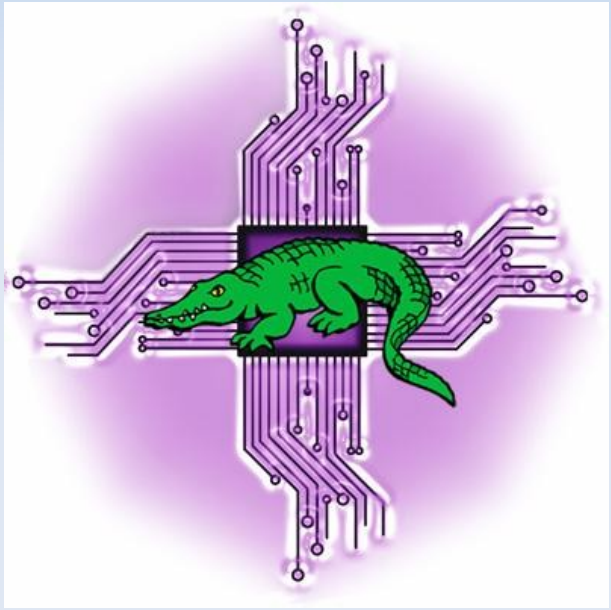
# Parallel Programming with MPI4py

## COMP90024 Cluster and Cloud Computing



## University of Melbourne, March 14 2024

lev.lafayette@unimelb.edu.au

# Outline of Lecture

This lecture will include:

An introduction to message passing, the MPI standard, and how mpi4py came about.

Creating a communications world, point-to-pint communication.

Broadcast communications, scatter and gather.

Throughtout, MPI4PY with environment modules and  Slurm on Spartan

# Message Passing Interface

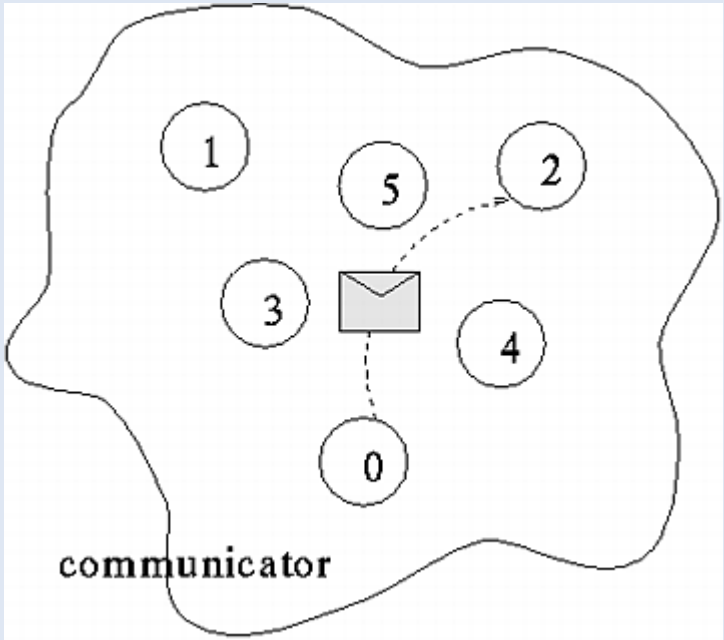
Message Passing involves setting up a communications world of different processing agents that can operate independently running subprograms and which can communicate with each other. With multicore and multinode systems this allows a program to scale.

Do not confuse with multithreading (OpenMP standard)!

Initial meeting for an MPI standard began in 1991, and a draft of v1 of the standard was introduced at Supercomputing ‘93, final version in 1994.

Initially designed for ANSI C and Fortran 77. Extended in v2 (1997) to include Fortran 90 and C++ . In v3 (2009) dropped C++ and added Fortran 2008. V4 (2021) improved fault tolerance, includes sessions, partititioned communications etc

Implemention of standard available as OpenMPI, MPICH. Language bindings available in Java, Julia, Python, R., et al.



# Python Performance

Python is a sequential programming language and has a lot of issues when it comes to performance.

For a computationally-intensive problems (a square matrix multiplication), Java is 8.8x faster than Python, CLang is 18x faster. Difference between an interpreted language versus a byte-code with JiT compilation, versus compilation to machine-code.

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{pmatrix}$$

Similar results when comparing overall energy efficiency; runtime, memory usage, energy consumption (CPU, DRAM). Compared against 27 programming languages, Python came last. Compiled languages (C, C++, Fortran, Rust, Pascal) were best. Interpreted languages like Python, Perl, Lua, Ruby were the worst.

# mpi4py

Despite these perfromance issues, Python is a very good language for developing programs where performance is less of an issue, and for working algorithms where it is an issue.

It is in this context that MPI4PY is a useful extension. It implements the most important outines from the MPI-2 standard in a simplified manner which provide working programs that can be converted to C or Fortran for performance.

Early releases (0.2.0) date back to May 2005; most recent release is Oct 4, 2023 (3.1.5).

Invoked on Spartan within SciPy-bundle/2022.05 (includes mpi4py-3.1.3) or as an independent module mpi4py/3.1.4.

Both require the OpenMPI/GCC toolchain foss/2022a to be loaded along with Python/3.10.4

Examples are in: /apps/examples/Python/MPI

# Interactive Point-to-Point mpi4py

Testing mpi4py examples should be done as an interactive job e.g.,

```
$ sinteractive --partition=sapphire --ntasks=4 --time=01:00:00
..
$ module load foss/2022a Python/3.10.4 mpi4py/3.1.4
$ cat helloworld.py
..
$ srun -n 4 python3 helloworld.py
..
$ cat mpi4py_test.py
..
$ srun -n 4 python3 mpi4py_test.py
..
$ module load SciPy-bundle/2022.05
$ less sendrecv.py
..
$ srun -n 4 python3 sendrecv.py
..

$ less taskpull.py
..
$ srun -n 4 python3 taskpull.py
..
```

# Batch Point-to-Point mpi4py

These are all point-to-point examples can also be run in batch mode with the Slurm workload manager.

$ less 2022hellompi.slurm
..
$ sbatch 2022hellompi.slurm
..
$ less 2022testmpi.slurm
..
$ sbatch 2022testmpi.slurm
..
$ less 2022sendrecv.slurm
..
$ sbatch 2022sendrecv.slurm
$ less 2022task-pull.slurm
..
$ sbatch 2022task-pull.slurm

As usual, the job submission status can be checked with
$ squeue -u [$user] .. and the output files checked for results (e.g., less slurm-56966395.out)

# Interactive Broadcast mpi4py

Testing broadcast mpi4py examples should be done as an interactive job e.g.,

```
$ sinteractive --partition=sapphire ––ntasks=4 --time=01:00:00
..
$ module load foss/2022a Python/3.10.4 mpi4py/3.1.4
$ cat broadcast.py
..
$ srun -n 4 python3 broadcast.py
..
$ module load SciPy-bundle/2022.05
$ less scatter-gather.py
..
$ srun -n 4 python3 scatter-gather.py
..
$ less reduce.py
..
$ srun -n 4 python3 reduce.py
..
```

These are examples of broadcast communication; root rank to others in the communications world in a single command, as opposed to point-to-point communication. Slurm versions are also available.


THANKS FOR WATCHING
& LISTENING PATIENTLY
makeameme.org