Demystifying HPC

How to Use and Optimize for HPC Systems

Chris Stylianou

Research Engineer

CaSToRC

eurocc.cyi.ac.cy

A Bit About Myself

- Research Engineer at CaSToRC, PhD
- *Training and Skills Development* Task Leader for EuroCC2
- Area of expertise in High Performance Computing (HPC)
- Contact & Info:
  - Email: c.stylianou@cyi.ac.cy
  - Website: cstyl.github.io

You can find the slides at: https://github.com/CaSToRC-CyI/EuroCC-HPC-with-Python-2025

Why Port an Application to an HPC System?

Accelerated Simulations

Solve complex simulations faster by leveraging parallel processing

Enhanced Model Complexity

Simulations with higher spatial and temporal resolution

Train large-scale ML models (Bln/Tln Parameters)

Ability to Tackle Larger Problems

Scale to larger domain sizes that are infeasible on local machines

Process massive datasets

Cost-Effectiveness

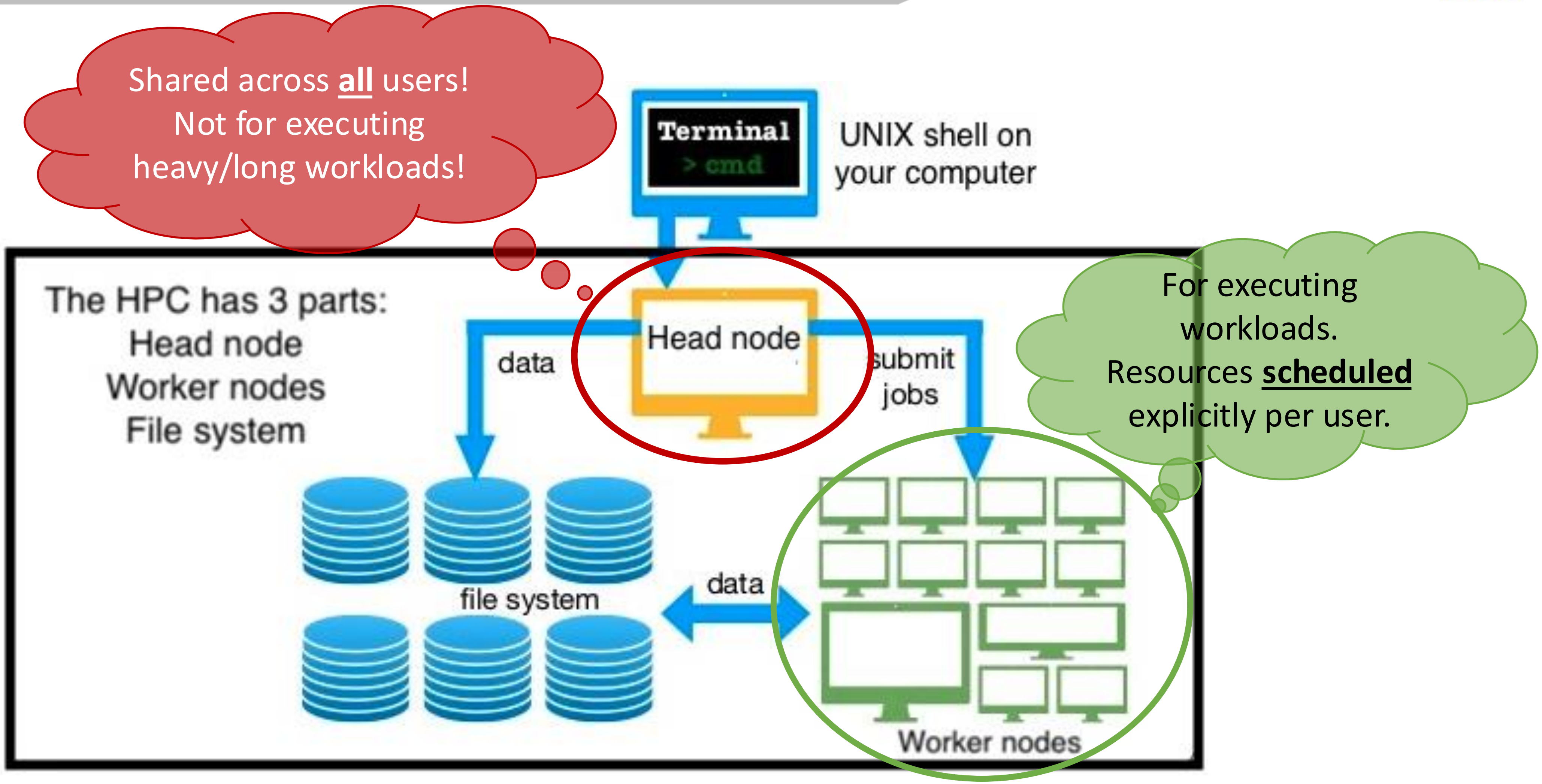
Reduce the need for physical experiments by conducting virtual testing

Reduced need to invest in high-end computing infrastructure.

Supercomputers	Cloud Computing

Specialized Hardware:
• Optimized for **parallel** and **capability computing**
  • solving the **largest** and most **complex problems**
• High-speed interconnects.
	General-Purpose Hardware:
	• Flexible and scalable **virtual machines**.
	• Designed for **capacity computing**
	  • handling many smaller tasks concurrently.
	• Standard networking and storage infrastructure.

Purpose: • Focused on scientific research, simulations, and high-performance tasks.	**Purpose:** • Ideal for diverse workloads, including web applications, and general-purpose computing.
Resource Allocation: • Fixed and highly controlled environment. • Users **share resources** via job **schedulers** like SLURM.	**Resource Allocation:** • On-demand provisioning with **pay-as-you-go pricing**. • Easily scales up or down based on need.


Shared across **all** users!
Not for executing heavy/long workloads!
Terminal
> cmd
UNIX shell on your computer
The HPC has 3 parts:
Head node
Worker nodes
File system
data
Head node
submit jobs
For executing workloads.
Resources **scheduled** explicitly per user.
file system
data
Worker nodes

Essential Skills for Running on a Supercomputer

Access and Setup	Secure access with credentials (e.g., SSH keys). Understand system layout (home, scratch, project directories).
Command-Line Basics	Navigate and manage files using Linux commands. Edit scripts with tools like vim, nano or Emacs.
Job Submission	Write job scripts specifying CPUs, GPUs, and memory. Submit (sbatch) and monitor (squeue) jobs with a scheduler.
Software Modules	Load required software (module load).
Data Management	Transfer files efficiently (e.g., scp, rsync). Organize inputs/outputs and monitor storage usage.

How to Access Cyclone

- Use **Secure Shell (SSH)** to log in to the supercomputing system.
- Requires a **username** and **password** or **key-based authentication**.

Tools for SSH by Platform:

- **Linux/Mac**: SSH is pre-installed; use the terminal.
- **Windows**:
  - Use tools like **PuTTY** or **Windows Subsystem for Linux (WSL)** for SSH access.
  - **Tip**: Install OpenSSH through PowerShell or the Settings menu.
    - Requires **Administrative priviledges**!
- **Example SSH Command:**

```
$ ssh -i /path/to/ssh/private/key cstyl@cyclone.hpcf.cyi.ac.cy
```

Accessing the System

```
cstyl@cstyl-2:~$ ssh -i ~/.ssh/cyclamen cstyl@cyclone.hpcf.cyi.ac.cy
   _____            _
  / ____|          | |
 | |    _   _  ___| | ___  _ __   ___
 | |   | | | |/ __| |/ _ \| '_ \ / _ \
 | |___| |_| | (__| | (_) | | | |  __/
  \_____\__, |\___|_|\___/|_| |_|\___|
         __/ |
        |___/


NONE of the data on this system is backed up. Please keep backups of valuable
 files at alternative locations. Maintaining backups is the sole responsibili
ty of the user.


Welcome to the Cyclone HPC system!
- System Documentation: https://hpcf.cyi.ac.cy/documentation/index.html
- Support: Email to hpc.support@cyi.ac.cy or open a ticket here:
  https://hpcf-support.atlassian.net/servicedesk/customer/portal/3
- Use "module avail" and "module load" to list/use available software
- You can now use the "qhist" command to check your project's usage
- Utilize --reservation=short for 1-hour jobs on specific nodes during
  work hours for no queue time

Last login: Thu Nov 21 16:06:56 2024 from 46.199.227.111
(base) [cstyl@front02 ~]$
```

Key Linux Commands:

Command	Purpose	Example
ls	List directory contents	`ls -l`
cd	Change directory	`cd /path/to/directory`
pwd	Print the current directory path	`pwd`
cp	Copy files or directories	`cp file1 file2`
mv	Move or rename files or directories	`mv oldname newname`
rm	Remove files (use cautiously!)	`rm file`

Best Practices:

- Use `ls` and `pwd` to verify your current location.
- Avoid running `rm` without confirming the file path.

Navigating Directories

```
(base) [cstyl@front02 ~]$ pwd
/nvme/h/cstyl

(base) [cstyl@front02 ~]$ ls -l
total 3783350
lrwxrwxrwx  1 cstyl qcd          15 Jun 14  2022 data_p069 -> /onyx/data/p069
lrwxrwxrwx  1 cstyl qcd          15 Jul 10  2023 data_p163 -> /onyx/data/p163
lrwxrwxrwx  1 cstyl qcd          15 Jul 11  2023 data_p165 -> /onyx/data/p165
lrwxrwxrwx  1 cstyl qcd          20 Apr 17  2024 edu20 -> /nvme/scratch/edu20/
drwxr-xr-x  4 cstyl qcd           9 Apr 27  2024 gpt-fast
lrwxrwxrwx  1 root  root         20 Mar 11  2024 scratch -> /nvme/scratch/cstyl/
drwxr-xr-x 13 cstyl qcd          15 Apr 16  2024 wee_archie
```

File System Structure

Directory Overview

Home Directory: Persistent storage for scripts and small files.

$HOME=/nvme/h/<username>

Scratch Directory: Temporary, high-speed storage for job data.

$SCRATCH=/nvme/scratch/<username>

Shared Directory: Collaboration space for team projects.

$DATA_<pid>=/onyx/data/<pid>

Best Practices for File Management

Store **active jobs** in the **scratch directory**.

Store **source code** and **build executables** in **home directory**.

Store **large** shared project **data** in **shared directory.**

Move important results to **home** or external storage to prevent loss.

Note: NO BACKUPS

What Are Modules?

- **Modules** manage software environments, ensuring compatibility with HPC resources.
- Load, switch, or unload software dynamically.
- **Note: No sudo access on HPC Systems!**

Key Commands:

Command	Purpose	Example
module avail	List all available software modules	module avail
module load	Load a specific software module	module load gcc
module unload	Unload a specific module	module unload gcc
module list	List currently loaded modules	module list

Tip: Check module dependencies to avoid conflicts.

Managing Modules

```
(base) [cstyl@front02 ~]$ python --version
Python 3.10.13

(base) [cstyl@front02 ~]$ module avail Python/
----------------------------- /eb/modules/all -----------------------------
   Python/2.7.16-GCCcore-8.3.0
   ...
   Python/3.10.8-GCCcore-12.2.0
   Python/3.11.5-GCCcore-13.2.0

(base) [cstyl@front02 ~]$ module load Python/3.11.5-GCCcore-13.2.0

(base) [cstyl@front02 ~]$ python --version
Python 3.11.5
```

What is SLURM?

Definition

SLURM (Simple Linux Utility for Resource Management) is a job scheduler that allocates resources and manages workloads on supercomputers.

It ensures efficient sharing of resources among multiple users.

Key Roles

Resource Allocation: Assigns CPUs, memory, GPUs, and other resources to jobs.

Job Scheduling: Prioritizes jobs based on factors like resource availability and queue position.

Monitoring: Tracks the status of jobs and manages failures or cancellations.

Why SLURM?

Maximizes system utilization.

Ensures **fair resource distribution**.

A SLURM script is a text file with commands that specify:

1. **Job Information**: Job name, output files, email notifications.
2. **Resource Requests**: CPUs, memory, GPUs, and wall time.
3. **Execution Commands**: The actual program or script to run.

```bash
#!/bin/bash
#SBATCH --job-name=example_job     # Job name
#SBATCH --output=output.txt        # Output file
#SBATCH --error=error.txt          # Error file
#SBATCH --ntasks=1                 # Number of tasks (CPUs)
#SBATCH --mem=4G                   # Memory allocation
#SBATCH --time=01:00:00            # Time limit (hh:mm:ss)

module load Python                 # Load required module

python script.py                   # Run Python script
```

Example Workflow:

1. Submit a job using sbatch.
2. Monitor progress with squeue.
3. Cancel if necessary with scancel.

Job Status Symbols:

- **R**: Running.
- **PD**: Pending (waiting in queue).
- **F**: Failed.

Category	Command	Action
Job Submission	sbatch <job_script>	Submit a batch job
	srun <command>	Run a command in parallel
Job Monitoring	squeue	View queued jobs
	squeue --user=$USER	View MY queued jobs
	scontrol show job <job_id>	Detailed job info
Job Management	scancel <job_id>	Cancel a job
	scontrol hold <job_id>	Hold a job
	scontrol release <job_id>	Release a held job
Resource Allocation	sinfo	Information about nodes and partitions
	scontrol show node <node_id>	Show Node details

Best Practices:

- Request only what your job needs to avoid wasting resources.
- Use --time to limit job runtime and prioritize efficiency.
- Use --account to specify which project to “charge”.

Resource	Description	Job Specification
Nodes	Number of Nodes	--nodes
CPU tasks	Number of CPU tasks per node	--ntasks-per-node
CPU threads	Number of CPU threads (cores) per task	--cpus-per-task
Memory	Amount of RAM per Job	--mem
GPUs	Request for GPUs if needed	--gres
System Partition	Either use the CPU or GPU part of the system	--partition

Example Use Case: Running a Python Script

Command Workflow:

1. Write the submission script (python_job.sh)
2. Submit: **sbatch** python_job.sh
3. Monitor: **squeue** -u <username>
4. Cancel (if needed): **scancel** job_id

Output:

- Job results and logs will appear in python_output.txt, errors in python_error.txt.
- Check logs for troubleshooting if the job fails.

```bash
#!/bin/bash
#SBATCH --job-name=python_job
#SBATCH --output=python_output.txt
#SBATCH --error=python_error.txt

#SBATCH --partition=cpu
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=2G
#SBATCH --time=00:30:00

module load Python
python example_script.py
```

Example of squeue Output

```
[(base) [cstyl@front02 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
  998469_[19-25%1]       cpu CNO_300K sbhowmic PD       0:00      1 (JobArrayTaskLimit)
   1007611_[4-5%1]       cpu Ag1H+_D1 sbhowmic PD       0:00      1 (JobArrayTaskLimit)
           1008164       cpu enthalpy  cy22kp1 PD       0:00      2 (Dependency)
           1008163       cpu analysis  cy22kp1 PD       0:00      1 (Dependency)
 1008403_[141-146]       cpu nanli44c  cy24nl1 PD       0:00      1 (Priority)
 1008402_[131-136]       cpu nanli44c  cy24nl1 PD       0:00      1 (Priority)
   1008392_[66-68]       cpu nanli44c  cy24nl1 PD       0:00      1 (Resources)
         998469_18       cpu CNO_300K sbhowmic  R   15:45:59      1 cn09
         1007611_3       cpu Ag1H+_D1 sbhowmic  R   10:08:58      1 cn02
           1008334       cpu data_min  jmoreno  R   23:43:31      1 cn06
           1008525       cpu       zn  eg20ra1  R   17:30:39      2 cn[15-16]
        1008392_65       cpu nanli44c  cy24nl1  R    7:51:38      1 cn13
        1008392_63       cpu nanli44c  cy24nl1  R   14:24:14      1 cn05
        1008392_64       cpu nanli44c  cy24nl1  R   14:24:14      1 cn12
        1008392_61       cpu nanli44c  cy24nl1  R   17:12:05      1 cn14
        1008392_62       cpu nanli44c  cy24nl1  R   17:12:05      1 cn17
        1008392_59       cpu nanli44c  cy24nl1  R   17:32:16      1 cn03
        1008392_60       cpu nanli44c  cy24nl1  R   17:32:16      1 cn04
```

Why Use Alternative Tools?

- Enhance **productivity** with **user-friendly interfaces**.
- **Simplify file management** and script editing for supercomputing workflows.
- Provide advanced features like **SSH integration**, **file transfer**, and **remote execution**.

Tool	Purpose	Best For	Platform
VS Code	Remote editing and debugging	Writing scripts, interactive debugging, SSH access	**Windows, Mac, Linux**
MobaXTerm	SSH client with GUI	SSH access, file management	Windows
FileZilla	File transfer	Transferring data to/from supercomputers	Windows, Mac, Linux

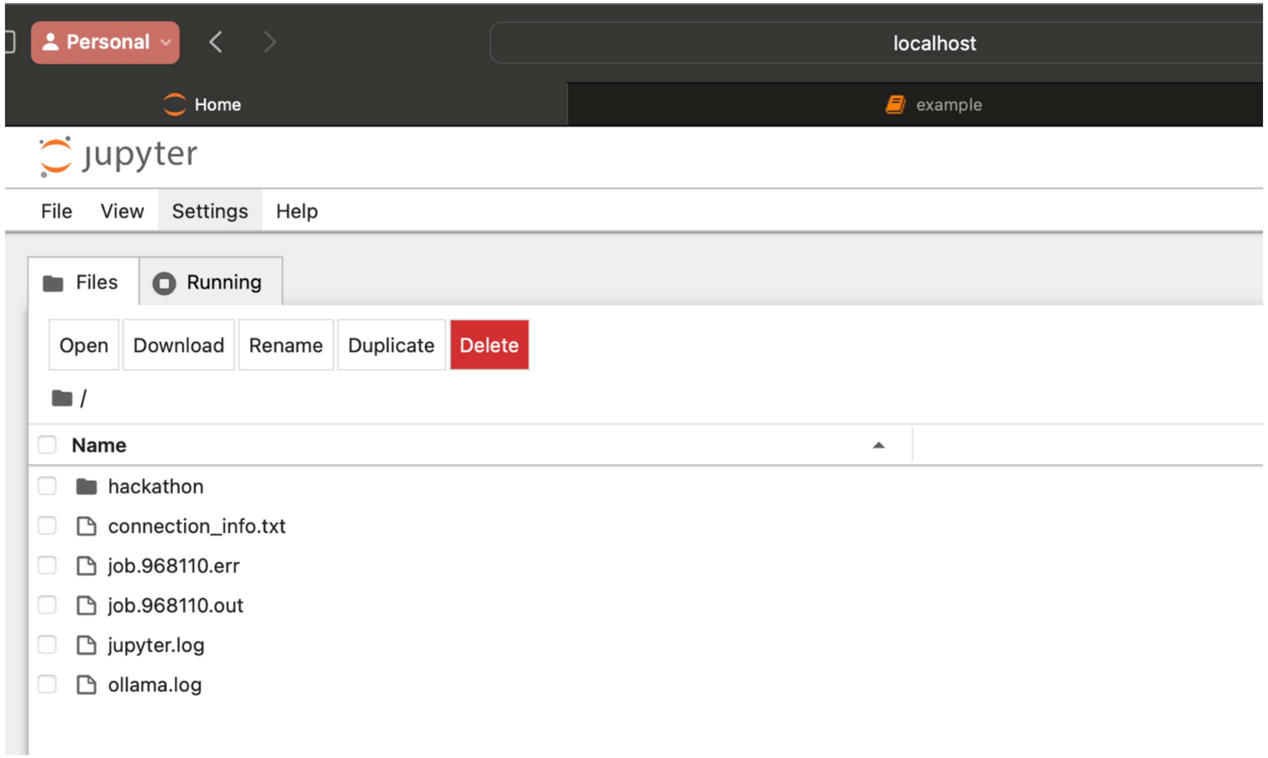
Jupyter Notebooks on HPC

Advantages of Jupyter Notebooks:

- **Interactive Workflows**: Combines code, visualizations, and narratives in a single interface.
- **AI and Data Analysis**: Perfect for tasks like data exploration, visualization, and model development.
- **Ease of Use**: Intuitive, web-based platform that lowers the learning curve for complex workflows.

Why Use HPC for Jupyter?

- Access to powerful compute resources (e.g., CPUs, GPUs).
- Handle large-scale datasets beyond the capabilities of local machines.
- Perform **AI training** and **numerical simulations** interactively and efficiently.



Maximum Wall Time for Jobs

Jobs are subject to a maximum runtime (e.g., 24 hours on Cyclone).

Long-running tasks may need checkpointing or splitting.

Compiler/Software Versions

Availability is restricted to pre-installed compilers and software.

Users must adapt to available versions or request installations.

No Sudo/Root Access

Administrative privileges are not granted.

Custom software or dependencies must be compiled within user space.

Queue Wait Times

Jobs may experience delays due to high demand for resources.

Resource availability depends on job priority and system load.

Shared Resources

Performance may vary due to shared compute and storage infrastructure.

Careful resource allocation is required to avoid bottlenecks.

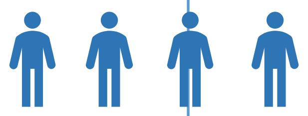
Steps to Scaling your Application on HPC Systems (1/2)

Start with Serial Execution

- Begin with a single-threaded version of your application - Verify correctness and establish a performance baseline.
- **Development Time: Low** – Minimal effort to get a working version.

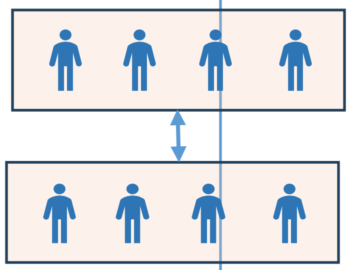
Transition to Multi-Threading (Single Node)

- Use parallel programming models like OpenMP for **shared memory** - Optimize for **multi-core CPUs** on a single node.
- **Development Time: Moderate** – Requires identifying parallel regions and potential thread synchronization.



Scale to Multiple Nodes (Distributed Computing)

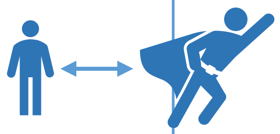
- Implement distributed memory parallelism using MPI. - Partition workloads and minimize inter-node communication.
- **Development Time: High** – Significant effort to design communication patterns and debug distributed code.



Steps to Scaling your Application on HPC Systems (2/2)

Leverage Heterogeneous Architectures (GPUs)

- **Offload compute-intensive tasks** to GPUs using CUDA, OpenACC, or HIP. - Scale further with multi-GPU setups across nodes.
- Optimize **data transfers** between host (CPU) and device (GPU).
- **Development Time: Very High** – Requires specialized knowledge of GPU programming and iterative tuning.



Profile and Optimize at Each Stage

- Profile applications after every major change to identify resource bottlenecks (CPU, memory, I/O, communication).
- Use tools like gprof, perf, or GPU-specific profilers (e.g., nvprof, Nsight).
- **Development Time: Ongoing** – Profiling and optimization are iterative and critical for scalability.

CFD Example – Workflow Overview

Problem Definition	• Define Governing Equations and set up the computational domain and boundary conditions
Discretization	• Break down the continuous domain into discrete grid. • FDM, FEM, FVM
Solver Setup	• Define Numerical Schemes for solving equations and select Solvers
Preprocessing	• Generate Computational Mesh, Initialise Parameters and Boundary Conditions
Solution Process	• Execute Solver – Iteratively solve equations and monitor convergence
Post Processing	• Visualisation of Results

CFD Example – Port and Scale a CFD Application

Understand the Serial Application

Profile the code to identify bottlenecks.

Verify scalability potential for parallelization.

Prepare for Parallelization

Refactor code for modularity.

Choose parallel paradigms: OpenMP (shared), MPI (distributed), CUDA/GPU.

Parallelize the Code

Single Node (OpenMP): Parallelize loops and hotspots.

Multi-Node (MPI): Partition domain, optimize communication.

GPU Acceleration: Offload compute-intensive tasks to GPUs.

Test and Debug

Validate accuracy against the serial version.

Debug using parallel tools.

Optimize for HPC Systems

Improve load balancing and memory usage.

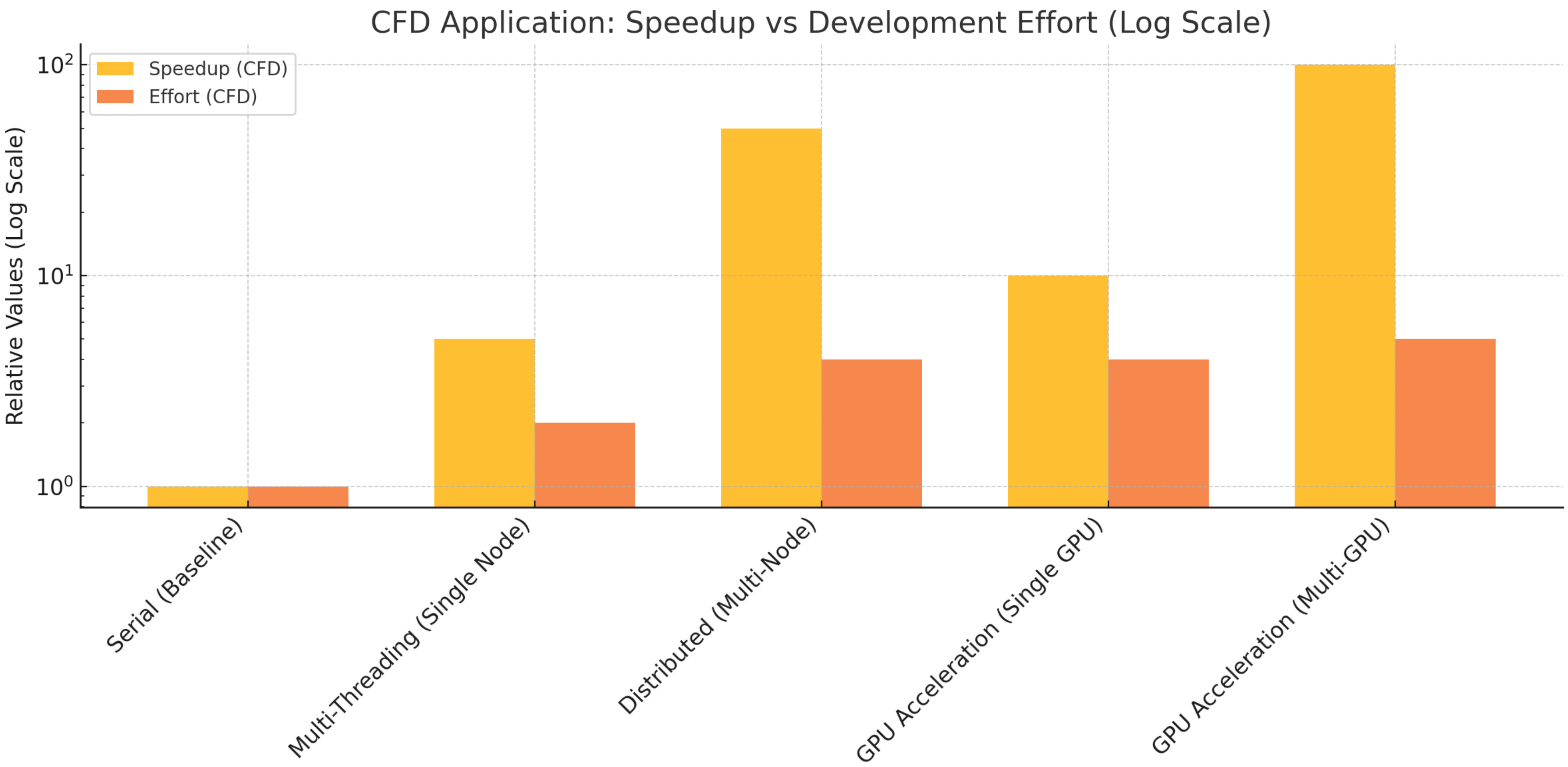
Reduce communication overhead and use optimized libraries (e.g., PETSc)

Scale the Application

Perform strong and weak scaling tests.

Adapt for multi-node and multi-GPU systems.

CFD Example – Speedup Vs Development Effort



Summary and Key Takeaways

- Supercomputers and cloud systems serve different purposes (**capability computing** vs. **capacity computing**).
- Supercomputers allow us to solve larger and more complex problems beyond the capabilities of personal computers or workstations
- Efficient navigation, resource management, and job scheduling are essential for success.
- Tools like Jupyter and alternative IDEs enhance **usability** and **productivity**.
- **Scalability is a Journey**: Porting and scaling applications require a step-by-step approach
  - From serial execution to multi-node and heterogeneous computing.
- **Development Effort vs. Performance**: Scaling offers significant benefits but requires careful planning and expertise.

Unlocking Cyclone: A Comprehensive Guide to HPC Workflows

https://castorc-cyi.github.io/eurocc-tutorials/

EURO²

Search docs

1. Introduction to HPC Systems

1.1. Overview

This tutorial provides a high-level introduction to High-Performance Computing (HPC) systems, with a focus on Cyclone's architecture and operational principles. Participants will explore the fundamental components of an HPC system, including compute nodes and file systems, while gaining insight into data management policies. Additionally, the tutorial introduces key concepts such as software management through modules, and job scheduling using SLURM, setting the stage for deeper exploration in subsequent tutorials.

1.2. Learning Objectives

By the end of this tutorial, participants will be able to:

1. Describe the architecture of an HPC system, including Cyclone's compute nodes and interconnects.
2. Identify and understand the use cases of Cyclone's file systems (home, scratch, shared directories).
3. Identify when to use an HPC system or alternative solutions such as Cloud systems or High-end Workstations.
4. Recognize the role of modules in managing software environments and how they simplify system use.
5. Understand the purpose of job scheduling and the function of SLURM in resource management.

1.3. Overview of HPC Architectures

An HPC system is typically composed of multiple interconnected **compute nodes** which work in

Thank you for your attention!

More information:

https://eurocc.cyi.ac.cy/
https://www.linkedin.com/company/eurocc2

Contact us at:
eurocc-contact@cyi.ac.cy

Co-funded by the European Union

Republic of Cyprus

RESEARCH & INNOVATION FOUNDATION

EuroHPC Joint Undertaking

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus (co-funded by the EU within the framework of the Cohesion Policy Programme "THALIA 2021-2027"), Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.