Inter-node Communication Performance Tuning

Luke Hawranick

August 2, 2023

West Virginia University
High Performance Computing and Visualization Group
Applied and Computational Mathematics Division
Information Technology Laboratory

Introduction

Introduction

Parallel Programming

- Serial programs perform tasks in linear fashion.

Parallel Programming

- Serial programs perform tasks in linear fashion.

> **Process** - an independent sequence of execution, running in a memory space separate from processes on other processing nodes

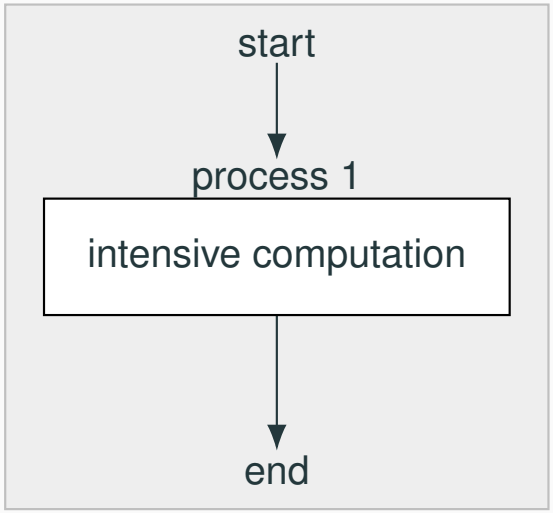


Figure 1: serial program design

Parallel Programming

- Serial programs perform tasks in linear fashion.

Process - an independent sequence of execution, running in a memory space separate from processes on other processing nodes

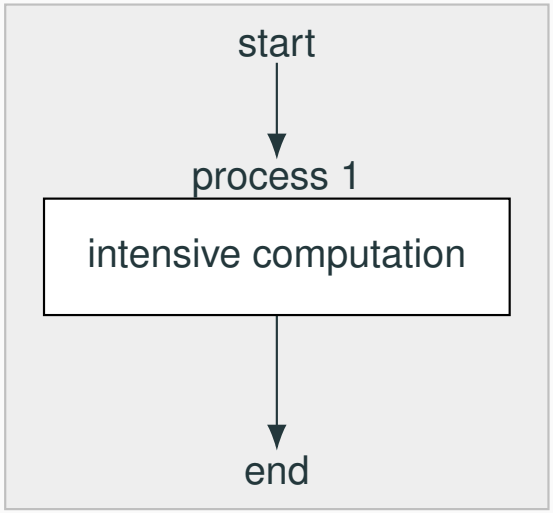


Figure 1: serial program design

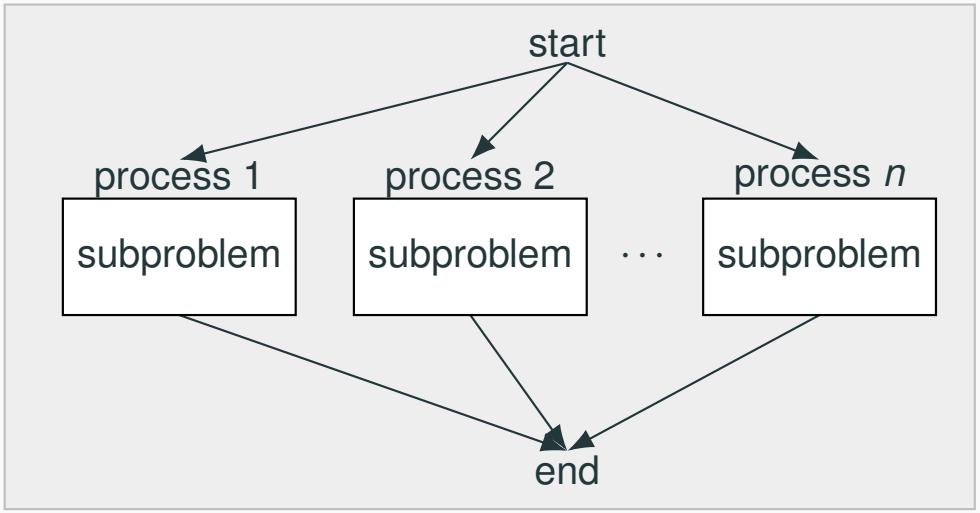


Figure 2: parallel program design

Introduction

Message Passing Interface (MPI) is a standardized library that enables communication and coordination among multiple processes.

- Because of the extremely scalable context of MPI, thorough testing of a specific application is important to have confidence in optimization of message passing parallel programs.

Simulator - High Level Overview

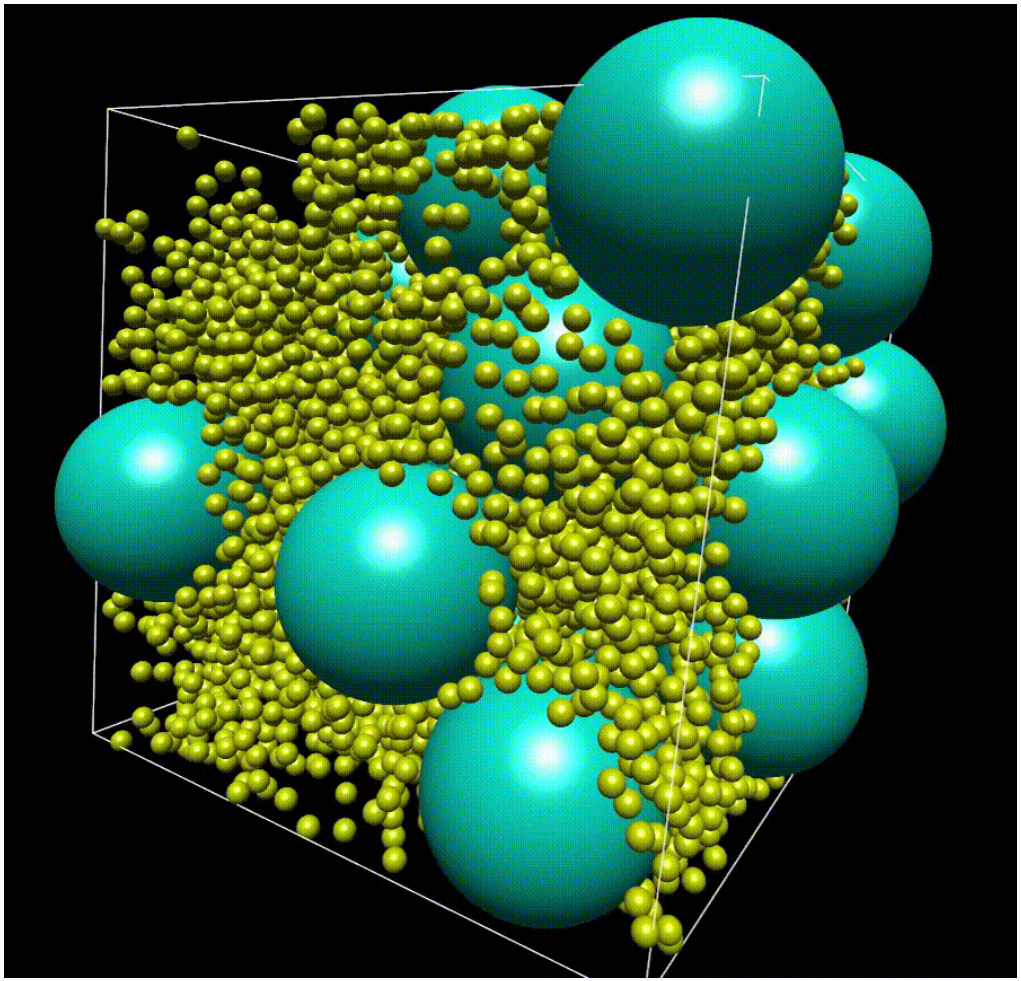
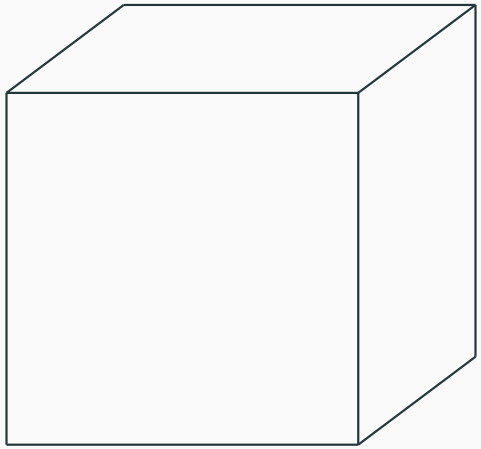


Figure 3: Quaternion Dissipative Particle Dynamics (QDPD) simulator output

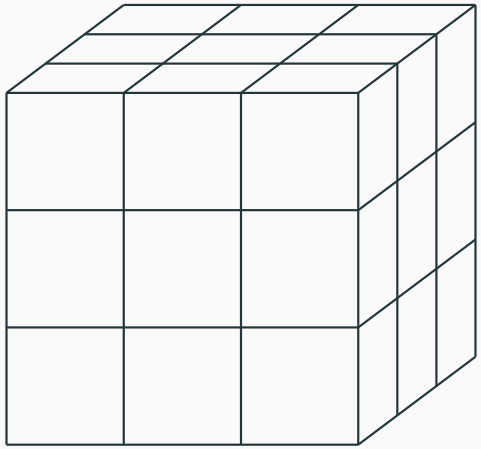
Objective

Simulator - High Level Overview



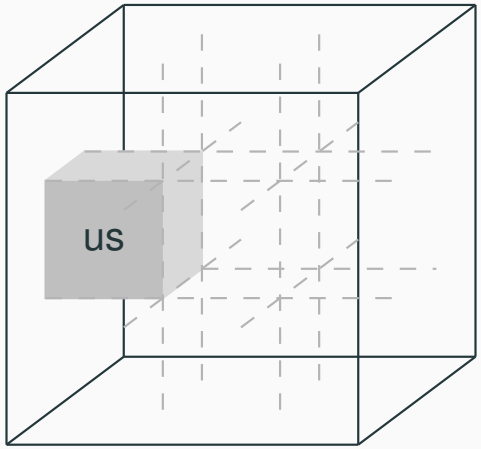
Objective

Simulator - High Level Overview



Objective

Simulator - High Level Overview

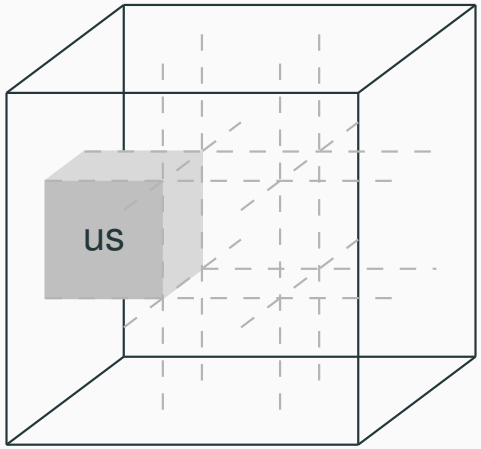


Process' responsibilities **for each time step**

- Update each particle's position in the next time step

Objective

Simulator - High Level Overview

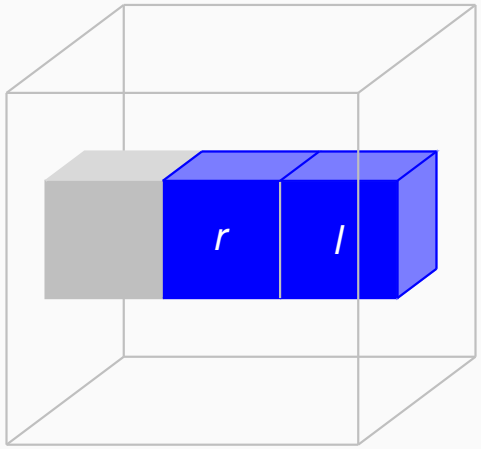


Process' responsibilities **for each time step**

- Update each particle's position in the next time step
- **COMMUNICATION:** Send particles that are calculated to move to another region.

Objective

Simulator - High Level Overview

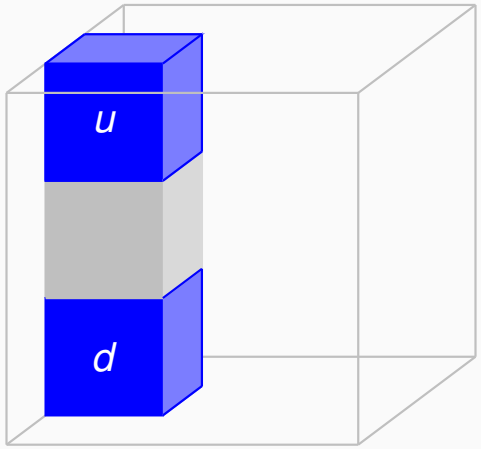


Process' responsibilities **for each time step**

- Update each particle's position in the next time step
- **COMMUNICATION:** Send particles that are calculated to move to another region.

Objective

Simulator - High Level Overview

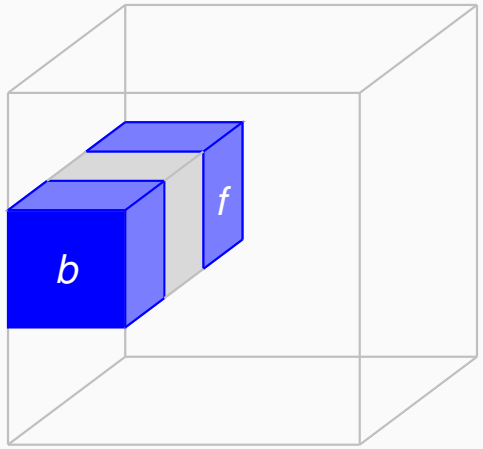


Process' responsibilities **for each time step**

- Update each particle's position in the next time step
- **COMMUNICATION:** Send particles that are calculated to move to another region.

Objective

Simulator - High Level Overview



Process' responsibilities **for each time step**

- Update each particle's position in the next time step
- **COMMUNICATION:** Send particles that are calculated to move to another region.

Objective

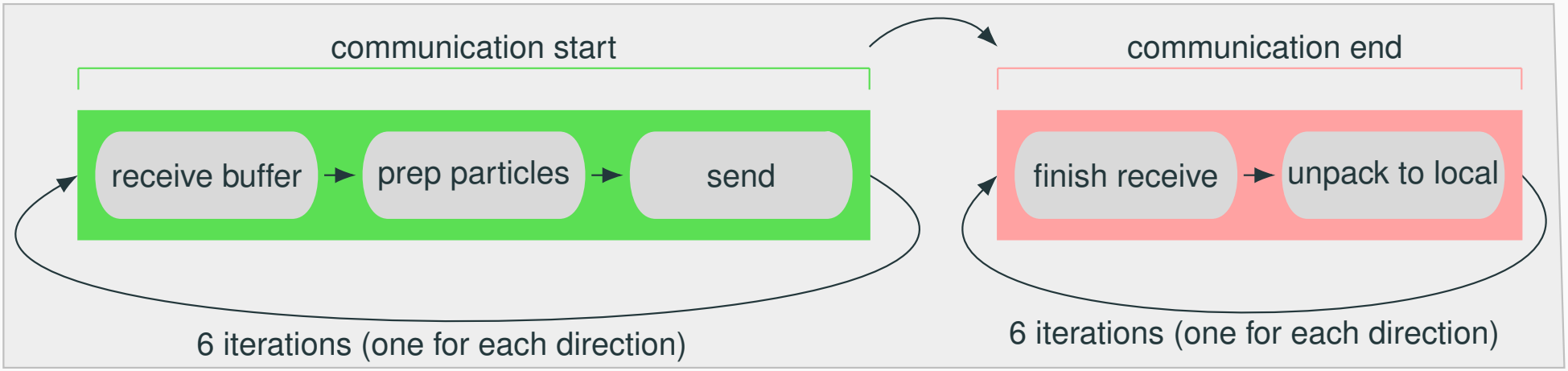


Figure 3: Each process' communication routine **per time step**

Objective

Possible Improvements

- 2 ways in question to prepare data to send between processes
- 2 MPI library implementations that we have access to: **Intel OneAPI** and **OpenMPI**
- Is it possible to perform computations while communicating, to improve efficiency?

Objective

Possible Improvements

- 2 ways in question to prepare data to send between processes
- 2 MPI library implementations that we have access to: **Intel OneAPI** and **OpenMPI**
- Is it possible to perform computations while communicating, to improve efficiency?

Why Bother? It Runs.

- Optimizing efficiency is extremely important in HPC
- The program runs for days, so a small consistent improvement in message communicating can bring results about drastically sooner.

Objective

Possible Improvements

- 2 ways in question to prepare data to send between processes
- 2 MPI library implementations that we have access to: **Intel OneAPI** and **OpenMPI**
- Is it possible to perform computations while communicating, to improve efficiency?

Why Bother? It Runs.

- Optimizing efficiency is extremely important in HPC
- The program runs for days, so a small consistent improvement in message communicating can bring results about drastically sooner.

Questions:

- How can we optimize communication?
- Can we show overlap of computation and communication?

Two Methods of Preparing Data

Particle Structure

```c
typedef struct particle {
    int ints[50];
    double reals[100];
    struct particle* next;
} particle;
```

Figure 4: C Struct Representation of a Particle

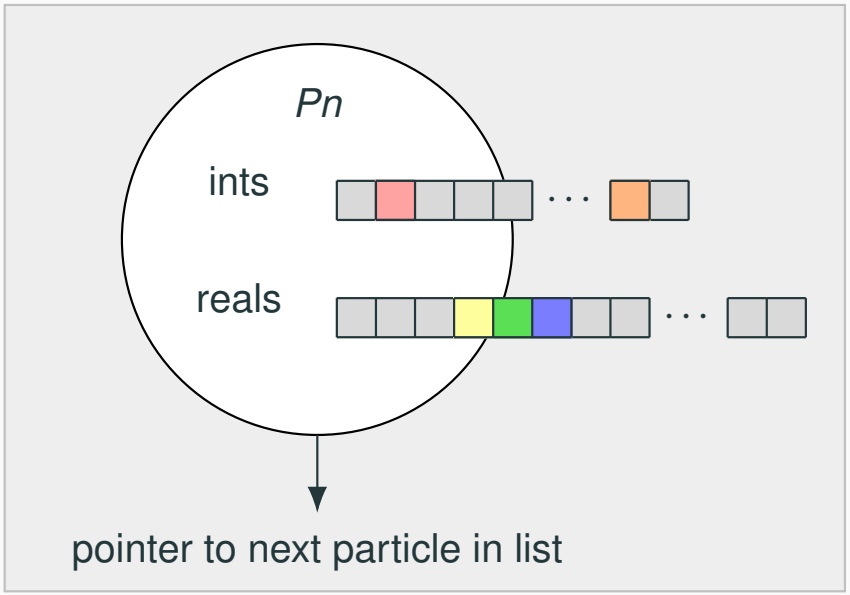
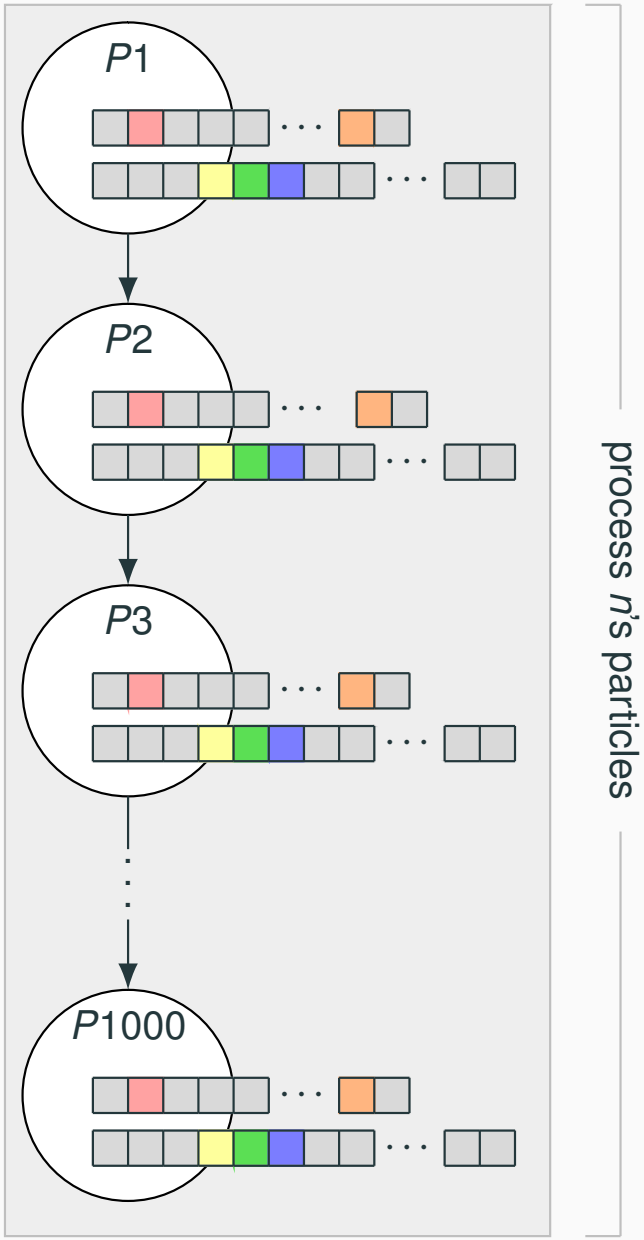


Figure 5: Graphical Representation of a Particle

ONLY data from the particle to send

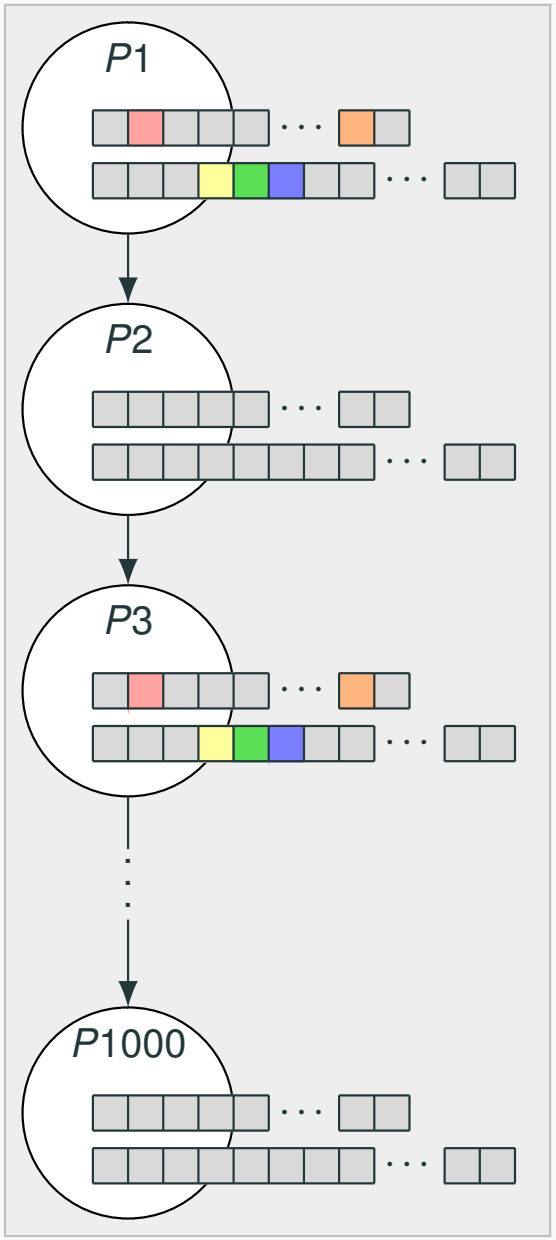
Method 1 - packing a buffer



Particles to send: $[1, 3]$

SEND BUFFER

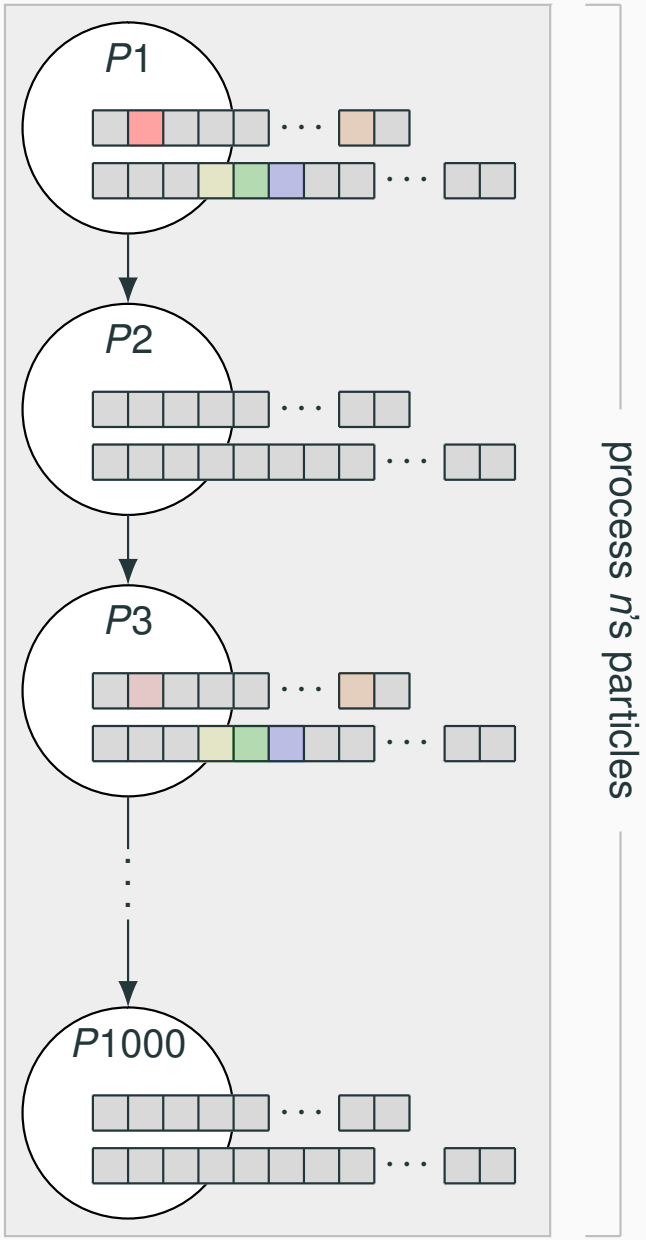
Method 1 - packing a buffer



Particles to send: $[1, 3]$

SEND BUFFER

Method 1 - packing a buffer

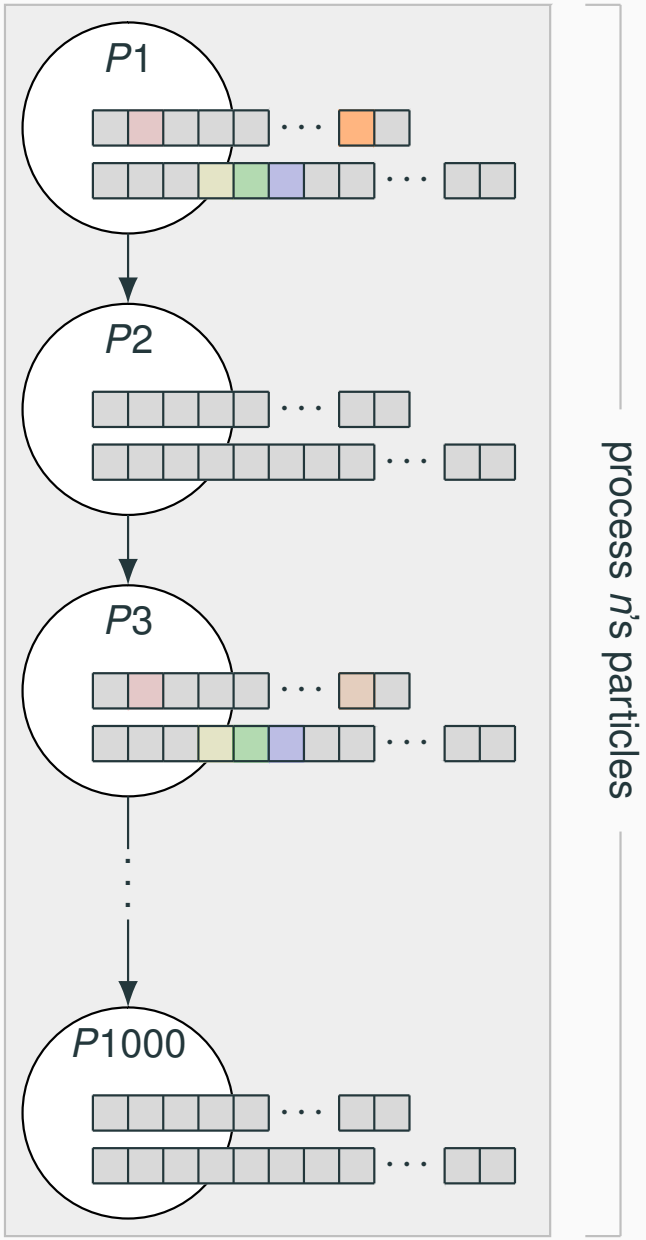


Particles to send: [1, 3]

SEND BUFFER

- MPI_Pack() function encodes data as char* in the buffer

Method 1 - packing a buffer



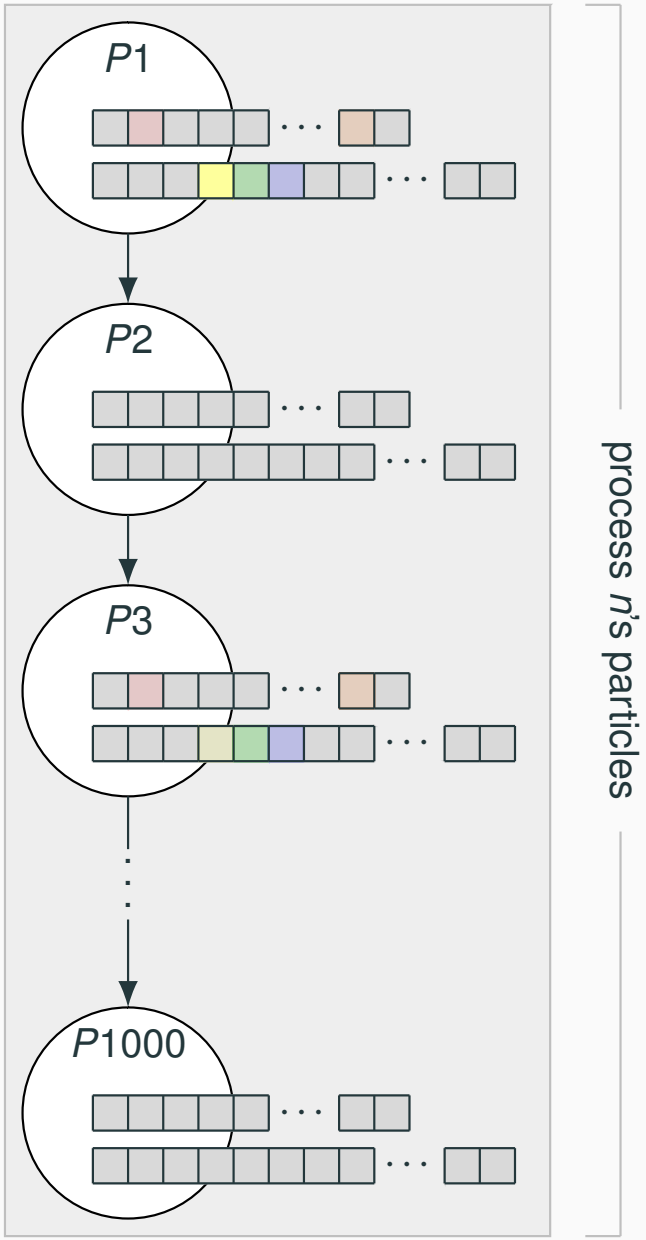
Particles to send: [1, 3]

SEND BUFFER

dcK6={tK?z

- MPI_Pack() function encodes data as char* in the buffer

Method 1 - packing a buffer



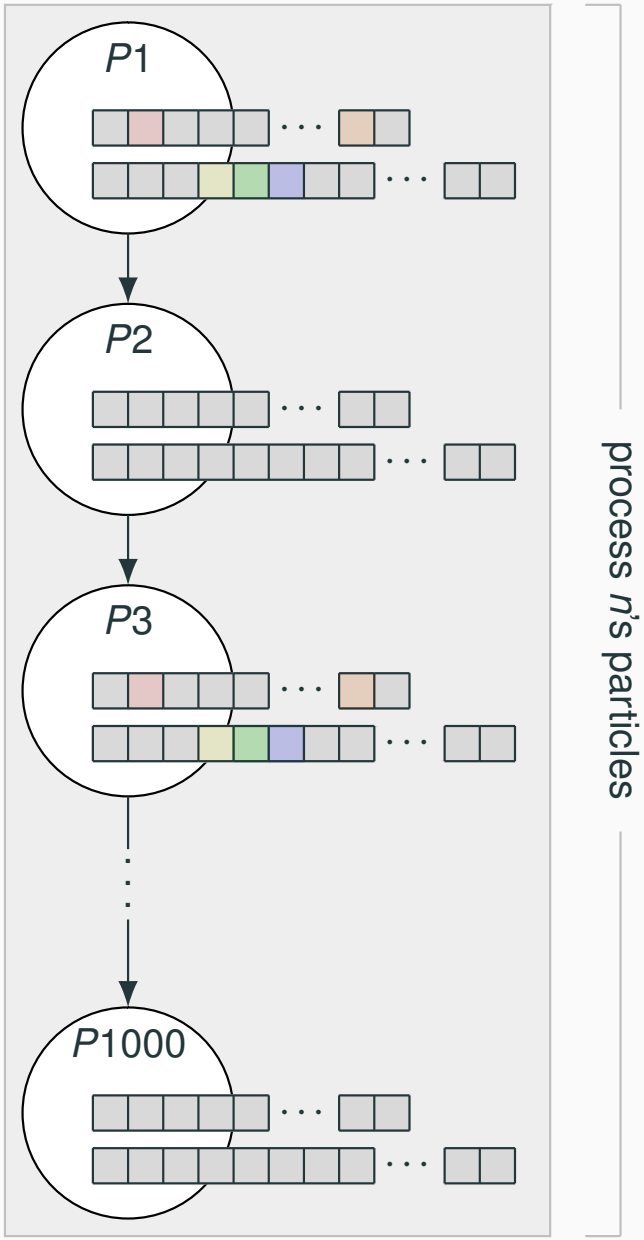
Particles to send: [1, 3]

SEND BUFFER

dcK6={tK?z Q#]J9

- MPI_Pack() function encodes data as char* in the buffer

Method 1 - packing a buffer



Particles to send: $[1, 3]$

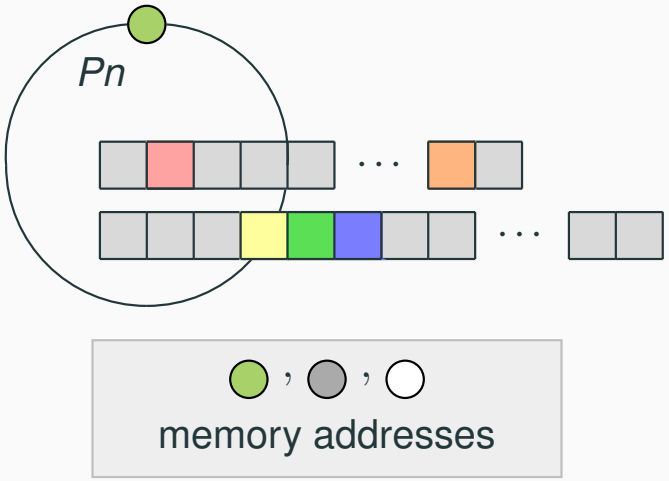
SEND BUFFER

dcK6={tK?zQ#]J9w3R]F#knMK$9&V,(&mrJ{C0RMbR:W$C1K

- MPI_Pack() function encodes data as char* in the buffer
- This char* buffer is sent to an adjacent process
- MPI_Unpack() function can decode char* encoded data

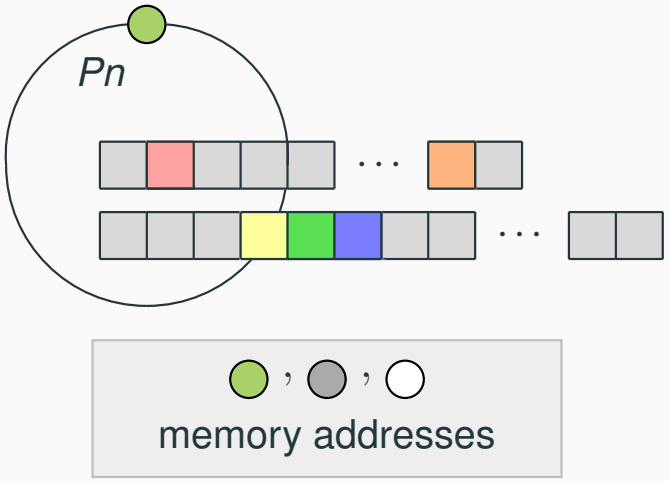
Method 2 - Memory Address Offset Datatypes

- Recall a single particle's contents.



Method 2 - Memory Address Offset Datatypes

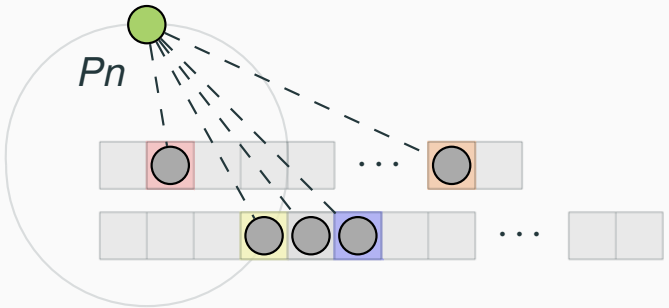
- Recall a single particle's contents.



- The same indices are being referenced to pull data from each particle. We can generalize this repetitiveness.

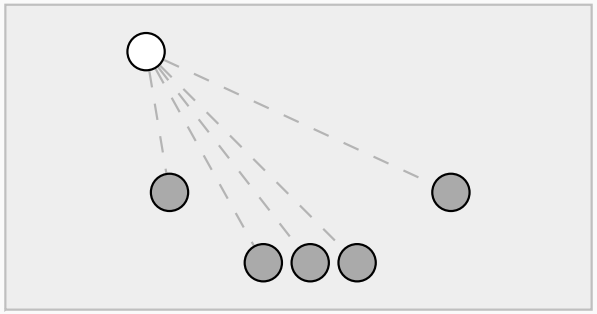
Method 2 - Memory Address Offset Datatypes

- Recall a single particle's contents.



- The same indices are being referenced to pull data from each particle. We can generalize this repetitiveness.
- Create an MPI Datatype to store the offsets of the **memory address of the particle** and the **memory address of the array indices**.

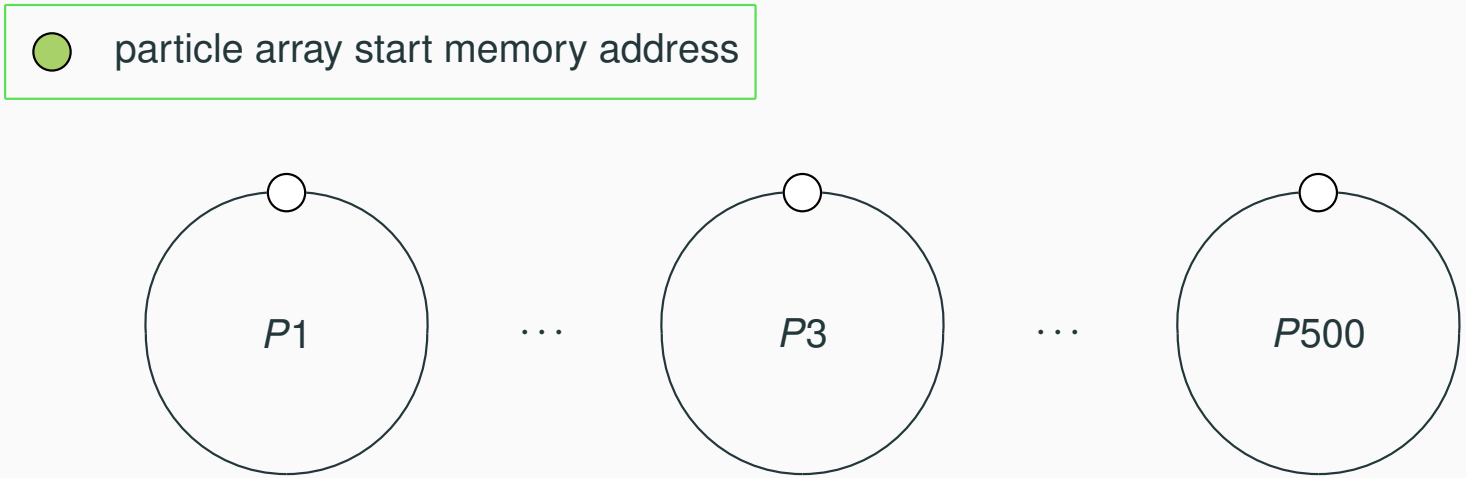
Method 2 - Memory Address Offset Datatypes



custom datatype 1

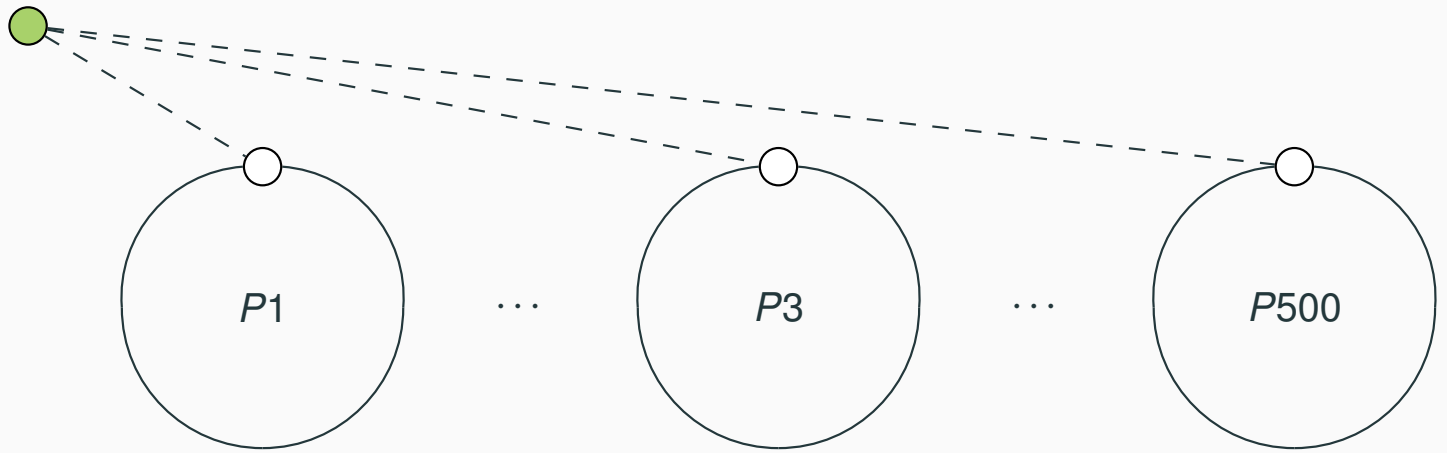
- The same indices are being referenced to pull data from each particle. We can generalize this repetitiveness.
- Create an MPI Datatype to store the offsets of the **memory address of the particle** and the **memory address of the array indices**.

Method 2 - Memory Address Offset Datatypes



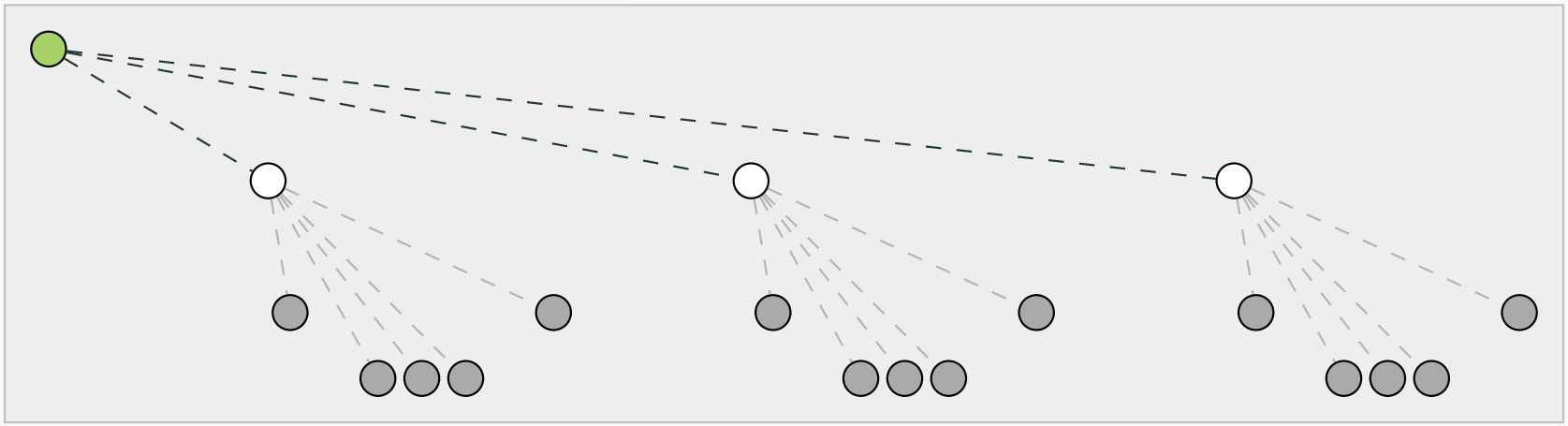
- The same indices are being referenced to pull data from each particle. We can generalize this repetitiveness.
- Create an MPI Datatype to store the offsets of the **memory address of the particle** and the **memory address of the array indices**.

Method 2 - Memory Address Offset Datatypes



- The same indices are being referenced to pull data from each particle. We can generalize this repetitiveness.
- Create an MPI Datatype to store the offsets of the **memory address of the particle** and the **memory address of the array indices**.
- Each time data needs to be sent, process data with a unique **second derived datatype**, created using memory offsets of sending particles from the start of the particle array

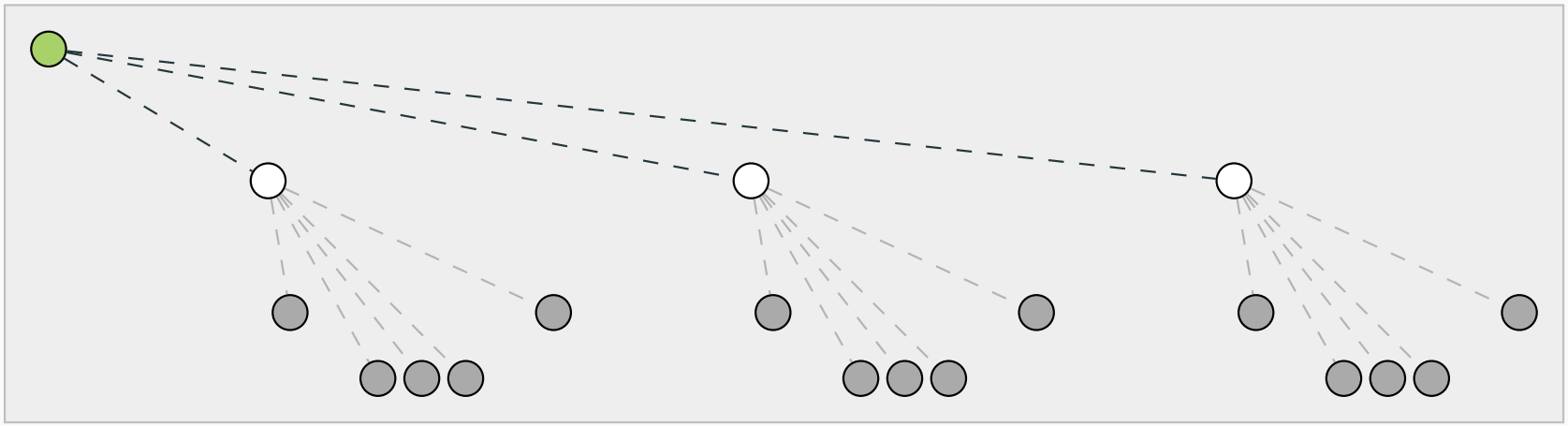
Method 2 - Memory Address Offset Datatypes



custom datatype 2

- The same indices are being referenced to pull data from each particle. We can generalize this repetitiveness.
- Create an MPI Datatype to store the offsets of the **memory address of the particle** and the **memory address of the array indices**.
- Each time data needs to be sent, process data with a unique **second derived datatype**, created using memory offsets of sending particles from the start of the particle array

Method 2 - Memory Address Offset Datatypes



custom datatype 2

- The same indices are being referenced to pull data from each particle. We can generalize this repetitiveness.
- Create an MPI Datatype to store the offsets of the **memory address of the particle** and the **memory address of the array indices**.
- Each time data needs to be sent, process data with a unique **second derived datatype**, created using memory offsets of sending particles from the start of the particle array

- Send one instance of the 2nd derived datatype
  - As long as we send to a particle* buffer, the received data will be readable.

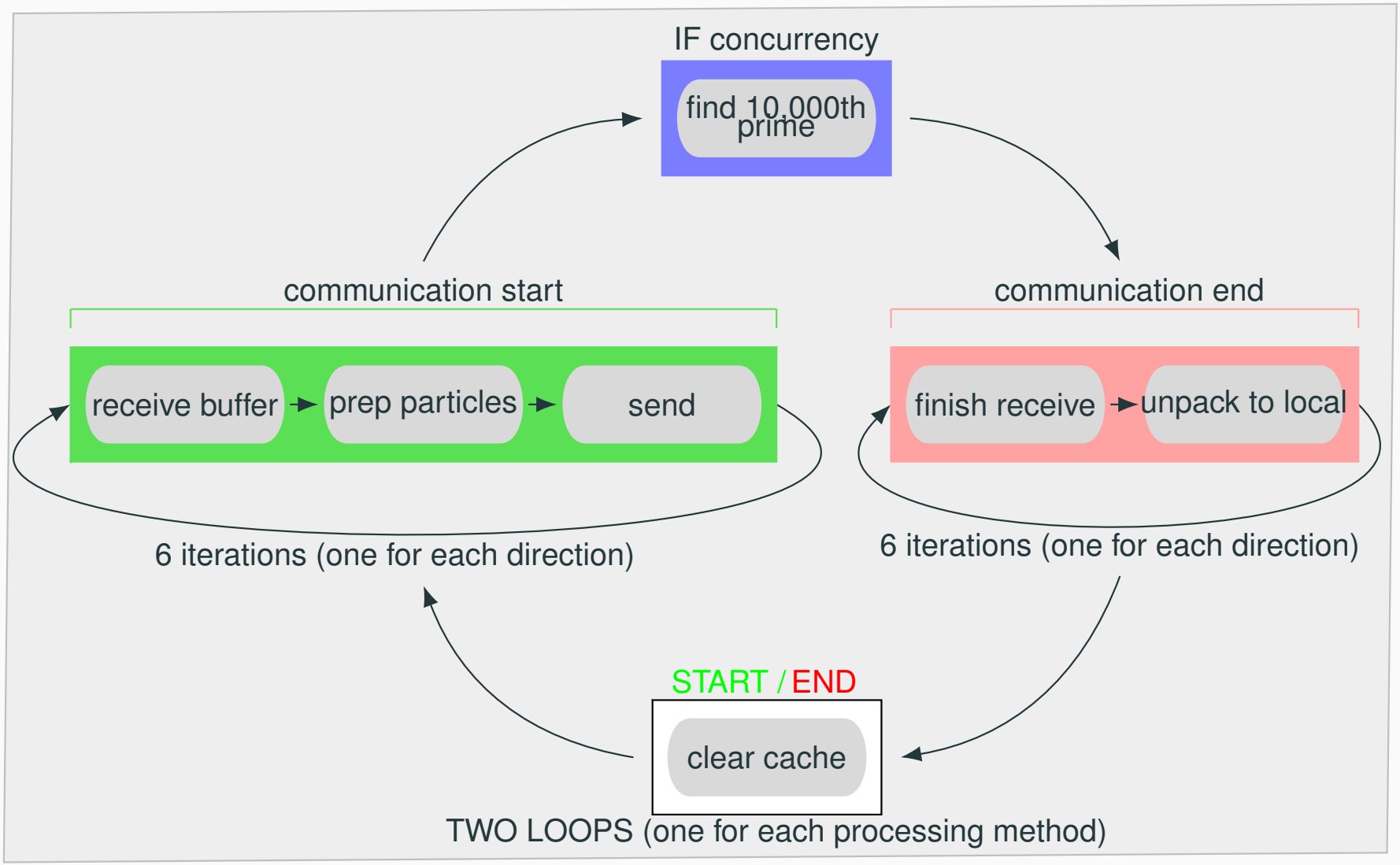
Methodology

Methodology

1. An input csv file specifies input data for a test run.
    - The number of particles to send to each adjacent process (2 - 8192).
    - Whether communication and computation should be performed concurrently.
2. Process 0 reads the input file and broadcasts this input data to the other processes with an MPI function call.
3. Each process is assigned a unique subvolume of the global volume and its adjacent processes are taken note of.
4. Each process is then given the same linked list of 5000 unique particles.

Methodology

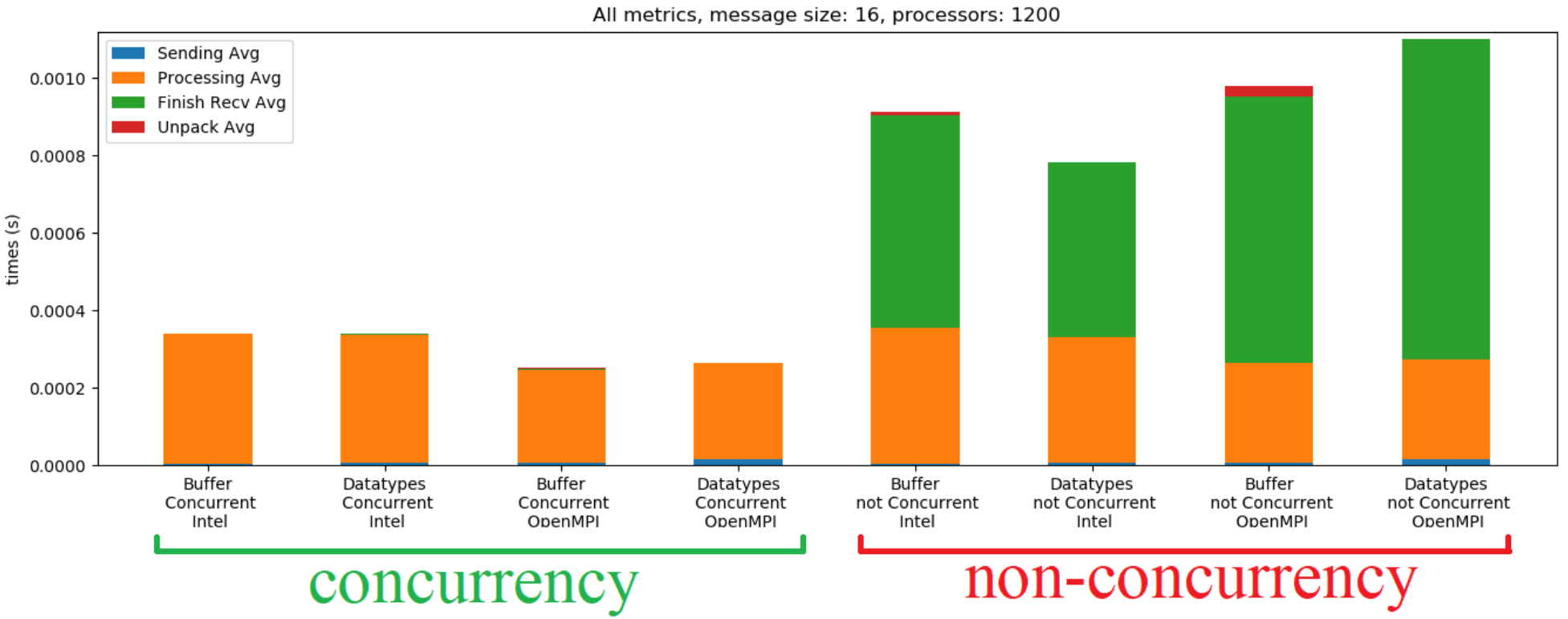
5. Each process chooses different particles to send to the 6 directions based off of a RNG seeded with the PID.
6. Obtain elapsed times for each primary functionality in the flowchart above.



Results

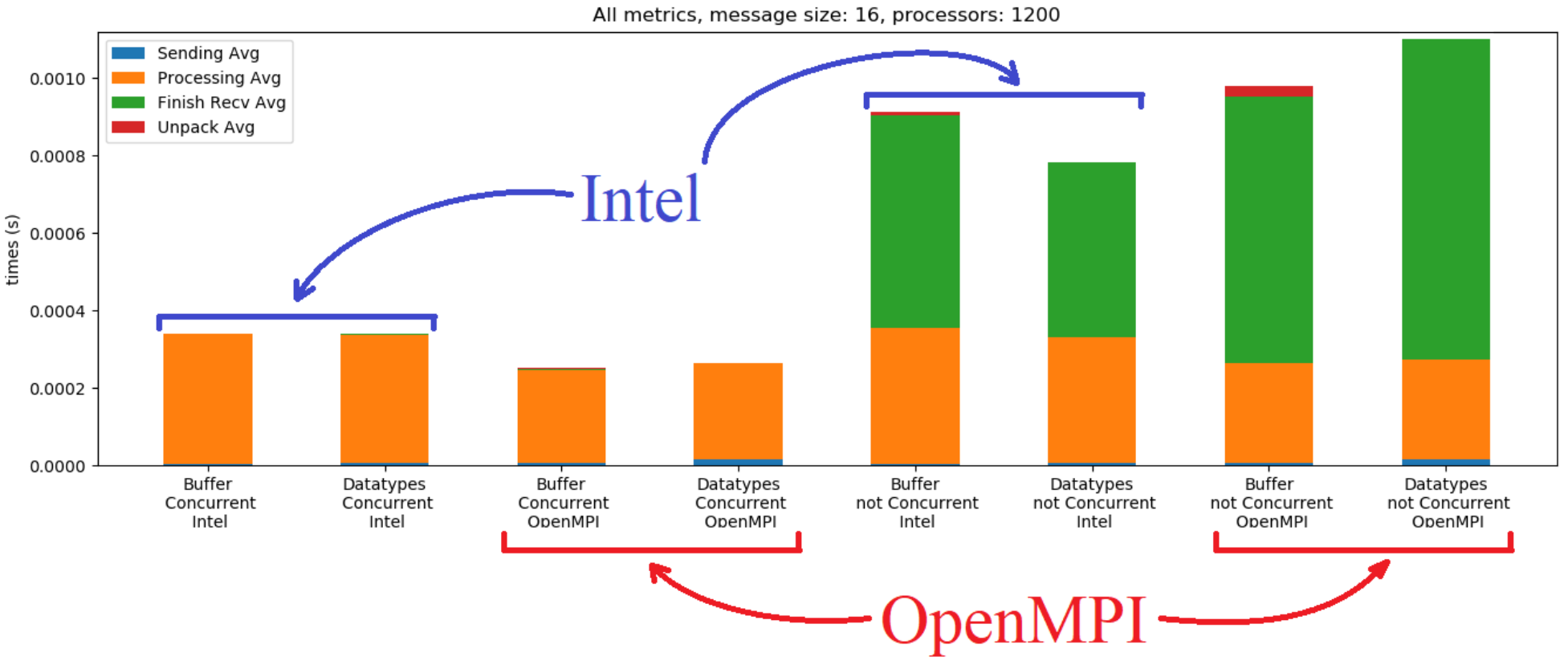
Results

- The communication testing C program was run on Raritan HPC with 300 - 1200 processes
  - Varying the number of processes does not change the output.
  - Varying the message size **does** however.



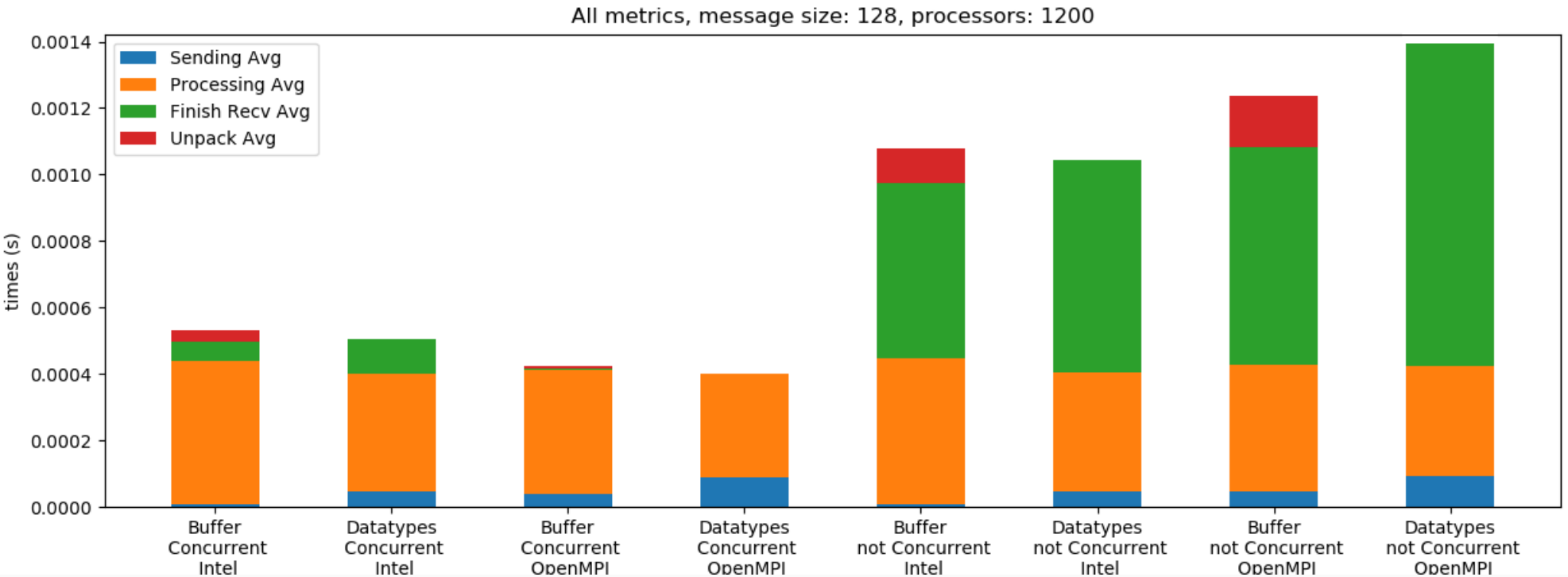
Results

- The communication testing C program was run on Raritan HPC with 300 - 1200 processes
  - Varying the number of processes does not change the output.
  - Varying the message size **does** however.



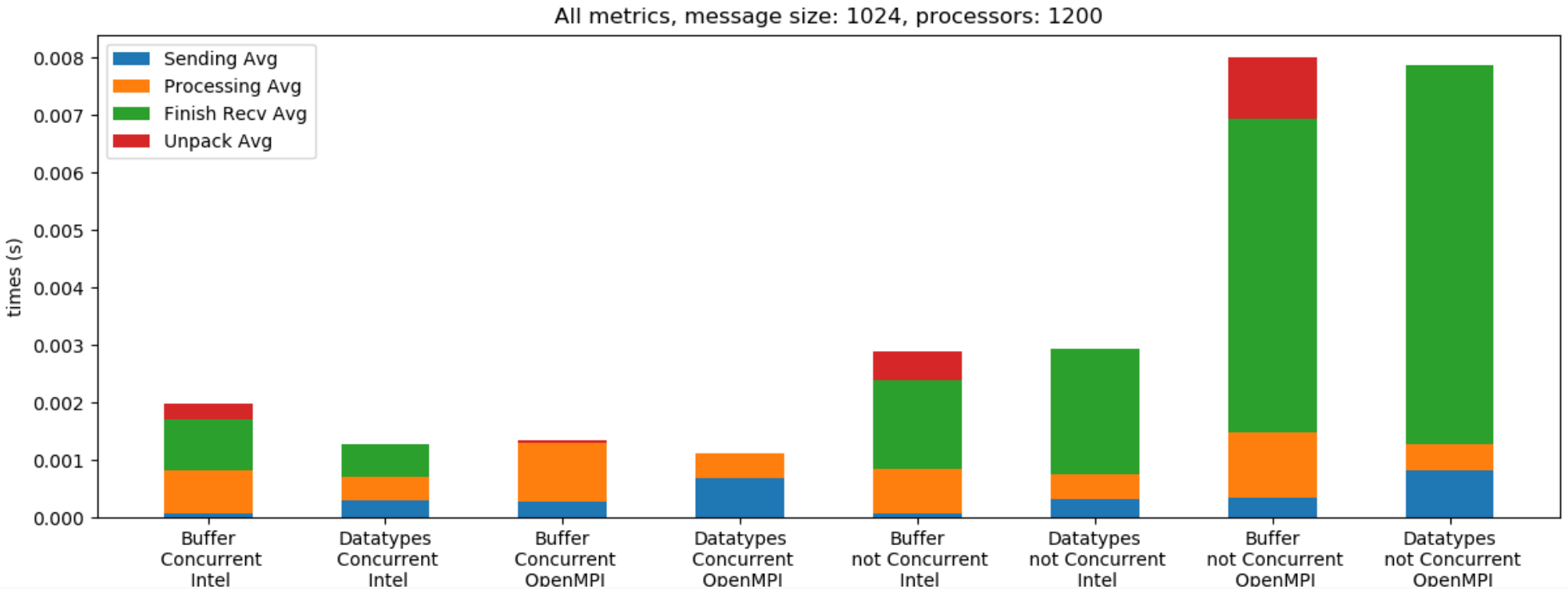
Results

- The communication testing C program was run on Raritan HPC with 300 - 1200 processes
  - Varying the number of processes does not change the output.
  - Varying the message size **does** however.



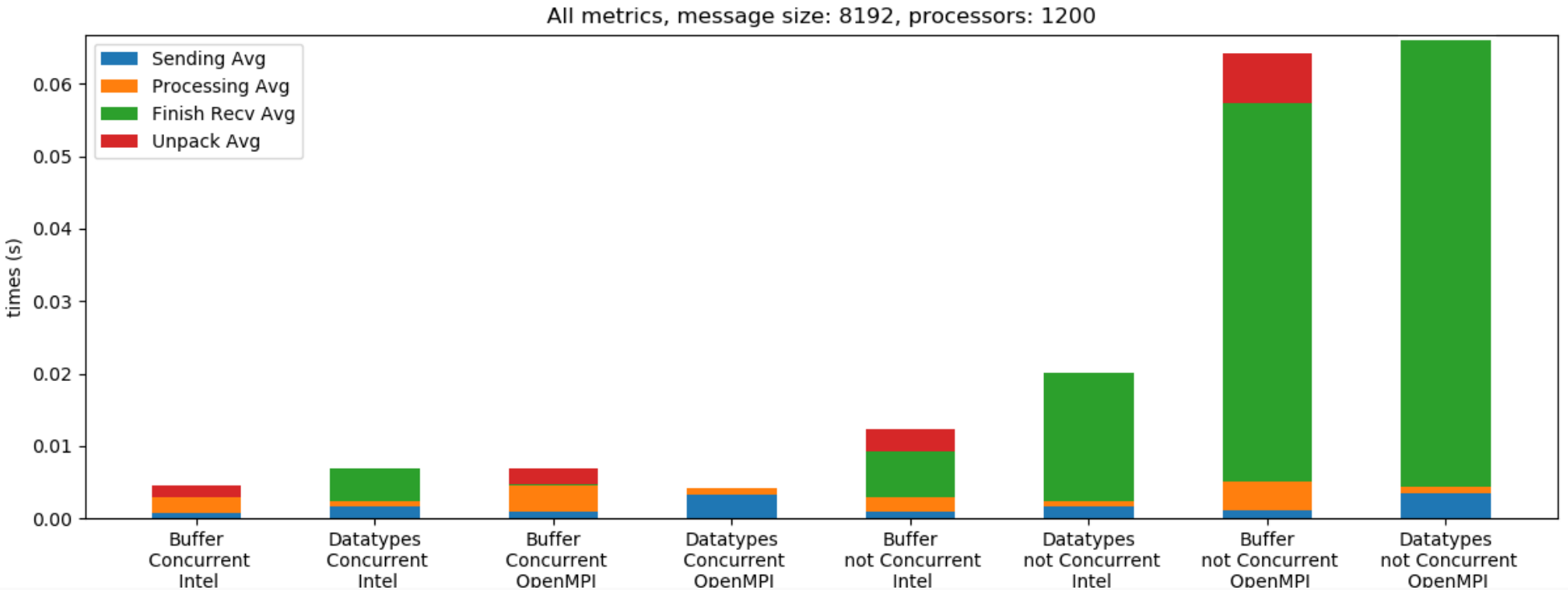
Results

- The communication testing C program was run on Raritan HPC with 300 - 1200 processes
  - Varying the number of processes does not change the output.
  - Varying the message size **does** however.



Results

- The communication testing C program was run on Raritan HPC with 300 - 1200 processes
  - Varying the number of processes does not change the output.
  - Varying the message size **does** however.



Results

Conclusions

- Communication **can** overlap with computation.
- The communication benchmark performance is not as simple as expected.
  - compiler
  - compiler options
  - MPI implementation
  - OS
  - other OS processes
  - network hardware
  - memory heirarchy

Future Work

- Overlap communication and computation whenever possible in the simulator
- Control some of the other variables mentioned in future tests.

Questions?