LA-UR-23-29621

Approved for public release; distribution is unlimited.

Title:	One-Sided MPI Window Communication for Domain Decomposed Implicit Monte Carlo
Author(s):	Heil, Raymond Thomas Scott, Cragun Taggart
Intended for:	Summer student final presentation
Issued:	2023-08-21

Los Alamos
NATIONAL LABORATORY

One-Sided MPI Window Communication for Domain Decomposed Implicit Monte Carlo

Cragun Scott, Ray Heil
Mentors: Alex and Kendra Long

August 18, 2023

LA-UR-23-XXXXX

About Us

⟵ Cragun Scott

- From Ogden, UT
- ME Undergrad at USU
- Enjoys Outside Adventures

Ray Heil ⟶

- From Gallup, NM
- CS Undergrad at Grinnell
- Local `sed` enjoyer

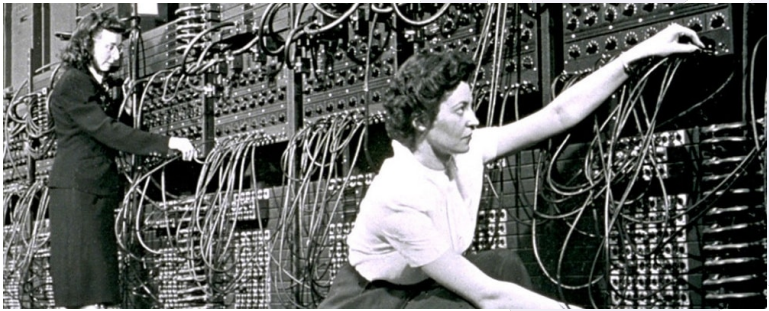
Los Alamos NATIONAL LABORATORY

Overview

1. Intro to IMC and Branson
2. Domain Decomposition
3. Project Goal and Motivation
4. MPI Windows and One-Sided Communication
5. Our Implementation
6. Testing and Results
7. Conclusion

Implicit Monte Carlo (IMC)

- Repeated pseudo-random sampling
- Used in high-fidelity radiation transport applications
- Implicit Monte Carlo developed at Livermore in the 1970s
  - Thermal radiative transport problems

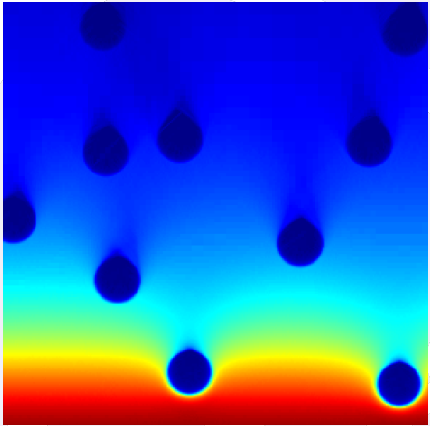


Branson

- Created by Alex Long (XCP-3)
- Implicit Monte Carlo radiation transport
- Used to study parallel algorithms
- Light, public-facing version of Jayenne

- Branson is **not** an acronym.

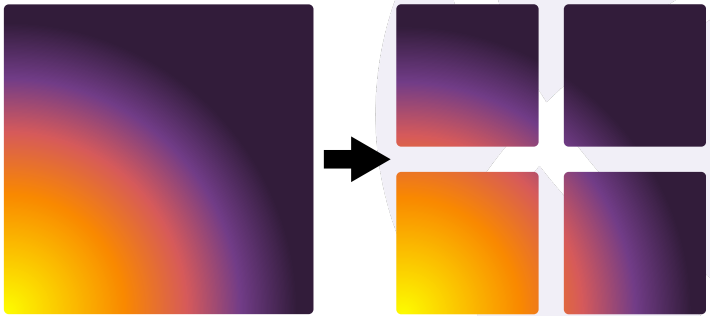
Branson uses a Domain Decomposed method to run most simulations.



Radiation wave through a foam with dense “inclusions.”

Domain Decomposition

- Distributes the problem across multiple MPI ranks
- Data is passed between ranks using MPI communication
- Uses less memory per rank, but requires more communication



Project Goal and Motivation

Go faster!

Less Communication Time $\Rightarrow$ More Simulation Time!

To do this, we explored one-sided communication

Two Communication Paradigms in MPI

Two-sided Communication

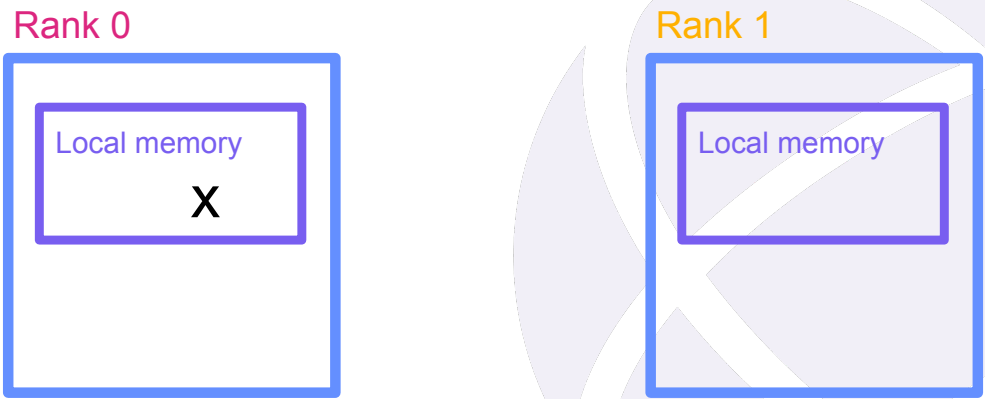
- Send and Recv
- More common, standard
- Both sides make MPI calls

One-sided Communication

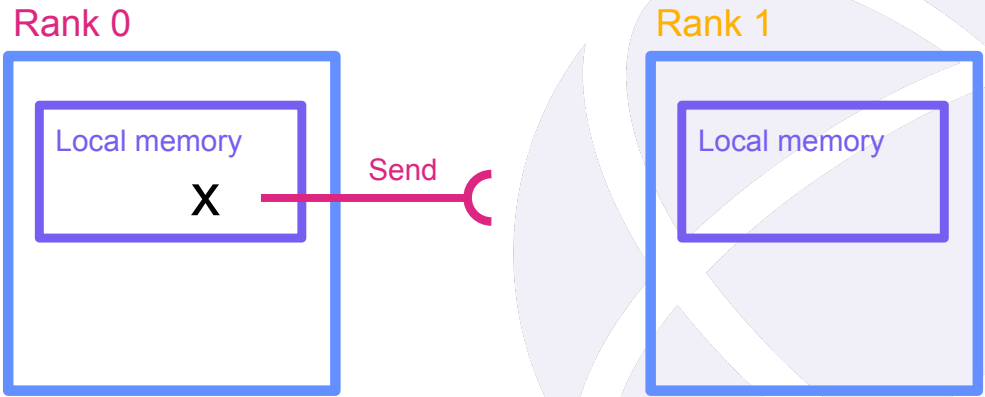
- Get or Put
- Less common, implementations vary
- Only one side makes MPI calls

Let's take a quick look at how each of these paradigms works!

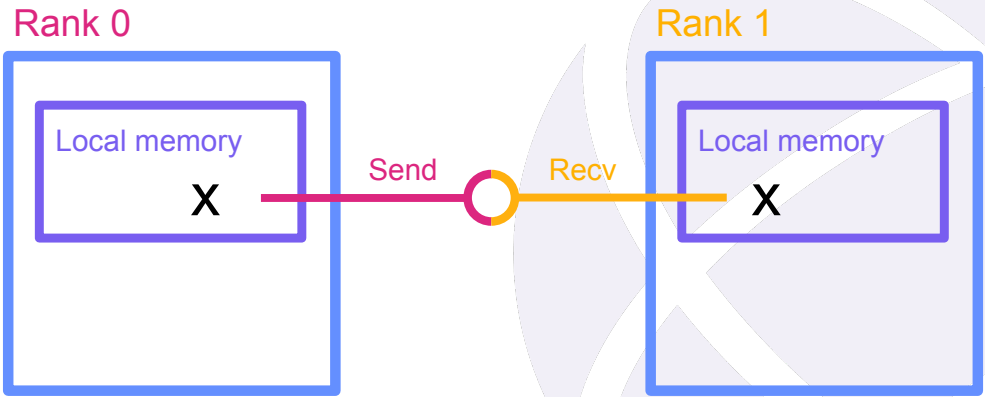
Two-Sided Communication: Initial State



Two-Sided Communication: Send

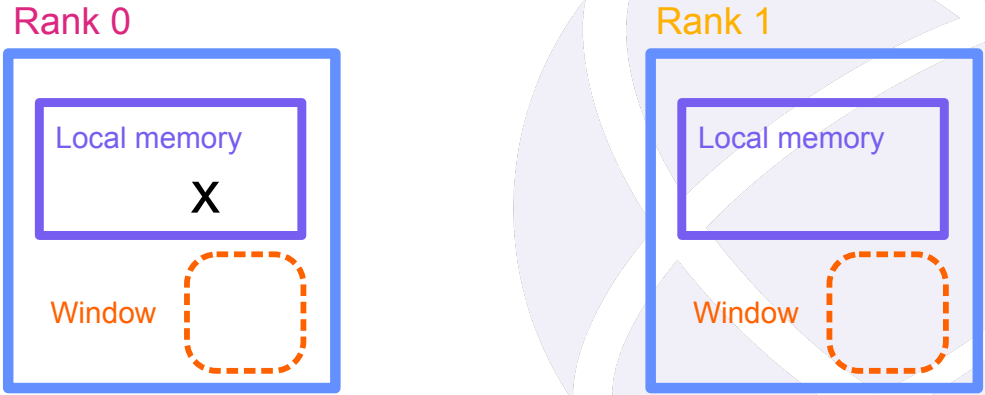


Two-Sided Communication: Recv



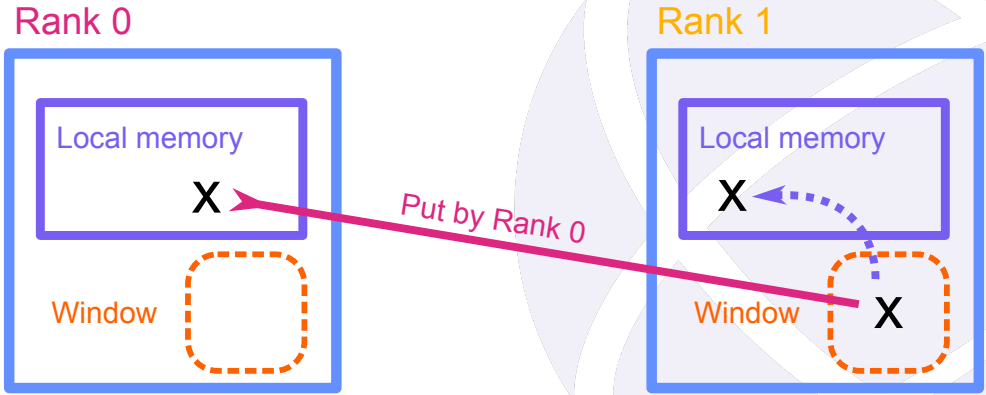
One-Sided Communication: Window Setup

We allocate some of our memory to serve as a window for remote memory access



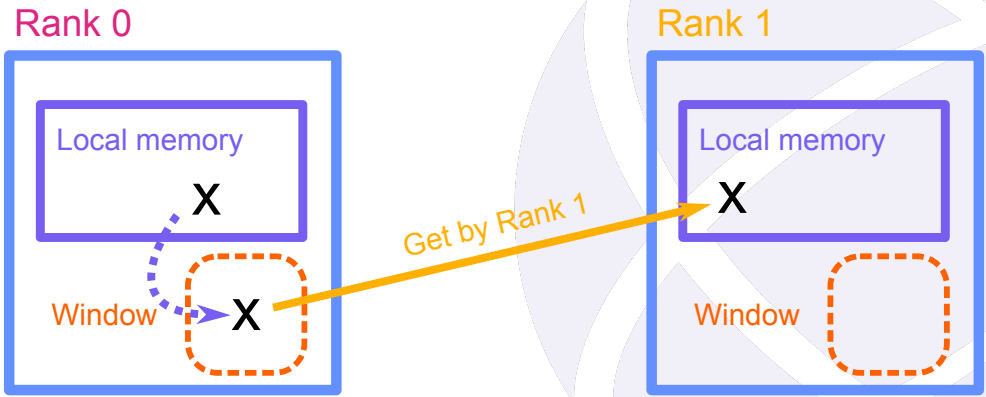
One-Sided Communication: Put

From local memory to a remote window



One-Sided Communication: Get

From a remote window to local memory



Our Implementations

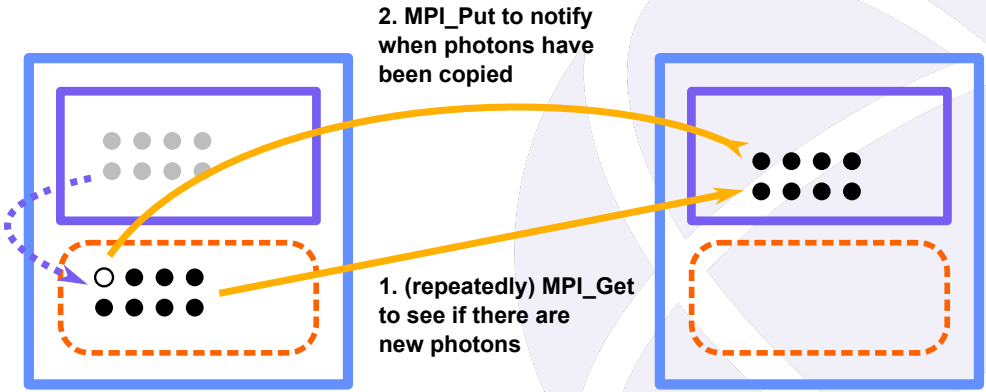
Two implementations:

- Outgoing photons go in our local window (“pickup”)
- Outgoing photons go in a remote window (“delivery”)

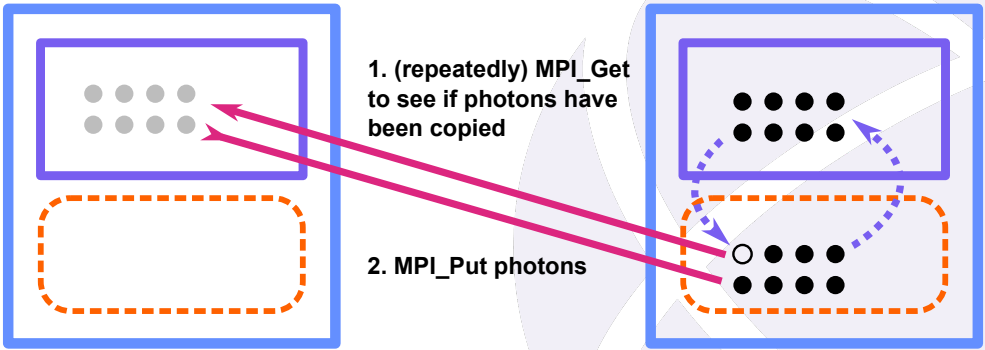
Common aspects:

- Transfer photons through window buffers
- Mark photons as done once copied
- Only write to an “open” window (marked as done)
  - Involves repeated checking
- Blocking operations

Pickup Implementation



Delivery Implementation



Difficulties

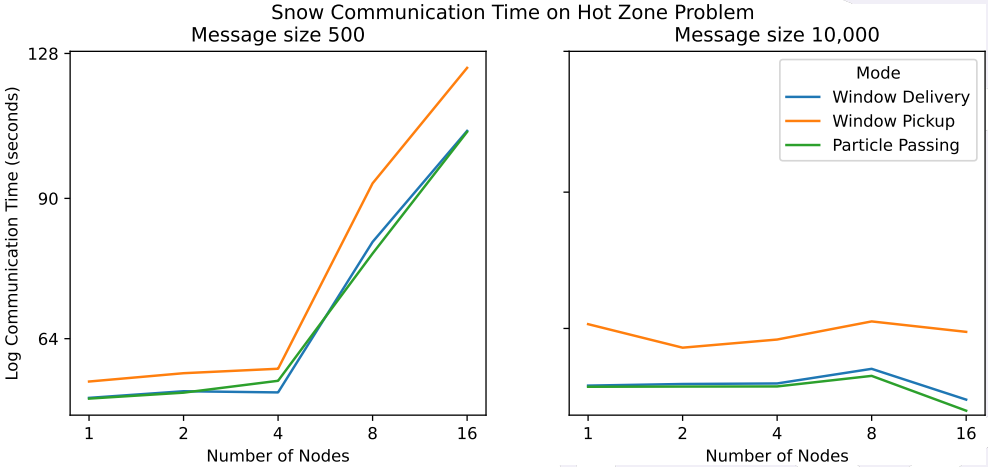
Window Issues

- Documentation is less clear than traditional comm
  - Shared vs Exclusive locks
- Not intended for asynchronous reading and writing
- Requires manual checking for transfer completion
  - Only one side of communication knows about completion status
- Debugging
  - hard to track corrupted data

Setbacks

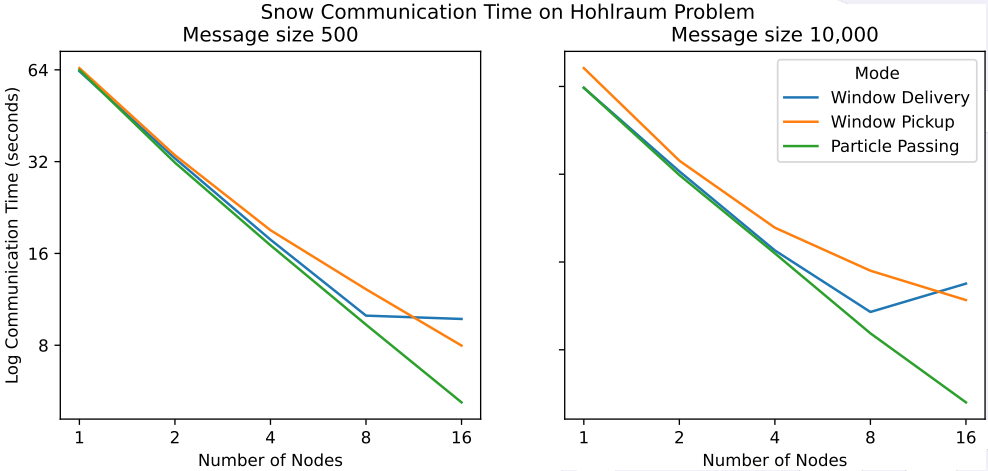
- Easy to transfer invalid or incomplete data
- Possible to overwrite your own data prematurely

Imbalanced Problem on Snow



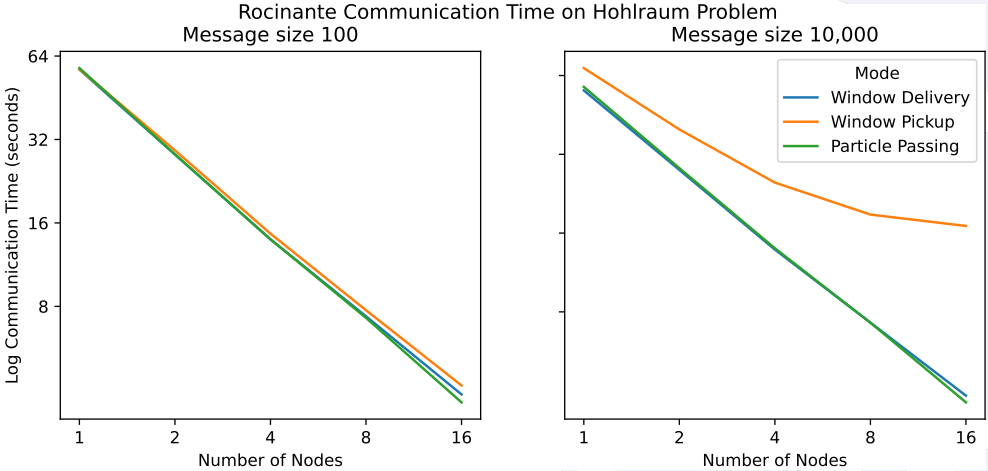
Window and Particle Passing methods on a Hot Zone problem

Balanced Problem on Snow



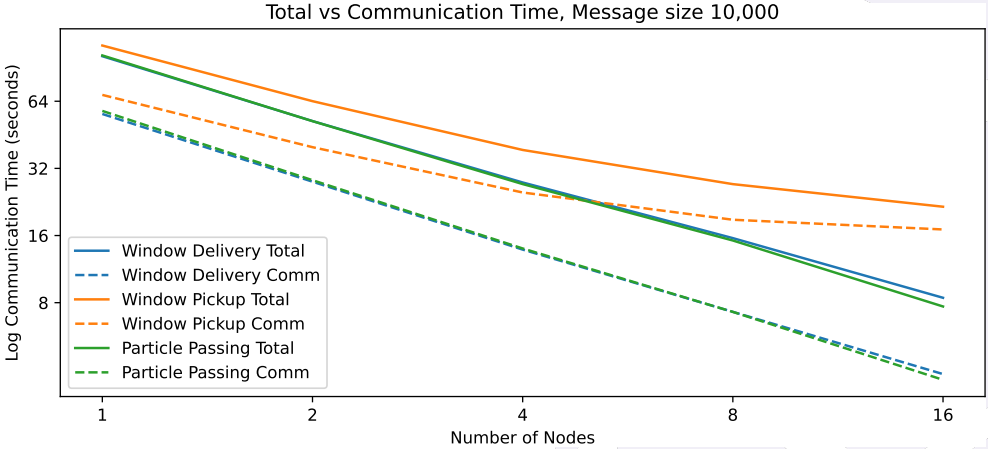
Window and Particle Passing methods on a 3D Hohlraum problem

Machine Matters: Balanced Problem on Rocinante



Window and Particle Passing methods on a 3D Hohlraum problem

Communication Time



Total run time compared to total communication time on 3D Hohlraum problem

Conclusion

- Using windows was harder to implement than anticipated
  - Vague documentation
  - MPI versions affected basic window behavior
  - Hard to debug non-reproducible behavior
- Our implementations run as fast as two sided non-blocking
  - Pickup method slower due to getting the entire message size every time
- Window communication may run faster after optimization efforts
- Other one-sided algorithms may be work better

Future Work

- Shared memory: openSHMEM (CPU) / NVSHMEM (GPU)
  - Potentially lower latency
  - NVSHMEM is a path for network communication inside GPU Kernels
- Fewer window locks when possible
  - Reduce cost of repeatedly locking/unlocking
- Explore other algorithms
  - Could allow for asynchronous memory access
  - Better buffering of outgoing photons
  - Use a “get-get” method

Any questions?

Thank you for coming to our talk!