Parallelization and Responsible Resource Use on Sedna

SEDNA Computational Roundtable Series
26 June 2025

Why parallelize?

- Speed
- Necessity
- Break up a task into multiple smaller jobs, may run quicker on a busy cluster (small jobs will likely start running before resources for one big job become available)

Discussion: What types of analyses are good for parallelization

- When the overall job consists of many small jobs or discrete tasks that are independent
  - Examples:
    - Bootstrapped phylogenetics
    - Phylogenetic gene tree inference
    - Differential gene expression
    - BLAST
    - Trimming and mapping reads
    - Replicate MCMCs
    - Sliding genomic window analyses
    - Genome repeat masking

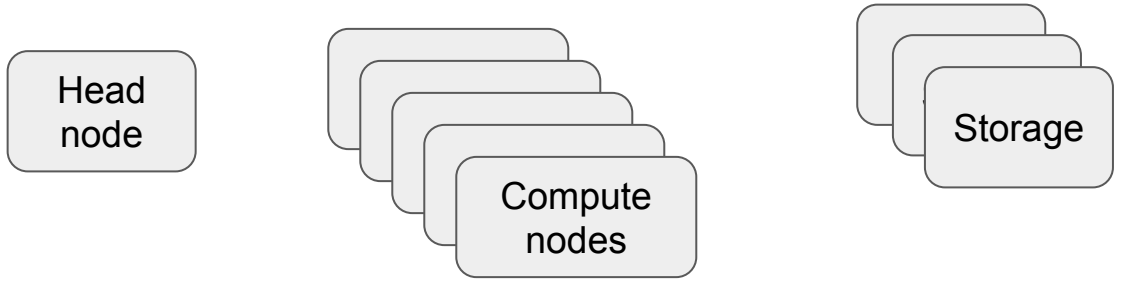
How much to parallelize?

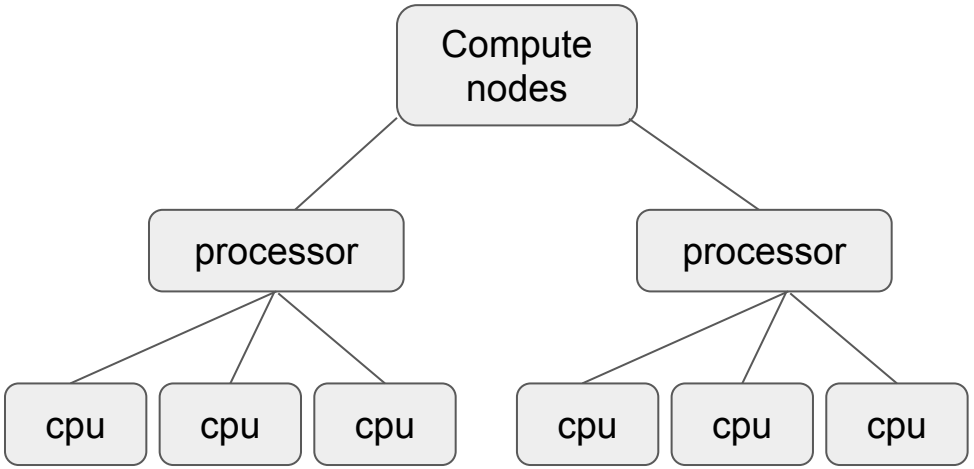
Ideally as much as possible. But…

Consider Sedna resources…

- Partitions, node, core, etc.

General HPC architecture

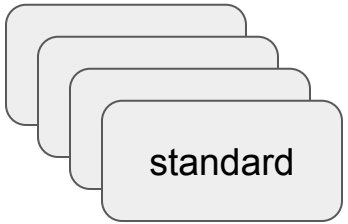


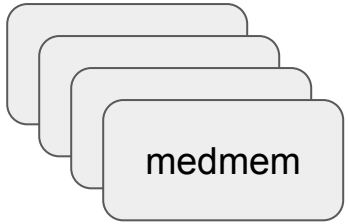


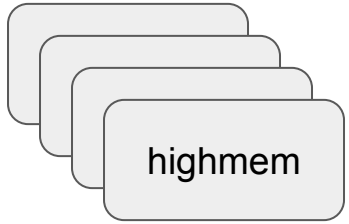
Sedna hardware

	/home	/share	/scratch1	/scratch2
Storage:	197 tb	197 tb	43 tb	91 tb

Compute Nodes







	standard	medmem	highmem
Number:	28	7	4
CPUs/node:	20	20	24
Memory/node:	96 gb	192 gb	1.5 tb

How much of Sedna can I use?

- Not all of it!
- Request the minimum resources you need for your job
  - You can check how much prior jobs have used with `seff`
- What if I have a really big job?
  - You still shouldn’t use all of Sedna.
  - But you can use more over the weekend or at night
  - Less of an impact if your jobs are short (so you can run more at once).
  - Check with other users for their plans in the SEDNA user G-chat
- Apply limits to how many jobs run at once

How to parallelize while limiting resource use

Run multiple task / processes in one slurm job:

- Gnu parallel

Parallelize with multiple jobs for each task:

- Array
- Snakemake
- Nextflow

Link to GitHub demo

Find the following examples on our github page

GNU parallel

Determine number of “jobs” to run at time (match cpu request in header)

Memory is divided by processors (# jobs indicated in script)

Feed parallel a list of file / sample names

Can test setup with: parallel --dry-run

Indicate your standard input with {} in code

```
#!/bin/bash
#SBATCH --job-name=fastp_trim
#SBATCH -c 10
#SBATCH --mem=100G
#SBATCH --partition=medmem
#SBATCH -t 1-0:0:0
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
cat Parallel_samplenames.txt | parallel -j 10 \
        fastp \
        --in1 $READS/{}_R1.fastq.gz \
        --in2 $READS/{}_R2.fastq.gz \
        --out1 $OUTDIR/{}_R1_trimmed.fastq.gz \
        --out2 $OUTDIR/{}_R2_trimmed.fastq.gz \
        --json $OUTDIR/{}_fastp.json \
        --html $OUTDIR/{}_fastp.html
```

Slurm arrays

```
#!/bin/bash
#SBATCH --job-name=fastp_array
#SBATCH -c 1
#SBATCH -t 1-0:0:0
#SBATCH --mem=10G
#SBATCH -p medmem
#SBATCH -o %x_%A_%a.out
#SBATCH -e %x_%A_%a.err
#SBATCH --array=[0-12]%13

#Create Array
FILES=$(find $READS/*_R1.fastq.gz -type f -exec basename {} _R1.fastq.gz \; )
FILES_ARRAY=(${FILES})
SAMPLE=${FILES_ARRAY[${SLURM_ARRAY_TASK_ID}]}

fastp \
        --in1 $READS/${SAMPLE}_R1.fastq.gz \
        --in2 $READS/${SAMPLE}_R2.fastq.gz \
        --out1 $OUTDIR/${SAMPLE}_R1_trimmed.fastq.gz \
        --out2 $OUTDIR/${SAMPLE}_R2_trimmed.fastq.gz \
       --json $OUTDIR/${SAMPLE}_fastp.json \
       --html $OUTDIR//${SAMPLE}_fastp.html
```

Each job gets memory and cpu indicated

Indicate how many slurm array tasks will be completed

Important: %13 means run 13 at a time. In this case all jobs but good to limit

Create a list of files and then assign each file to a slurm task id (0-12)

Indicate sample with variable

snakemake

Important: defines how many simultaneous jobs snakemake can run at a time

Command Line Launch

```
snakemake --executor slurm --default-resources slurm_account=nefsc slurm_partition=standard --jobs 10
--use-conda --config config.yaml
```

Specify configuration file

Snakemake Rule

```
rule pre_fastqc:
    output:
        "log/pre_fastqc_{sample}.done"
    conda:
        "envs/fastp-0.24.0.yaml"
    resources:
        runtime=10000,
        cpus_per_task=1,
        mem_mb=1000
    shell:
        """
        …
        """
```

Customized resources for each job detailing runtime, memory usage, and # cores

Configuration

```
runtime: 100
cpus_per_task: 1
mem_mb: 1000
```

Default runtime, memory usage, and # cores defined in config file (yaml format)

nextflow

Nextflow tips for HPC users:
https://seqera.io/blog/5_tips_for_hpc_users/

Command line launch

```
nextflow -c "/path/to/my/config.txt"
```

Usually best to specify resources in a config file rather than at the command line

Configuration file

```
executor {
   name = 'slurm'
   queueSize = 10
}
process {
   queue = 'standard'
   cpus = 8
   memory = '24 GB'
   time = '2h'
   clusterOption = "OTHER SLURM OPTIONS"

   withLabel: big_mem {
      memory = '64 GB'
      executor.queueSize = 2
    }
}
```

Important: maximum number of jobs to submit simultaneously

Default queue, # CPUs, RAM, and runtime per job

Use this for other SLURM options

Can use labels for processes that require more/less than the default resources

Check on resource usage

- Seff
  - Check job efficiency with seff <jobid>
    - CPU utilized: cpu hours used by job
    - Wall clock time: time span that job ran
    - CPU efficiency: average rate of CPU usage
    - Peak memory usage / memory requested
- Sacct
  - sacct -o jobid,jobname,nodelist,partition,state,ReqMem,MaxRSS,ReqCPUS,elapsed,Timelimit,submit -j <job id>
- Re-adjust scripts if your requests aren't efficient
- How do I find the <jobid> for a job that already finished?
  - sacct --state=CD --starttime=2025-06-01 --endtime=now
    - "--state=CD" → only show completed jobs, can use other state codes
    - "--starttime=2025-06-01 --endtime=now" → show jobs that completed between 6/1/25 and now, can adjust the date

Planning for a cloudy computing future…

- In cloud computing, our shared resource is $$$, not hardware
- Resource monitoring may be even more important if we are paying per CPU hour (time-based or usage-based pricing)
- Optimizing CPU, memory, and walltime will help minimize computing costs
- Workflow managers will be critical; Snakemake and Nextflow have built-in support for parallelization in cloud computing environments