MPI4PY on Compute Canada Clusters

Harold Hodgins, MSc
PhD candidate in Biology

University of Waterloo

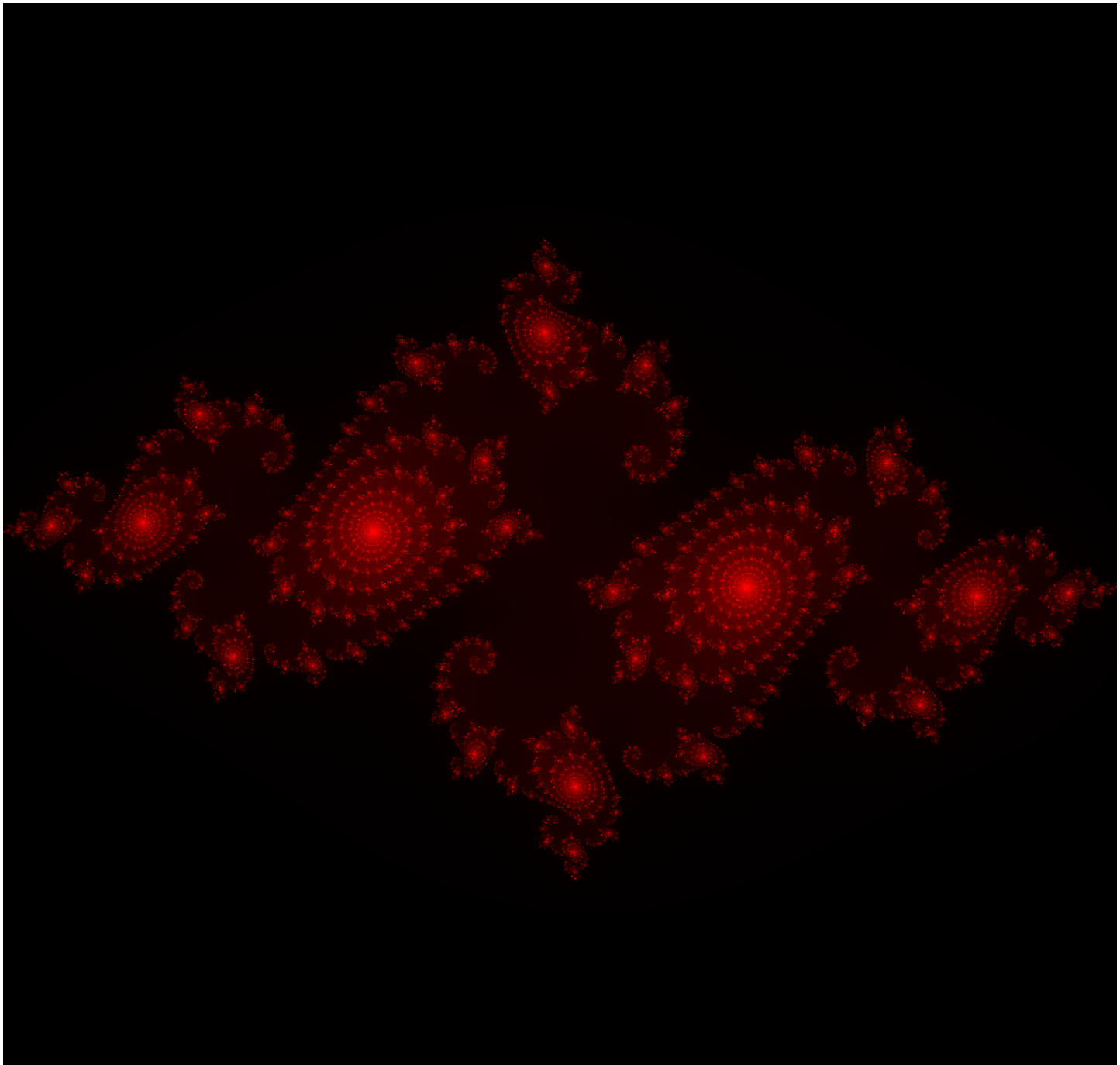
2023-02-13

Outline I

Who Am I

- BSc in Computer Science from Wilfirid Laurier University
  - Majors in Computer Electronics Applied Mathematics
  - Minor in Biochemistry
- MSc in Biology from University of Waterloo
  - Petabase-scale Data Mining Identifies Novel Clostridial Species and Neurotoxins Associated with Ancient Human DNA
- Experience programming in several languages
  - Python, Bash, C/C++, Java/Groovey, PIC Assembly, MATLAB/R
- Over ten years of experience using various HPC systems
  - Carbon Compute Cluster : Argonne National Labs
  - Sharcnet : University of Waterloo, University of Toronto
  - Compute Canada (now the Digital Research Alliance of Canada) : BC, Ontario, Quebec
- Thousands of compute hours running
  - Black white boxes
- custom scripts / pipelines / visualizations
- Run Linux as my daily OS
- Used Python with MPI4PY for my CP431 projects with Julia sets for my final project

Julia Sets



Why Python

- Designed from the start for better code readability

Why Python

- Designed from the start for better code readability
- Allows expression of concepts in fewer lines of code

Why Python

- Designed from the start for better code readability
- Allows expression of concepts in fewer lines of code
- Has dynamic type system, variables do not have to be declared

Why Python

- Designed from the start for better code readability
- Allows expression of concepts in fewer lines of code
- Has dynamic type system, variables do not have to be declared
- Has automatic memory management

Why Python

- Designed from the start for better code readability
- Allows expression of concepts in fewer lines of code
- Has dynamic type system, variables do not have to be declared
- Has automatic memory management
- Has large number of easily accessible, extensive libraries (eg. NumPy, SciPy)

Why Python

- Designed from the start for better code readability
- Allows expression of concepts in fewer lines of code
- Has dynamic type system, variables do not have to be declared
- Has automatic memory management
- Has large number of easily accessible, extensive libraries (eg. NumPy, SciPy)

Why Python

- Designed from the start for better code readability
- Allows expression of concepts in fewer lines of code
- Has dynamic type system, variables do not have to be declared
- Has automatic memory management
- Has large number of easily accessible, extensive libraries (eg. NumPy, SciPy)

All this can make developing new code “easier” and faster.

Why Not Python

- Generally **slower** then compiled languages and **less memory efficient**

Why Not Python

- Generally **slower** then compiled languages and **less memory efficient**
- ~~Only recently used in high performance computing environments. Ie fewer tutorials available.~~

Why Not Python

- Generally **slower** then compiled languages and **less memory efficient**
- ~~Only recently used in high performance computing environments. Ie fewer tutorials available.~~

Why Not Python

- Generally **slower** then compiled languages and **less memory efficient**
- ~~Only recently used in high performance computing environments. Ie fewer tutorials available.~~

You should use the right tool for the job. Sometimes that's Python and sometimes that's C/C++ or even Assembly.

How to Speed up Python

- Multi-threading/proccessing libraries

How to Speed up Python

- Multi-threading/proccessing libraries
- NumPy & Scipy

How to Speed up Python

- Multi-threading/proccessing libraries
- NumPy & Scipy
- Cython

How to Speed up Python

- Multi-threading/proccessing libraries
- NumPy & Scipy
- Cython
- Numba

How to Speed up Python

- Multi-threading/proccessing libraries
- NumPy & Scipy
- Cython
- Numba
- MPI4PY

How to Speed up Python

- Multi-threading/proccessing libraries
- NumPy & Scipy
- Cython
- Numba
- MPI4PY

How to Speed up Python

- Multi-threading/proccessing libraries
- NumPy & Scipy
- Cython
- Numba
- MPI4PY

See Dr. Pawel Pomorski Slides Numpy, Cython, Multiprocessing and Numba.

So Why Not use Multi-threading or Multi-processing

- Your professor wants you to learn how to use MPI :)

So Why Not use Multi-threading or Multi-processing

- Your professor wants you to learn how to use MPI :)
- Multi-threading/processing doesn't scale beyond one computer

Why MPI4PY

- MPI4Py provides an interface very similar to the MPI-2 standard C++ Interface

Why MPI4PY

- MPI4Py provides an interface very similar to the MPI-2 standard C++ Interface
- Focus is in translating MPI syntax and semantics: If you know MPI, MPI4Py is "obvious"

Why MPI4PY

- MPI4Py provides an interface very similar to the MPI-2 standard C++ Interface
- Focus is in translating MPI syntax and semantics: If you know MPI, MPI4Py is "obvious"
- You can communicate Python objects!!

Why MPI4PY

- MPI4Py provides an interface very similar to the MPI-2 standard C++ Interface
- Focus is in translating MPI syntax and semantics: If you know MPI, MPI4Py is "obvious"
- You can communicate Python objects!!

Why MPI4PY

- MPI4Py provides an interface very similar to the MPI-2 standard C++ Interface
- Focus is in translating MPI syntax and semantics: If you know MPI, MPI4Py is "obvious"
- You can communicate Python objects!!

What you lose in performance, you gain in shorter development time, and potentially fewer lost neurons.

Caveats

- There are hundreds of functions in the MPI standard, not all of them are necessarily available in MPI4Py, most commonly used are

Caveats

- There are hundreds of functions in the MPI standard, not all of them are necessarily available in MPI4Py, most commonly used are
- No need to call MPI_Init() or MP_Finalize()

Caveats

- There are hundreds of functions in the MPI standard, not all of them are necessarily available in MPI4Py, most commonly used are
- No need to call MPI_Init() or MP_Finalize()
  - MPI_Init() is called when you import the module

Caveats

- There are hundreds of functions in the MPI standard, not all of them are necessarily available in MPI4Py, most commonly used are
- No need to call MPI_Init() or MP_Finalize()
  - MPI_Init() is called when you import the module
  - MPI_Finalize() is called before the Python process ends

Hello to the World

```
#!/usr/bin/env python3
"""
Parallel Hello World
"""
from mpi4py import MPI

size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()

print('Greetings. I am process {} of {} on {}'.format
    ↪ (rank, size, name ) )
```

MPI4PY Communications

- COMM WORLD is the collection of all processes

MPI4PY Communications

- COMM WORLD is the collection of all processes
- To get size: MPI.COMM_WORLD.Get_size() or MPI.COMM_WORLD.size

MPI4PY Communications

- COMM WORLD is the collection of all processes
- To get size: MPI.COMM_WORLD.Get_size() or MPI.COMM_WORLD.size
- To get rank: MPI.COMM_WORLD.Get_rank() or MPI.COMM_WORLD.rank

MPI4PY Communications

- COMM WORLD is the collection of all processes
- To get size: MPI.COMM_WORLD.Get_size() or MPI.COMM_WORLD.size
- To get rank: MPI.COMM_WORLD.Get_rank() or MPI.COMM_WORLD.rank

MPI4PY Communications

- COMM WORLD is the collection of all processes
- To get size: MPI.COMM_WORLD.Get_size() or MPI.COMM_WORLD.size
- To get rank: MPI.COMM_WORLD.Get_rank() or MPI.COMM_WORLD.rank

See Texas tutorials for more options

Point to Point Communication

- Send a message from one process to another

Point to Point Communication

- Send a message from one process to another
- Messages can contain any number of native or user defined types with an associated message tag

Point to Point Communication

- Send a message from one process to another
- Messages can contain any number of native or user defined types with an associated message tag
- MPI4Py handles the packing and unpacking for user defined data types

Point to Point Communication

- Send a message from one process to another
- Messages can contain any number of native or user defined types with an associated message tag
- MPI4Py handles the packing and unpacking for user defined data types
- Two types of communication: Blocking and non - Blocking

Collective Communications

Used to send messages to multiple processes at once. Eg. Broadcast, Scatter, Gather, Reduction

See Texas tutorials for examples and specifics

Transferring Python Data

```
#!/usr/bin/env python3
"""
Send Python Data
"""

from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0 :
    data = { 'a': 7 , 'b' : 3.14 }
    comm.send( data , dest=1, tag=11)
    print( 'Message set, data is : ', data )
elif rank == 1 :
    data = comm.recv( source=0, tag=11)
    print( 'Message Received, data is : ', data )
```

Transferring Numpy Data I

```
#!/usr/bin/env python3
"""
Send Numpy Data
"""

from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

# pass explicit MPI data types
if rank == 0 :
    data = numpy.random.randint(0,100, size=(2, 4),
        ↪ dtype='i')
    comm.Send( [ data, MPI.INT ] , dest=1, tag=77)
```

Transferring Numpy Data II

```
elif rank == 1 :
    data = numpy.empty ( (2,4), dtype='i' )
    comm.Recv( [ data, MPI.INT ], source=0 , tag=77)
    print(data)

# automatic MPI data type discovery
if rank == 0 :
    data = numpy.random.randint(0,100, size=(2, 3, 4)
        ↪ , dtype='i')
    comm.Send( data, dest=1, tag=13)
elif rank == 1 :
    data = numpy.empty( (2,3,4), dtype='i' )
    comm.Recv( data, source=0, tag=13)
    print(data)
```

Communication Modes

- Use nonblocking communication to overlap communication with computation

Communication Modes

- Use nonblocking communication to overlap communication with computation
- Isend() Irecv() return immediately. Their buffers are **NOT SAFE** for reuse

Communication Modes

- Use nonblocking communication to overlap communication with computation
- Isend() Irecv() return immediately. Their buffers are **NOT SAFE** for reuse
- Only isend() is implemented for python objects

Communication Modes

- Use nonblocking communication to overlap communication with computation
- Isend() Irecv() return immediately. Their buffers are **NOT SAFE** for reuse
- Only isend() is implemented for python objects
- Use Test() or Wait() to check if the communication has finished

Communication Modes

- Use nonblocking communication to overlap communication with computation
- Isend() Irecv() return immediately. Their buffers are **NOT SAFE** for reuse
- Only isend() is implemented for python objects
- Use Test() or Wait() to check if the communication has finished
- Use Cancel() to cancel the communication

Communication Modes

- Use nonblocking communication to overlap communication with computation
- Isend() Irecv() return immediately. Their buffers are **NOT SAFE** for reuse
- Only isend() is implemented for python objects
- Use Test() or Wait() to check if the communication has finished
- Use Cancel() to cancel the communication
- Use comm.Iprobe(source=target, tag=11) to check for incoming if you wanted to use irecv

Compute Canada Clusters

- Documentation at
  `https://docs.computecanada.ca/wiki/Graham`
- Python documentation at
  `https://docs.computecanada.ca/wiki/Python`
- MPI documentation at
  `https://docs.computecanada.ca/wiki/MPI`

Submitting Jobs

- Load correct python module and mpi4py (module load python/3.8 mpi4py)

Submitting Jobs

- Load correct python module and mpi4py (module load python/3.8 mpi4py)
- Use virtual environment for code with external libraries

Submitting Jobs

- Load correct python module and mpi4py (module load python/3.8 mpi4py)
- Use virtual environment for code with external libraries
- Create run_job.sh

Submitting Jobs

- Load correct python module and mpi4py (module load python/3.8 mpi4py)
- Use virtual environment for code with external libraries
- Create run_job.sh
- Sbatch run_job.sh

Submitting Jobs

- Load correct python module and mpi4py (module load python/3.8 mpi4py)
- Use virtual environment for code with external libraries
- Create run_job.sh
- Sbatch run_job.sh
- Use salloc –time=1:0:0 –ntasks=4 –nodes=2 –account=def-someuser for interactive jobs

Submitting Jobs

- Load correct python module and mpi4py (module load python/3.8 mpi4py)
- Use virtual environment for code with external libraries
- Create run_job.sh
- Sbatch run_job.sh
- Use salloc –time=1:0:0 –ntasks=4 –nodes=2 –account=def-someuser for interactive jobs

Submitting Jobs

- Load correct python module and mpi4py (module load python/3.8 mpi4py)
- Use virtual environment for code with external libraries
- Create run_job.sh
- Sbatch run_job.sh
- Use salloc –time=1:0:0 –ntasks=4 –nodes=2 –account=def-someuser for interactive jobs

```
#!/bin/bash
#SBATCH --account=def-some-user
#SBATCH --nodes=2
#SBATCH --ntasks=8 # number of MPI processes
#SBATCH --mem-per-cpu=512M # memory; default unit is
    ↪ megabytes
#SBATCH --time=0-00:15 # time (DD-HH:MM)

module load StdEnv/2020 intel/2020.1.217 openmpi
    ↪ /4.0.3 scipy-stack/2022a mpi4py/3.1.2 python
    ↪ /3.9
#export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK #Use
    ↪ this with multi-threaded code
echo 'Hello World Example'
srun ./hello_world.py # mpirun or mpiexec also work
echo 'Send Python Data Example'
srun ./send_python_data.py
echo 'Send Numpy Data Example'
srun ./send_numpy_data.py
```

Live Demo

What every could go wrong :)

Summary

- start with from mpi4py import MPI

Summary

- start with from mpi4py import MPI
- comm = MPI.COMM_WORLD

Summary

- start with from mpi4py import MPI
- comm = MPI.COMM_WORLD
- comm.send() vs comm.Send()

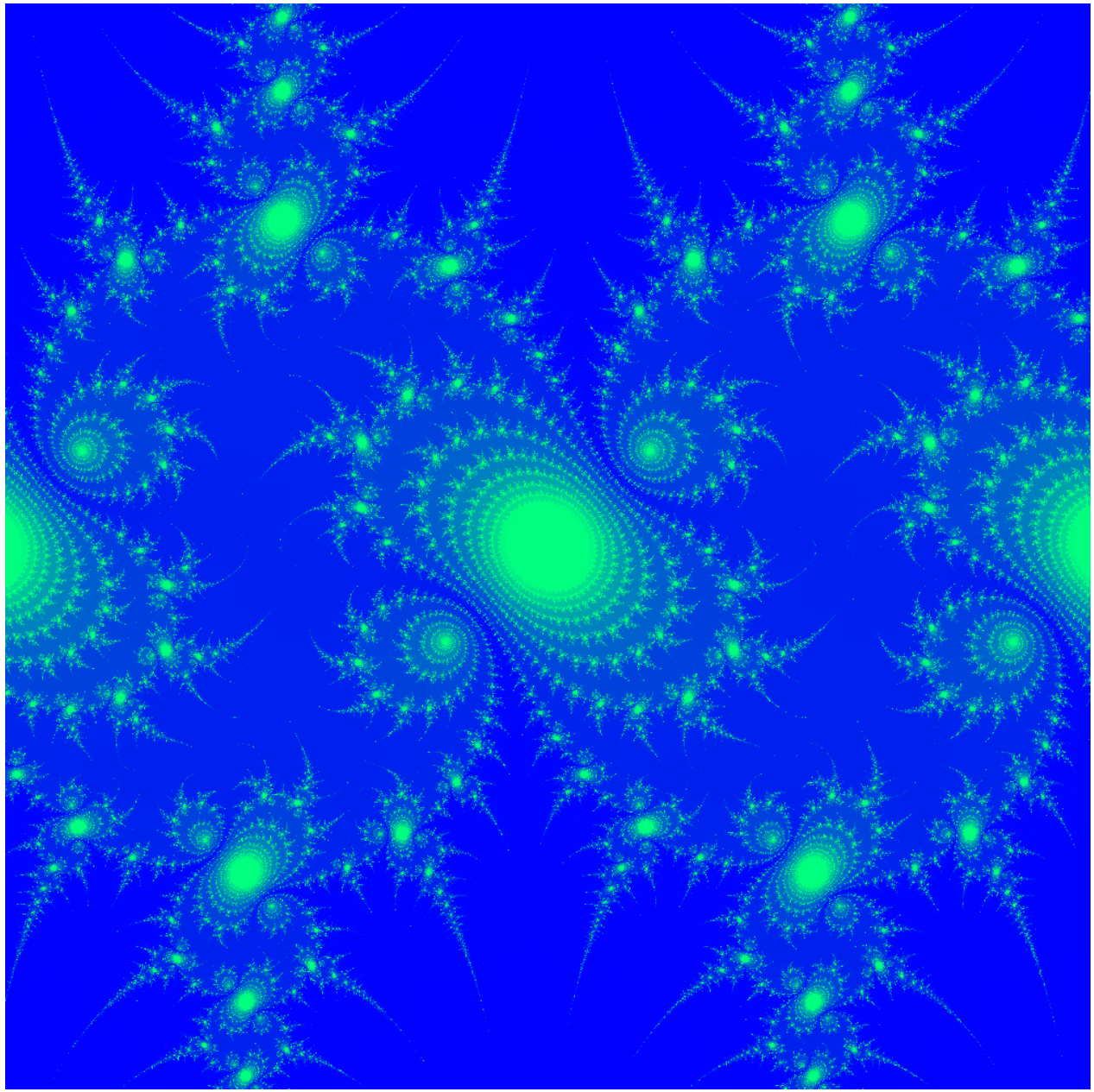
Summary

- start with from mpi4py import MPI
- comm = MPI.COMM_WORLD
- comm.send() vs comm.Send()
- module load

Summary

- start with from mpi4py import MPI
- comm = MPI.COMM_WORLD
- comm.send() vs comm.Send()
- module load
- sbatch run_job.sh or salloc

Questions



References


Pawel Pomorski : Python for high performance computing
https://helpwiki.sharcnet.ca/wiki/images/4/4c/Hpc_python_beamer.pdf


Pawel Pomorski : Speeding up Python code with Numba
https://helpwiki.sharcnet.ca/wiki/images/4/4e/Numba_webinar.pdf


Yaakoub El Khamra : Hpc python tutorial: Introduction to mpi4py
https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=be16db01-57d9-4422-b5d5-17625445f351&groupId=13601


Antonio Gomez-Iglesias : mpi4py hpc python
https://portal.tacc.utexas.edu/documents/13601/1102030/4_mpi4py.pdf/f43b984e-4043-44b3-8225-c3ce03ecb93b


Michael McGoodwin : Julia Jewels: An Exploration of Julia Sets
https://mcgoodwin.net/julia/juliajewels.html