Online Training

Session 3: Parallelism

17 March 2026

Dane Kennedy · ilifu Bioinformatics Support
University of Cape Town

View on github.io

printable version

ilifu Components (review)

Login node

- Where you land when you log in (also known as "head node")
- Run SLURM commands / submit jobs, but **not** software or heavy processes

Compute nodes

- Where your processes run (also known as "worker nodes")
- Access software via modules / Singularity containers
- 1 node = 32 CPUs, ~240 GiB RAM (Main partition)

Storage

- Permanent distributed storage (CephFS), accessible from all nodes
- Your home directory and project directories live here

Network

- Fast Ethernet connects all nodes — **no InfiniBand**
- This limits multi-node scaling (more on this later)

Partitions — check with `sinfo`

Partition	Nodes	CPUs / node	RAM (usable)
Main + Devel	~85	32	~240 GB
HighMem	8	32 / 96	503 GB / 1.5 TB
GPU	7	24–48	232–354 GB + GPUs
Jupyter	Shared with GPU nodes		

ilifu

What is Parallelism?

> **Oxford definition for parallel processing:**
> a mode of operation in which a process is split into parts, which are executed simultaneously on different processors attached to the same computer [or different computers attached to the same cluster].

- A cluster includes many connected nodes, each with its own RAM & CPUs
- A node = single computer / server / VM / machine / box
- The work is partitioned into smaller jobs, sometimes with a partition of the dataset

Parallelism (Wikipedia)

ilifu

Serial vs Parallel Program

Serial: one step at a time

```
1 grades = [50, 65, 82, 79, 48, 72]
2 total = 0
3 for grade in grades:
4     total = total + grade
5 average = total / number_of_grades
```

All instructions run sequentially on a single CPU.

```
grades = [50, 65, 82, 79, 48, 72]
total = 0
total = 0 + 50 = 50
total = 50 + 65 = 115
total = 115 + 82 = 197
total = 197 + 79 = 276
total = 276 + 48 = 324
total = 324 + 72 = 396
average = 396 / 6 = 66
```

Parallel: split → process → combine

```
1 grades = [50, 65, 82, 79, 48, 72]
2 chunk = len(grades) // N  # choose N=3
3 grades_part = grades[rank*chunk:(rank+1)*chunk]
4 average_part = sum(grades_part) / len(grades_part)
5 average = sum(all_averages) / N
```

All 3 workers process their chunk simultaneously.

```
grades = [50, 65, 82, 79, 48, 72]
# Split: each worker takes 2 consecutive grades
W0: grades_part = [50, 65]
W1: grades_part = [82, 79]
W2: grades_part = [48, 72]
# Process: all workers compute simultaneously
W0: avg_part = 115/2 = 57.5
W1: avg_part = 161/2 = 80.5
W2: avg_part = 120/2 = 60.0
# Combine (W0):
average = (57.5+80.5+60.0)/3 = 66
```

Split data
Divide into N chunks

→

Process simultaneously
N workers in parallel

→

Combine results
Reduce / gather

Divide-and-conquer algorithm

Parallelism Requirements

- Executing portions of program simultaneously
- Possible when we have many processors (cores/CPUs)
- Capacity dependent on structure of both hardware AND software
- Requires overall control/coordination mechanism
    - i.e. message passing in MPI / threading / OpenMP

Parallelism on the ilifu Cluster

- A cluster includes many connected nodes
- Each node has RAM and multiple cores (some nodes also have GPUs)
- Work of job is partitioned into smaller jobs
    - Usually on independent parts of the data

Single-node vs Multi-node Parallelism

Single-node (shared memory)

- Distributes work over many CPUs *within one node*
- Typically implemented using threads / OpenMP
- GPU acceleration (also single-node)

Node (32 CPUs)

Core 1 · Core 2 · … · Core 32

shared memory — fast communication

Multi-node (distributed memory)

- Distributes work over many tasks across 1+ nodes
- Requires MPI: `mpirun`, `srun`, `mpicasa`
- MPI version inside/outside container must match
- Hybrid parallelism: MPI + OpenMP / MPI + GPU

Node A
Tasks 1–4

⟷ network

Node B
Tasks 5–8

⚠ **ilifu does not have InfiniBand.** Network latency between nodes is higher than at e.g. the CHPC. Multi-node MPI adds communication overhead that can make jobs *slower* than single-node — the live demo will prove this.

Parallelism in SLURM

Type	`--nodes`	`--ntasks`	`--cpus-per-task`	Notes
Serial	1	1	1	Default; no parallelism
OpenMP / Threading	1	1	N	Set `OMP_NUM_THREADS=N`
MPI (single-node)	1	N	1	`module load openmpi` + `mpirun`
MPI (multi-node)	N	n×N	1	Cross-node overhead; use cautiously on ilifu
Hybrid MPI+OpenMP	1+	N	n	Both `ntasks` and `cpus-per-task` > 1

Tip: On ilifu, keep `--nodes=1` unless your workload genuinely requires multi-node MPI. Single-node processes communicate via shared memory — no network latency.

ilifu

Key SLURM Parameters

```
#SBATCH --nodes=                # number of nodes allocated
#SBATCH --cpus-per-task=        # CPUs per task (threading/OpenMP)
#SBATCH --mem-per-cpu=          # memory per CPU
#SBATCH --mem=                  # memory per node
#SBATCH --ntasks-per-node=      # number of tasks per node
```

Parameter	Controls	Typical value
`--nodes`	Number of machines	1 unless multi-node MPI
`--ntasks-per-node`	MPI processes per node	≤ 32 (CPUs per node)
`--cpus-per-task`	Threads per MPI process	≥ 2 for OpenMP / hybrid
`--mem`	RAM per node	Total = N × `--mem`
`--mem-per-cpu`	RAM per CPU	Alternative to `--mem`

SLURM parameters docs

SLURM — Serial and Multi-CPU Jobs

Serial job (1 CPU)

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1

python myscript.py
```

OpenMP / threading job (N CPUs)

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=N
#SBATCH --mem-per-cpu=XGB

export OMP_NUM_THREADS=N
python myscript.py
```

When to use each

- **Serial:** code has no internal parallelism — increasing CPUs or nodes will *not* help
- **OpenMP:** code uses threading internally (e.g. numpy, scikit-learn, many bioinformatics tools like BWA, STAR)

Watch out for

- Always set `OMP_NUM_THREADS` explicitly — some software defaults to *all* cores on the node, not just your allocation
- Maximum 32 CPUs per node on the Main partition

> **Note:** The maximum CPUs per node (32) will not always give maximum speedup — test with fewer cores first.

SLURM — Multi-task (MPI) Jobs

Using `--ntasks` (SLURM allocates nodes)

```
#SBATCH --ntasks=N
#SBATCH --cpus-per-task=1
#SBATCH --mem=XGB

module add openmpi
mpirun python myscript.py
```

Using `--ntasks-per-node` (you control layout)

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=N
#SBATCH --cpus-per-task=1
#SBATCH --mem=XGB

module add openmpi
mpirun python myscript.py
```

Key differences

- **`--ntasks=N`** — SLURM decides how to spread N tasks across nodes; may use multiple nodes
- **`--ntasks-per-node=N`** — you specify exactly how many tasks per node; requires `--nodes` to be set

On ilifu

- Prefer `--nodes=1 --ntasks-per-node=N` to stay on one node
- Avoids accidental multi-node allocation that adds latency
- N cannot exceed CPUs per node (≤ 32 on Main)

SLURM — Multi-node and Hybrid Jobs

Multi-node MPI

```
#SBATCH --nodes=N
#SBATCH --ntasks-per-node=n
#SBATCH --cpus-per-task=1
#SBATCH --mem=XGB
module add openmpi
mpirun python myscript.py
```

N nodes × n tasks each = N×n total MPI processes.

`--mem` is *per node*, so N×XGB total RAM.

Hybrid MPI + OpenMP

```
#SBATCH --ntasks=N
#SBATCH --cpus-per-task=n
module add openmpi
mpirun python myscript.py
```

N MPI processes × n OpenMP threads each.

Total cores = N×n.

> **Remember:** Inter-node communication is slower than intra-node on ilifu. Always benchmark before scaling to multiple nodes.

💻 Live Demo

Demo Repository

Best Practices

✗ Don't

- Run software / heavy processes on the login node
- Transfer data via the login node — use `transfer.ilifu.ac.za`
- Over-allocate resources and hog the cluster
- Use `srun` / screen / tmux / mosh for long jobs (volatile — job dies if connection drops)
- Store large files in your home directory (`/users`)

✓ Do

- Use `sbatch` for all persistent job submission
- Use `transfer.ilifu.ac.za` for external/internal data transfers
- Check available resources with `sinfo` before large jobs
- Request realistic walltime and memory — shorter = higher queue priority
- Clean up: old raw data, temp products, `/scratch` files
- Use Singularity containers for reproducible software environments

Best Practices docs

Thank you for your time!

We hope this presentation was helpful.

Contact Support

support@ilifu.ac.za

Documentation

docs.ilifu.ac.za