Introduction to MPI with MPI4Py

Esteban Meneses, PhD
emeneses@pitt.edu

Kenneth P. Dietrich School of Arts and Sciences
Center for Simulation and Modeling (SaM)
THE FOLLOWING PRESENTATION HAS BEEN RATED

PG-18
PROGRAMMERS STRONGLY CAUTIONED

MOST CODES ARE NOT PROPERLY OPTIMIZED

PYTHON FANATICS MAY FIND SOME MATERIAL OFFENSIVE
Downloads

http://coco.sam.pitt.edu/~emeneses/teaching/mpi
Tools

- Python
  - Data management
- MPI4Py
  - Interface
- MPI
  - Communication

Source of images: http://www.wikipedia.org/
Why Python?

- Simple syntax, very expressive.
- Good mix of programming paradigms.
- Great for rapid prototyping.
- One of the 3 most popular languages in high performance computing (HPC); along with C and Fortran.
- Mature language (since 1991) with a huge user base and lots of extension libraries.
Why MPI?

- *De facto* standard language for parallel computing on HPC gear.
- Simple communication model between processes in a program.
- Multiple implementations; highly efficient for different platforms.
- Well established community (since 1994).
- Large ecosystem of tools and applications built on MPI.
Why MPI4Py?

- Well regarded implementation of MPI on Python among competing alternatives.
- Clean and efficient MPI interface for Python.
- Covers most of the MPI-2 standard, including dynamic process creation.
- Extensible and compatible implementation.
- Responsive technical support.
Contents

• Basics of Message Passing Interface (MPI)

• Smith-Waterman Algorithm

• Parallel Smith-Waterman in MPI
Basics of Message Passing Interface (MPI)
Distributed Memory Systems

- Each processor has its own private memory.
- A network connects all the processors.
Message-Passing Paradigm

• A parallel program is decomposed into processes, called **ranks**.
• Each rank holds a portion of the program’s data into its private memory.
• Communication among ranks is made explicit through messages.
• Channels honor first-in-first-out (FIFO) ordering.
Single-Program Multiple-Data (SPMD)

- All processes run the same program, each accesses a different portion of data.
- All processes are launched simultaneously.
- Communication:
  - Point-to-point messages.
  - Collective communication operations.
Features of Message Passing

• **Simplicity**: the basics of the paradigm are traditional communication operations.

• **Generality**: can be implemented on most parallel architectures.

• **Performance**: the implementation can match the underlying hardware.

• **Scalability**: the same program can be deployed on larger systems.
Message Passing Interface (MPI)

• Standard for operations in message passing.
• Led by MPI Forum (academia & industry).
• Implementations:
  – Open-source: MPICH, Open MPI.
  – Proprietary: Cray, IBM, Intel.
MPI Ranks

- Ranks have private memory.
- Each rank has a unique identification number.
- Ranks are numbered sequentially: \([0, n-1]\).
MPI Communicators

- Groups of ranks among which a rank can communicate.
- **COMM_WORLD** is a communicator including all ranks in the system.
MPI4Py Primer

• Importing library:
  from mpi4py import MPI

• Getting important information:
  comm = MPI.COMM_WORLD
  rank = MPI.COMM_WORLD.Get_rank()
  size = MPI.COMM_WORLD.Get_size()
  name = MPI.Get_processor_name()

• Executing in parallel:
  mpirun -np <P> python <code>.py
Exercise 1

- Write a parallel Python “Hello World”. Run the program with different number of ranks.
- Example:
  ```
  mpirun -np 4 python hello.py
  ```
- Output:
  Hello, world! This is rank 0 of 4 running on kolmakov.sam.pitt.edu
  Hello, world! This is rank 3 of 4 running on kolmakov.sam.pitt.edu
  Hello, world! This is rank 1 of 4 running on kolmakov.sam.pitt.edu
  Hello, world! This is rank 2 of 4 running on kolmakov.sam.pitt.edu
Point-to-point Operations

- Synchronous instructions to send a message from one source rank to a destination rank.

```python
comm.send(data, dest=j)
data = comm.recv(source=i)
```
Exercise 2

- Implement a parallel ping-pong in Python. A message carrying a counter is exchanged between the two ranks 1000 times. Rank 1 will increment the counter upon reception.

- Example:
  mpirun -np 2 python pingpong.py

- Output:
  Total number of message exchanges: 1000
Exercise 3

• Implement a parallel Python program that creates a ring of ranks. Each rank gets a random integer value [0,100]. The program computes in each rank the sum of all the values by circulating them around the ring.

• Example:
  
  ```bash
  mpirun -np 4 python ring.py
  ```

• Output:
  
  ```text
  [0] Total sum: 172
  [1] Total sum: 172
  ```
Collective Communication Operations

- Instructions to exchange data including all the ranks in a communicator.
- The root rank indicates the source or destination of the operation.
- **Broadcast**: one to many.
  \[
  \text{comm.bcast(data, root=0)}
  \]
- **Reduction**: many to one.
  \[
  \text{comm.reduce(data, op=\text{MPI.SUM}, root=0)}
  \]
Collectives

<table>
<thead>
<tr>
<th>Ranks</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Broadcast:

<table>
<thead>
<tr>
<th>Ranks</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Reduce:

\[ A = X \oplus Y \oplus Z \oplus V \]
Exercise 4

- Write a parallel Python program where each rank gets a random integer value \([0,100]\). The program gets in each rank the sum of all the values in the ranks using only broadcast and reduction.

- Example:
  ```
  mpirun -np 4 python collective.py
  ```

- Output:
  ```
  [0] Total sum: 220
  [1] Total sum: 220
  ```
Collectives (cont.)

<table>
<thead>
<tr>
<th>Ranks</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>A</td>
</tr>
<tr>
<td>Y</td>
<td>A</td>
</tr>
<tr>
<td>Z</td>
<td>A</td>
</tr>
<tr>
<td>V</td>
<td>A</td>
</tr>
</tbody>
</table>

**AllReduce**

\[ A = X \oplus Y \oplus Z \oplus V \]

<table>
<thead>
<tr>
<th>Ranks</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>C</td>
</tr>
<tr>
<td>Y</td>
<td>B</td>
</tr>
<tr>
<td>Z</td>
<td>A</td>
</tr>
</tbody>
</table>

**Scan**

\[ C = X \oplus Y \]

\[ B = X \oplus Y \oplus Z \]

\[ A = X \oplus Y \oplus Z \oplus V \]

<table>
<thead>
<tr>
<th>Ranks</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

**Scatter**

<table>
<thead>
<tr>
<th>Ranks</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

**Combine**

<table>
<thead>
<tr>
<th>Ranks</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>
Collectives (cont.)

Ranks

<table>
<thead>
<tr>
<th>Ranks</th>
<th>Data</th>
<th>Ranks</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
<td>V</td>
<td>V</td>
</tr>
</tbody>
</table>

AllGather

Ranks

<table>
<thead>
<tr>
<th>Ranks</th>
<th>Data</th>
<th>Ranks</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₀</td>
<td>X₀</td>
<td>X₁</td>
<td>X₁</td>
</tr>
<tr>
<td>Y₀</td>
<td>Y₀</td>
<td>Y₁</td>
<td>Y₁</td>
</tr>
<tr>
<td>Z₀</td>
<td>Z₀</td>
<td>Z₁</td>
<td>Z₁</td>
</tr>
<tr>
<td>V₀</td>
<td>V₀</td>
<td>V₁</td>
<td>V₁</td>
</tr>
</tbody>
</table>

AllToAll
Smith-Waterman Algorithm
Comparing Biological Sequences

- How similar are two sequences $X$ and $Y$?
  - Perform local alignment of $X$ and $Y$ to identify regions of similarity.

- Find the highest scoring alignment between subsequences of $X$ and $Y$.
  - Longest common subsequence (LCS).

- Example:
  $X=$CCTGAATATAGCAAG $Y=$TTGAGACCATGGGCAAATC
  $LCS=$TGAAATGCAA $score=10$
Smith-Waterman Algorithm

- **Optimal** algorithm: finds the LCS of two sequences $X$ and $Y$.
- Dynamic programming strategy.
  - Matrix stores intermediate values.
- Modification of Needleman-Wunsch algorithm.
- Simple and elegant.
Pseudocode

Smith-Waterman($X,Y$)
    for $i ← -1$ to $n-1$ do
        $L[i,-1] ← 0$
    for $i ← -1$ to $n-1$ do
        $L[i,-1] ← 0$
    for $i ← 0$ to $n-1$ do
        for $j ← 0$ to $m-1$ do
            if $X[i] = Y[j]$ then
                $L[i,j] ← L[i-1,j-1]+1$
            else
                $L[i,j] ← \max\{L[i-1,j],L[i,j-1]\}$
        return $L[n-1,m-1]$
Example

\[ X = \text{CCTGAATATAGCAAG} \quad Y = \text{TTGAGACCATGGCAAATC} \]

<p>| | | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
**Exercise 5**

- Run the `smith-waterman.py` program with different sequences.
  
  **Example:**
  
  ```python
  python smith-waterman x.txt y.txt
  ```

- Note the total execution time as the size of the input sequences increase.
Parallel Smith-Waterman in MPI
Problem Decomposition

• How to split data and work among ranks?

• Consider data dependencies.
Overlap of Computation and Communication

- Finer communication enhances parallelism, but increases communication costs.

- Challenge: find an optimum communication block.
Thread Coarsening

- Finer decomposition leads to abundant communication.
- Trade-off between synchronization and communication costs.
Speedup

- How faster the parallel version is compared to the sequential version.
- $T_1$: sequential time.
- $T_P$: time in parallel.

$$\text{Speedup} = \frac{T_1}{T_P}$$
Exercise 6

- Run the `parallel-sw.py` program with different sequences.
- Example:
  ```
  mpirun -np 8 python parallel-sw.py longX.txt longY.txt 4
  ```
- Run the program with different values for the block size.
- Run the program with different number of processors.
Exercise 7

• Choose a couple of input sequences and compute the speedup of the parallel version.

• Example:

  python smith-waterman.py longX.txt longY.txt
  Total time: 0.049111 seconds
  Size of longest common subsequence: 69

  mpirun -np 2 python parallel-sw.py longX.txt longY.txt 16
  Total time: 0.031008 seconds
  Size of longest common subsequence: 69

• Speedup = 0.0491/0.0310 = 1.5839
Concluding Remarks

• MPI consists of a simple communication interface: send, recv, bcast, reduce, and more.
• MPI lets you write scalable code, from your desktop all the way to petascale systems.
• Parallelizing codes is an art:
  – Computation-communication overlap.
  – Thread coarsening.
Acknowledgments

- Dr. Allan Orozco for coordinating the setup of the lab and the overall seminar.
- Dr. Lisandro Dalcin for his help on understanding the internals of MPI4Py.
Thank you!
Q&A

emeneses@pitt.edu
esteban.meneses@acm.org