

High Performance Computing: tutorial on using Deucalion

Rodrigo Ventura

Institute for Systems and Robotics - Lisbon

Instituto Superior Técnico

July 2025

`rodrigo.ventura@tecnico.ulisboa.pt`

Introduction to the Deucalion

Deucalion supercomputer

Located at University of Minho, co-founded by FCT and EuroHPC

257th in processing speed in the TOP500 (Nov-2024), peak of 10 PetaFLOPS

Operational since 2024

Main specs:

- 2165 nodes (ARM and x86)
- 132 GPU boards (NVIDIA A100)
- 11 PB storage (SSD and HDD)

FCCN

fct



Deucalion supercomputer

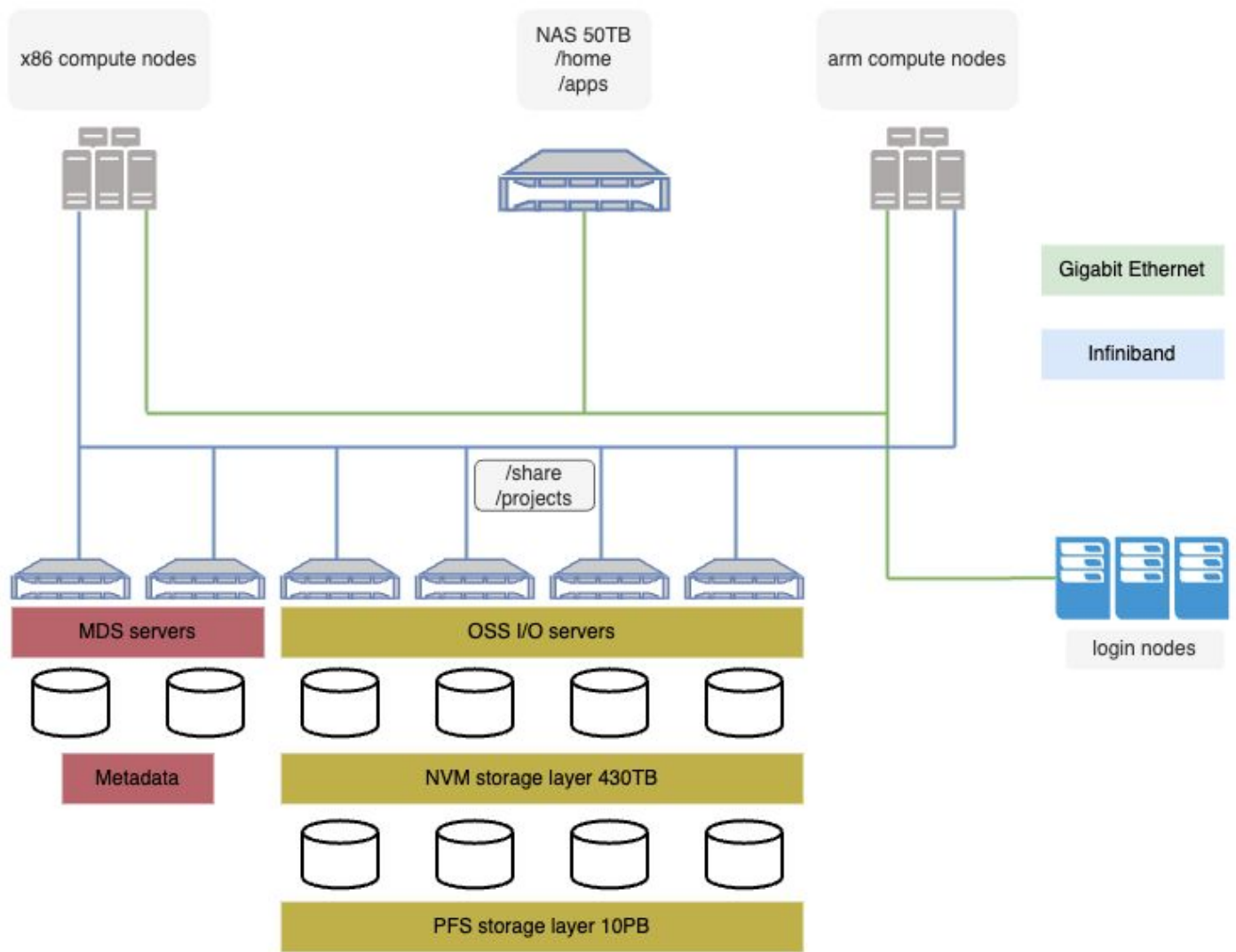
Partitions:

- ARM: 1632 nodes (Fujitsu A64FX, 48 cores)
- x86: 500 nodes (AMD EPYC, 64 cores)
- GPU: 33 nodes (x86), each with 4x Nvidia Ampere A100 40 GB or 80 GB

Storage:

- 430 TB High-speed NVMe partition
- 10.6 PB high-speed based Parallel File System partition

System architecture



How to get access

EuroHPC open calls

URL: <https://eurohpc-ju.europa.eu>

FCT calls

URL: <https://www.fct.pt>

HPCvLAB @ ULISBOA

URL: <https://forms.office.com/e/ucGDiRESgC>

Point of contact: Thales Silva (IPFN)



How to get access

Once you get the “Welcome” message, follow the steps for registration:

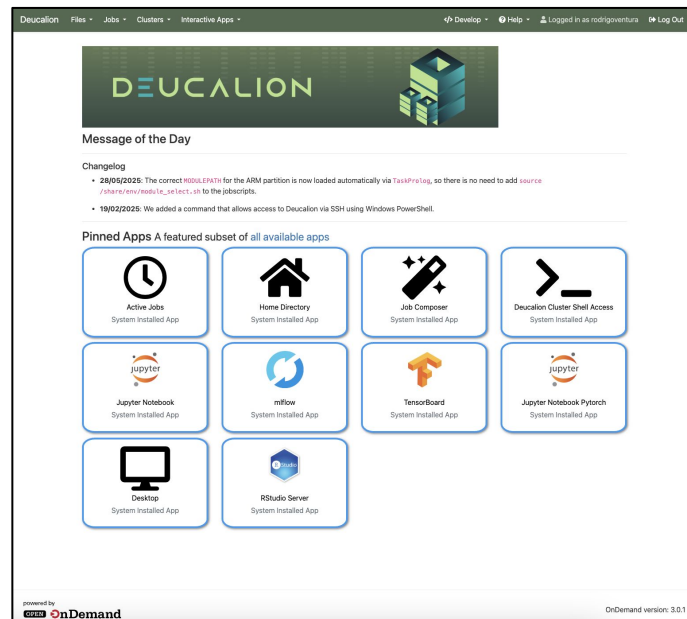
URL: <https://docs.macc.fccn.pt/start/> (included in the “Welcome” e-mail)

After registration you'll get:

- login credentials to the web portal at
URI: <https://login.deucalion.macc.fccn.pt/>
- possibility of setting up a SSH key pair
(needed for ssh access)

Main documentation site:

URL: <https://docs.macc.fccn.pt/>



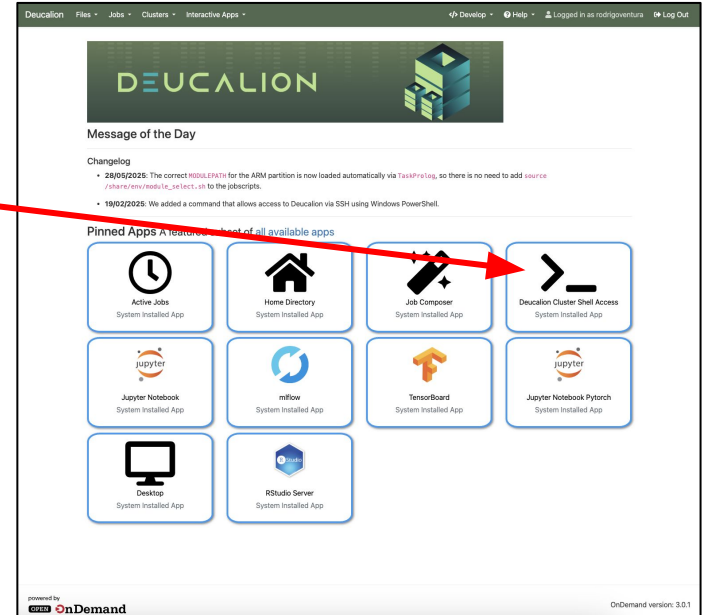
User interface

Most direct access is via ssh

```
$ ssh username@login.deucalion.macc.fccn.pt
```

As an alternative, there is a web-based ssh access here

IMPORTANT: this accesses a special login node that should **not** be used for computing!



Modules

Software is available as loadable modules.

Modules allow several combination of software versions.

- `module help` – provides documentation on this command
- `module avail` – list all available modules
- `module spider <name>` – search for modules related to <name>
- `module load <name>` or `ml <name>` – loads module <name>
- `module unload <name>` – unloads module <name>
- `module purge` – unloads all modules
- ...

Modules

Overview of

available modules:

			EasyBuild					
ACTC	(1)	Graphviz	(1)	Perl-bundle-CPAN	(2)	ensemblen	(1)	libxslt
ADIOS	(1)	HDF	(1)	Perl	(11)	expat	(7)	libxsm
AMSYS	(1)	HDF5	(7)	Pillow-SIMD	(1)	expecttest	(3)	libyaml
ATK	(1)	HOMER	(2)	Pillow	(3)	fastahack	(1)	lxml
Abseil	(1)	HPL	(2)	Print	(1)	fermi-lite	(1)	lz4
AmberTools	(1)	HTSLib	(2)	PnetCDF	(1)	fftwcodec	(2)	magma
Anaconda3	(1)	HarfBuzz	(3)	PostgreSQL	(2)	filevercmp	(1)	make
Armadillo	(1)	Horovod	(1)	PyFoam	(1)	flatbuffers-python	(1)	makeinfo
Autocore	(8)	ICU	(4)	PyTorch	(4)	flex	(8)	matplotlib
Automake	(8)	IDBA-UD	(1)	PyYAML	(3)	flit	(2)	maturin
Autotools	(8)	IPython	(1)	PySAM	(2)	fontconfig	(4)	mctc-lib
BSMTools	(2)	ISL	(1)	Python-bundle-PyPI	(2)	fonttools	(1)	meson-python
BLIS	(5)	ImageMagick	(1)	Python	(12)	foxs	(6)	mlpack
BMA	(1)	Imath	(1)	Ohull	(3)	freebayes	(1)	mpi4py
BamTools	(2)	Inspector	(1)	Qt5	(3)	freetype	(4)	mpiP
Base1	(1)	JasPer	(3)	QuantumESPRESSO	(1)	fson	(1)	mpmath
BeautifulSoup	(1)	Java	(2)	R	(2)	gettext	(9)	mtree
Biopython	(2)	JsonCpp	(1)	REMCORA	(5)	gfbf	(2)	multicharge
Bison	(9)	Julia	(1)	RFdiffusion	(2)	glib	(2)	multichoose
Boost	(5)	JupyterLab	(1)	Rust	(5)	git	(4)	nano
Bowtie2	(2)	KaHIP	(3)	SAMtools	(2)	gmpy2	(2)	ncurses
Brctl	(4)	LAME	(3)	SCOTCH	(3)	gnuplot	(2)	netCDF-C++4
Brunli	(1)	LERC	(1)	SCONS	(1)	gimpi	(6)	netCDF-Fortran
CDO	(1)	LLVM	(3)	SDL2	(1)	googletest	(2)	netCDF
CFITSIO	(1)	LibTIFF	(4)	SIOnLib	(2)	gperf	(4)	nettle
CGAL	(3)	Libint	(3)	SOCI	(1)	graphite2	(2)	networkx
CLHEP	(1)	LittleCMS	(3)	SQLite	(6)	groff	(6)	nlohmann_json
CMake	(9)	Lua	(2)	SSW	(1)	gzip	(5)	nodejs
CONCOCT	(1)	M4	(9)	SWIG	(1)	hatchling	(2)	numactl
CP2K	(2)	MAQAO	(1)	ScalLAPACK	(6)	help2man	(7)	numpy
CUDA	(6)	MEGANIT	(2)	Scalasca	(1)	htop	(1)	parallel
Catch2	(2)	MERIC	(2)	SciPy-bundle	(6)	hwloc	(7)	patchelf
Ceres	(1)	METIS	(3)	Score-P	(2)	hypothesis	(5)	picard
ChemFP2	(1)	MPC	(2)	SeqLib	(1)	iccifort	(1)	pixman
ChemM-Database	(1)	MPPFR	(3)	Szip	(4)	impi	(3)	pkg-config
ChemM	(2)	MVAPICH2	(1)	TAU	(2)	imkl-FFTW	(2)	pkgconf
CppUnit	(1)	Mako	(3)	TOPAS	(2)	imkl	(4)	poetry
CubaGUI	(1)	Mamba	(2)	Tcl	(6)	impi	(3)	pp1acer
CubaLib	(2)	Maven	(1)	Tk	(3)	intel-compilers	(2)	prodigal
CubaWriter	(2)	MaxBin	(1)	Tkinter	(4)	intel-tensorflow	(1)	protobuf-python
Cython	(3)	Mesa	(3)	TurboVNC	(1)	intel	(3)	protobuf
DB	(4)	Meson	(5)	UCC	(5)	intervaltree	(1)	pullseq
DBus	(2)	Mesquite	(2)	UCK-CUDA	(5)	intltool	(4)	py-cpuinfo
DendroPy	(2)	MetaBAT	(2)	UCK	(6)	jax	(2)	pybind11
Dooxygen	(6)	Mini-XML	(2)	UDUNITS	(3)	jbigkit	(4)	pydantic
ELPA	(1)	Miniconda3	(1)	UnZip	(5)	json-c	(1)	pytest-flakefinder
EasyBuild	(1)	NAMD	(4)	VCFtools	(1)	json-foretran	(2)	pytest-resunfailures
Eigen	(5)	NASM	(4)	VMD	(1)	jupyter-server	(1)	pytest-shard
FFTW-MPI	(4)	NCCL	(4)	VTune	(1)	jupyterlmod	(1)	pytest-xdist
FFTW	(6)	NLopt	(2)	WFA2	(1)	libGLU	(3)	svtcat

Introduction to SLURM

Introduction to SLURM

SLURM is an

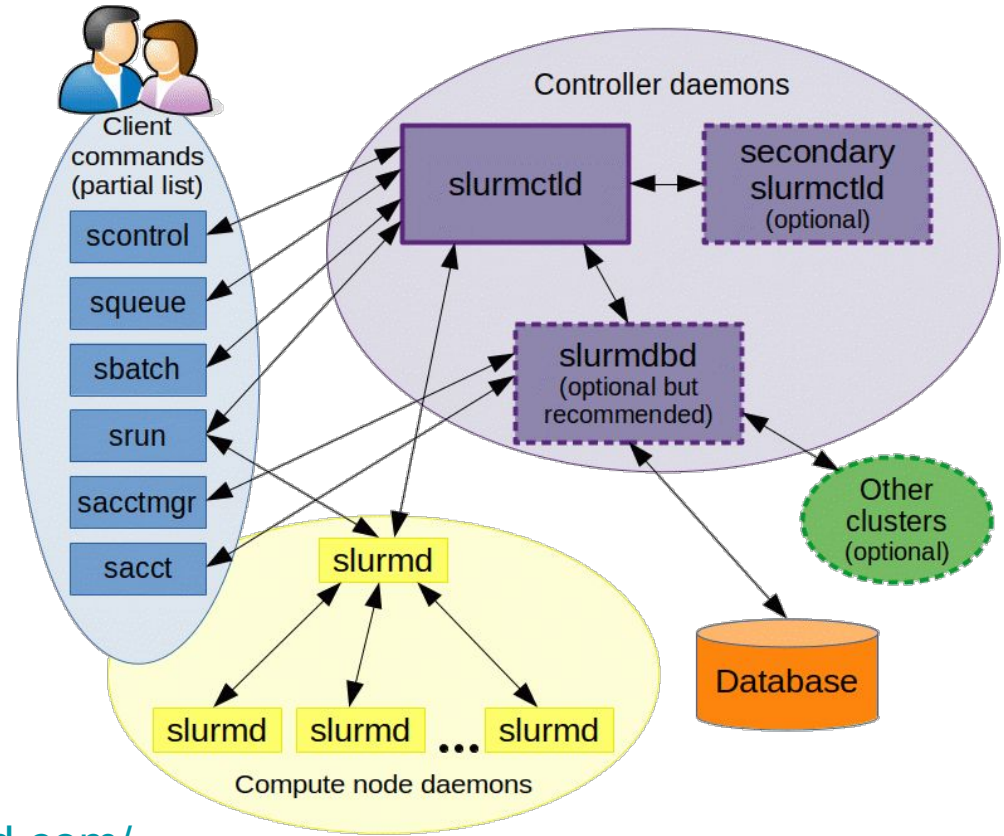
open source,

fault-tolerant,

highly scalable,

cluster management, and

job management system



URL: <https://slurm.schedmd.com/>

Introduction to SLURM

Each user is associated to one (or more) accounts:

```
$ billing
```

Account	Used (h)	Limit (h)	Used (%)	
				(ARM)
f202500002hpcvlabistula	362375	500000	72.47	(x86 / GPU)
f202500002hpcvlabistulg	543	1250	43.48	

Introduction to SLURM

The cluster features several partition, each one with a job queue:

dev-arm

16 nodes

normal-arm

1616 nodes

large-arm

1616 nodes (same nodes but more time)

dev-x86

8 nodes

normal-x86

492 nodes

large-x86

492 nodes (idem)

dev-a100-40

17 nodes

4 GPU/node

normal-a100-40

17 nodes (idem)

4 GPU/node

dev-a100-80

16 nodes

4 GPU/node

normal-a100-80

16 nodes (idem)

4 GPU/node

NOTE: Development should be done in dev-* partitions

Introduction to SLURM

```
$ sinfo --summarize
```

```
PARTITION      AVAIL  TIMELIMIT  NODES (A/I/O/T)  NODELIST
ooda           up      8:00:00      10/1/0/11  cna[0017-0027]

dev-arm        up      4:00:00      1/15/0/16  cna[0001-0016]
normal-arm     up  2-00:00:00  399/640/577/1616  cna[0017-1632]
large-arm      up  3-00:00:00  399/640/577/1616  cna[0017-1632]

dev-x86        up      4:00:00      0/8/0/8   cnx[001-008]
normal-x86     up  2-00:00:00  115/197/180/492  cnx[009-500]
large-x86      up  3-00:00:00  115/197/180/492  cnx[009-500]

dev-a100-40    up      4:00:00      4/13/0/17  gnx[502-504,506,510-513,516,522,525-531]
normal-a100-40 up  2-00:00:00  4/13/0/17  gnx[502-504,506,510-513,516,522,525-531]
dev-a100-80    up      4:00:00      9/7/0/16  gnx[501,505,507-509,514-515,517-521,523-524,532-533]
normal-a100-80 up  2-00:00:00  9/7/0/16  gnx[501,505,507-509,514-515,517-521,523-524,532-533]
```

Introduction to SLURM

How to get an interactive session in a partition – USE for debugging ONLY

```
$ hostname
```

```
ln03
```

```
$ salloc --account=f202500002hpcvlabistula --partition=dev-arm
```

```
salloc: Pending job allocation 449910
```

```
salloc: job 449910 queued and waiting for resources
```

```
salloc: job 449910 has been allocated resources
```

```
salloc: Granted job allocation 449910
```

```
salloc: Waiting for resource configuration
```

```
salloc: Nodes cna0002 are ready for job
```

```
$ hostname
```

```
cna0002.deucalion.macc.fccn.pt
```

```
$ uname --all
```

```
Linux cna0002.deucalion.macc.fccn.pt 4.18.0-348.23.1.el8_5.aarch64 #1 SMP
```

```
Wed Apr 27 19:07:21 UTC 2022 aarch64 aarch64 aarch64 GNU/Linux
```

Introduction to SLURM

How to run tasks online – use for debugging ONLY

```
$ srun --account=f202500002hpcvlabistula --partition=dev-arm  
--nodes=2 --ntasks=4 hostname
```

```
srun: job 450380 queued and waiting for resources
```

```
srun: job 450380 has been allocated resources
```

```
cna0002.deucalion.macc.fccn.pt
```

```
cna0002.deucalion.macc.fccn.pt
```

```
cna0001.deucalion.macc.fccn.pt
```

```
cna0001.deucalion.macc.fccn.pt
```

Introduction to SLURM

How to run batch job – use for actual computing

First, define a batch file as a shell script, e.g.,

hello_world.sh:

```
#!/bin/bash

#SBATCH --job-name=hello_world
#SBATCH --account=f202500002hpcvlabistula
#SBATCH --nodes=2
#SBATCH --ntasks=4

srun hostname
```

Introduction to SLURM

How to run batch jobs – use for actual computing

```
$ sbatch --partition=dev-arm hello_world.sh
```

```
Submitted batch job 454239
```

```
$ cat slurm-454239.out
```

```
cna0004.deucalion.macc.fccn.pt
```

```
cna0004.deucalion.macc.fccn.pt
```

```
cna0003.deucalion.macc.fccn.pt
```

```
cna0003.deucalion.macc.fccn.pt
```

Introduction to SLURM

How to inspect the job queues:

```
$ squeue
```

```
      JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
449866 dev-a100- test_vlm leagabay  R    2:01:09      1 gnx501
449826 large-arm 0.6nm_tg spustova PD      0:00     512 (AssocGrpCPUMinutesLimit)
447306 large-arm cpa_3600 takis.pu PD      0:00      8 (AssocGrpCPUMinutesLimit)
316281 large-arm h3_hyper muhammad PD      0:00     14 (AssocGrpCPUMinutesLimit)
449926 large-arm      test bernardo  R    15:14     32 cna[0183-0214]
448826 large-arm KG_Sigma jpeessoa R   16:00:23      1 cna0021
448391 large-arm 0.6nm_tg spustova R 1-22:45:05     96
cna[0132-0182,0277-0312,0648-0656]
448388 large-arm 0.6nm_tg spustova R 1-22:50:32     96 cna[0552-0647]
448387 large-arm 0.6nm_tg spustova R 1-23:04:02     96 cna[0456-0551]
448386 large-arm 0.6nm_tg spustova R 1-23:12:56     96 cna[0036-0131]
[...]
```

TIP: use “`squeue --user=$USER`” to examine your queue jobs; use “`watch squeue --user=$USER`” for a periodic update.

Introduction to SLURM

Summary of commands:

- `sinfo` – show information about partitions
- `salloc` – allocate resources for interactive session
- `srun` – run a command online
- `sbatch` – submit a batch job
- `squeue` – show the job queue

Other useful commands:

- `scancel` – cancel a job in the queue
- `sview` – graphical user interface to inspect SLURM state (requires X11)

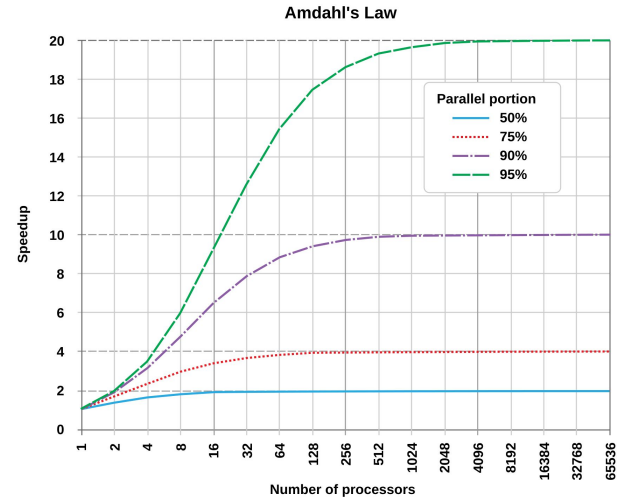
Introduction to MPI

Introduction to MPI

- High Performance Computing (HPC) scales up with the number of CPU cores
- Performance plateaus with the number of cores due to memory bus bottleneck
- The natural way to scale up is increasing the number of computers

MPI – Message Passing Interface

- portable message-passing protocol
- highly scalable and flexible
- open source implementations (e.g., OpenMPI)



Introduction to MPI

Concepts and terminology:

- **process** – running program (might be multi-threaded)
- **process group** – group of processes communicating among them
- **rank** – identifies processes in a group, from 0 to $N-1$ (where N is the **size**)
- **communicator** – mechanism providing communication among a process group
- **collective function** – communication among a process group, for spreading, collecting, and processing collected results

Introduction to MPI

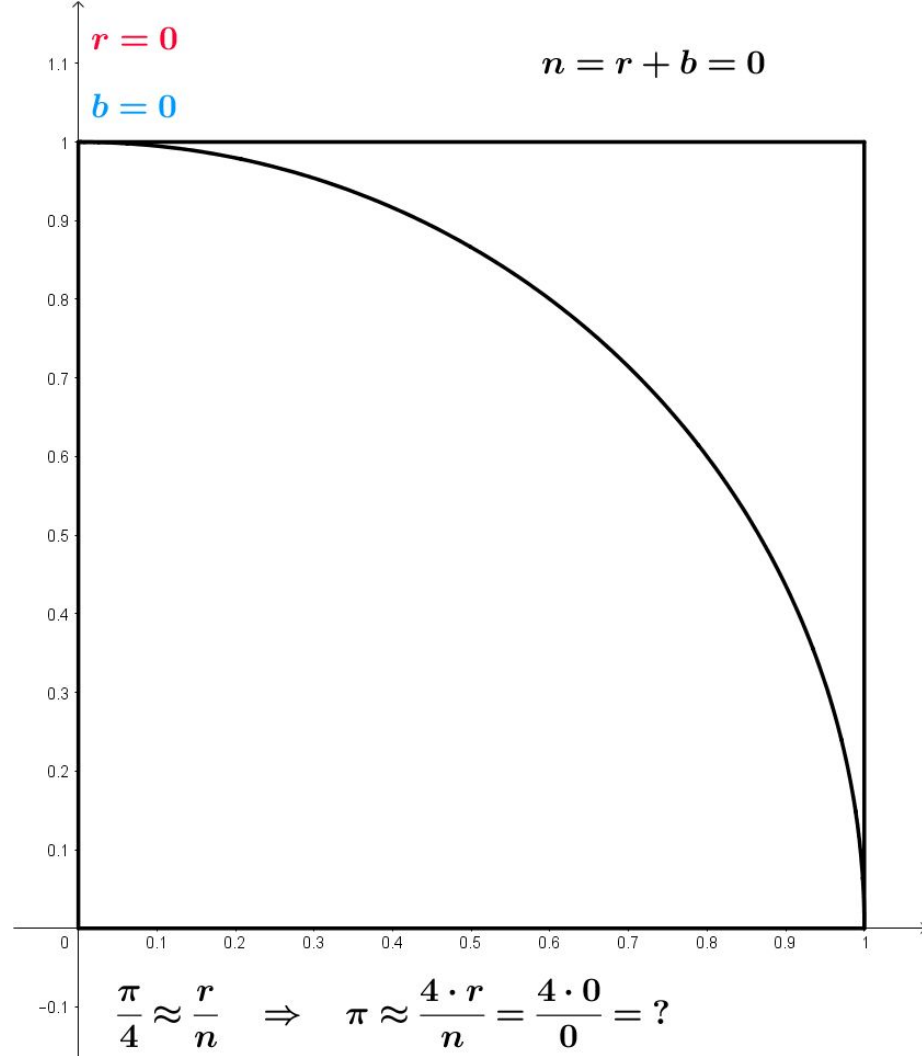
Working example:

- approximate π using Monte Carlo
- parallelize generation of points
- at the end collect counts and return approximation of π

Demo github repository here:



URL: <https://github.com/rventura/demo-hpc>



Introduction to MPI

- import Python packages (**mpi4py** is an interface for MPI)
- **comm** is the default communicator (world)
- get **rank** of process and group **size**

```
import numpy as np
from mpi4py import MPI

# Number of points per process
N = 10**8

# Setup OpenMPI
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
```

Introduction to MPI

- rank=0 is usually the central process (designated **root**)
- **Wtime()** is MPI's portable way of getting the wall time
- **bcast()** broadcasts a value from root to all processes in the group (collective f.)

```
if rank==0:
    # Master process
    t0 = MPI.Wtime()
    base = int(t0)
else:
    # Slave process
    base = None

# Ensure different seeds per process
seed = rank + comm.bcast(base, root=0)
np.random.seed(seed)

print(f"I'm instance {rank} in {size} and my seed is {seed}.")
```

Introduction to MPI

- Monte Carlo run, performed by all processes in the group
- divides run in chunks for performance, exploiting numpy arrays

```
# Main computation
chunk = 10**6
n_inside = 0
for nc in (N//chunk) * [chunk] + [N%chunk]:
    # Dividing the computation in chunks
    points = np.random.uniform(0, 1, (nc,2))
    inside = (np.sum(points**2, axis=1) <= 1)
    n_inside += np.sum(inside)
```

Introduction to MPI

- collects and sums count from all processes (collective f.)
- root process computes π estimation, error, and computing time

```
# Collect results by summing all n_inside
total_inside = comm.reduce(n_inside, op=MPI.SUM, root=0)

if rank==0:
    # Master process
    t = MPI.Wtime() - t0
    result = 4 * total_inside / (N*size)
    error = np.abs(result - np.pi)
    print(f"My approximation of  $\pi$  is {result} (error is {error}).\nIt took
          {t} seconds.")
```

Introduction to MPI

A selection of point-to-point communication methods:

- `send(obj, dest, tag=0)` – send *obj* to *dest* rank with *tag*
- `isend()` – non-blocking version of `send()`, returns a *request*
- `recv(buf=None, source=ANY_SOURCE, tag=ANY_TAG, status=None)` – blocks until a message is received
- `irecv()` – non-blocking version of `recv()`, returns a *request*
- `request.wait()` – wait for completion of request

Note: mpi4py methods handle arbitrary Python objects by “pickle”ing them; capitalized versions (e.g., `Send()`) handle raw MPI data types

Introduction to MPI

A selection of collective communication methods:

- `bcast(obj, root=0)` – broadcasts *obj* from *root* to all others
- `scatter(sendobj, root=0)` – distributes a *sendobj* list from *root* to all others
- `gather(sendobj, root=0)` – collects all *sendobj*'s into a list at *root*
- `reduce(sendobj, op=SUM, root=0)` – collects all *sendobj*'s at *root* and applies *op* to them all

Note: operation *op* can be predefined (MIN, MAX, SUM, PROD, ...) or even a Python function that reduces two elements into one

Introduction to MPI

More about MPI:



<https://mpi4py.readthedocs.io/>

The screenshot shows the documentation page for MPI for Python. The page has a blue header with the title 'MPI for Python' and a version dropdown set to '4.0.3'. Below the header is a search bar labeled 'Search docs'. On the left side, there is a 'CONTENTS' menu with the following items: Introduction, Overview, Tutorial, mpi4py, mpi4py.MPI, mpi4py.typing, mpi4py.futures, mpi4py.util, and mpi4py.run. The main content area on the right includes the title 'MPI for Python', author 'Lisandro Dalcin', contact 'dalcin@gmail.com', and date 'Feb 13, 2025'. There is an 'Abstract' section with the text: 'MPI for Python provides Python bindings for the Message Passing Interface (MPI) standard, allowing Python applications to exploit multiple processors on workstations, clusters and supercomputers.' Below the abstract is a paragraph: 'This package builds on the MPI specification and provides an object oriented interface resembling the MPI-2 C++ bindings. It supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communication of any picklable Python object, as well as efficient communication of Python objects exposing the Python buffer interface (e.g. NumPy arrays and builtin bytes/array/memoryview objects).' At the bottom right, there is a 'Contents' section with a dropdown menu showing 'Introduction', 'What is MPI?', and 'What is Buffer?'. A version dropdown at the bottom right shows '4.0.3'.

Q & A