun cat /sys/fs/cgroup/memory/slurm/uid_$(id -u)/job_$SLURM_JOBID/memory.limit_in_bytes
-
- exit

Exercise: asking for resources

- `srun -t 05:00 -N 1 -n 200 –pty -E $SHELL`
-
-

Exercise: asking for features

- `srun -t 05:00 -n 1 -c 4 -C HT --pty -E $SHELL`
- grep Cpus.*list /proc/self/status
- exit
- `srun -t 05:00 -n 1 -c 4 -C noHT --pty -E $SHELL`
- grep Cpus.*list /proc/self/status
- exit
- srun -t 01:00 -C avx512 --pty -E $SHELL
- hostname

Data movement

- Your work is highly data intensive
- Data and compute should be as close as possible to achieve best performance
- Slurm provides per-job $TMPDIR and $SCRATCHDIR
- Nodes have at least 250GB of fast TMPDIR, **use it!**
- If you can’t, use $SCRATCHDIR
-
- Use /g shares only as a source of input data and a place to store results

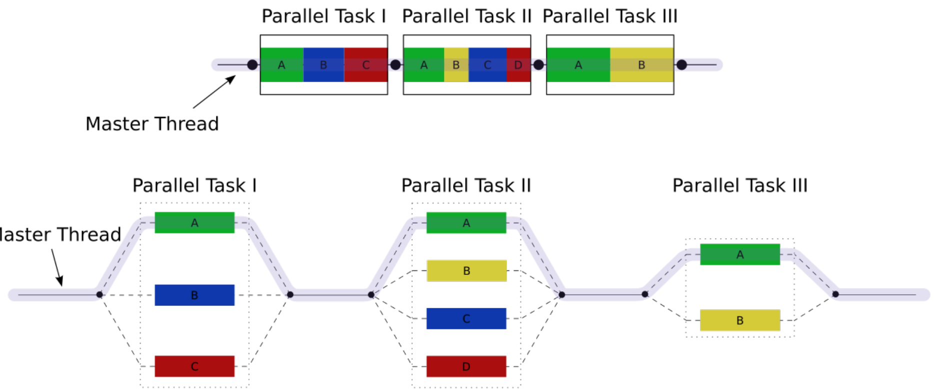
Example: Data movement

- This job script illustrates a method of copying input to many nodes

```
#!/bin/bash
#SBATCH -t 03:00
#SBATCH -N 4
#SBATCH -n 4
#SBATCH --ntasks-per-node=1
#SBATCH –tmp=50G
#SBATCH --gres=tmp:50G

#copy source data to node local tmp
sbcast /g/somewhere/project/input_data $TMPDIR/

module load …
#do stuff …

#wrap up
srun -N $SLURM_NNODES cp $TMPDIR/results /g/somewhere/p
```

OpenMP

- Shared memory parallelism
- A method to parallelize within the same node
- Obeys 10+ environment variables
- Slurm sets OMP_NUM_THREADS based on cpus requested by job



Exercise: OpenMP

- Prepare this job script
- Use `sbatch` to submit it
- Vary number of cores per task
- Observe “Number of threads” and “Best rate Triad” differences

```
#!/bin/bash
#SBATCH -t 00:01:00
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1 #vary this 1..128

module load STREAM
stream_1Kx10M
```

Exercise: notifications

- Slurm can send you emails
- They include some job efficiency statistics
- Useful to tune your exact resource request

```
#!/bin/bash
#SBATCH -t 00:01:10
#SBATCH -N 1 -n 1
#SBATCH -J stress
#SBATCH --mail-type BEGIN,END,FAIL
#SBATCH --mail-user=your.mail@embl.de

#do something
module load stress

cd $TMPDIR
stress -t 60 -c 1 -i 1 -m 1 -d 1
```

Exercise: GPU

- Slurm implements gpu as "generic resource" (gres)
- You can ask for some number of them
- Use constraint to select specific gpu model
- Check wiki for exact gpu hardware available

```
#!/bin/bash
#SBATCH -p gpu
#SBATCH -n 7
#SBATCH --mem=50G
#SBATCH -C gpu=1080Ti
#SBATCH --gres=gpu:1080Ti:2

#run relion on 7 cpu cores and 2 gpus
module load RELION

#do relion stuff ...
```

Why is my job queued?

- Your job sits in the queue in state PENDING
- Use `scontrol show job [job id]` to understand why

```
JobId=828772 JobName=CL3d_round2K2.sh
   UserId=dauden(21588) GroupId=cmueller(574) MCS_label=N/A
   Priority=3209 Nice=0 Account=cmueller QOS=normal
   JobState=PENDING Reason=Resources Dependency=(null)
   ...
```

- See `man squeue` to understand State and Reason fields

Job states

- Pending
- Running
- Completed
- Cancelled
- Failed
- Suspended
- Many more, see man squeue

Exercise: why did my job fail?

- Submit such job script
- Use sacct -j [jobid] to determine exit code and failing step
- Anything non-zero is a problem
- Standard ones defined in /usr/include/sysexits.h
- Bash has a couple of its own
- Every software can implement its own ...

```
#!/bin/bash
#SBATCH -t 00:01:00
#SBATCH -N 1
#SBATCH -n 1

#do something that fails …
exit 1
```

Best practices: Slurm

- Use your local machine or short small interactive job to experiment and test
- Use srun to run single commands from your scripts or external workflow managers (such as snakemake)
- Use sbatch and job scripts for everything where you want to preserve information about environment used (module load statements)
- Use notifications to fine tune your cpu, memory and runtime requests

Best practices: R

- While capable of using multiple threads via OpenMP, no performance benefit has been seen
- Recommend to use it with -N 1 -c 1
- If possible, try parallelizing it with MPI (at least three ways to do that)
-
- Explore alternatives (like Julia)

Best practices: GPU

- Gpu2-5 offer 28 cores and 8 GPUs
- Slurm knows which GPU is closest to which core
- If software knows about OpenMP or MPI, try to use 3-4 cores per GPU, otherwise use 1
- Best job throughput achieved with 7 cores per 2 gpus

How to approach parallelization

- Single operation over large dataset
- Think of splitting it into smaller chunks and do them at the same time
- If you’re doing things in loops, look for independent data
- Typically “for [all elements of an array] do ...”
- Figure out a way to execute these loop steps in parallel
- Use some form of shared memory model
- Parallel loop constructs
- Independent workers
- Use some tool that helps you with that

One of the options: Jug

- Demo by Renato Alves

Conclusion

- To achieve best performance:
- Put data and compute as close together as possible
- Use memory instead of filesystem
- Identify independent data and implement some parallelism on it

Q & A

Thanks