CRD

Experiences of Applying One-Sided Communication to Nearest-Neighbor Communication

Hongzhang Shan[1], Weiqun Zhang[1], Yili Zheng[1], Samuel Williams[1], Stephane Ethier[2], Bei Wang[3], Zhengji Zhao[3]

1) CRD, LBNL; 2) PPPL 3) ; PICSciE, Princeton Univ. 4) NERSC, LBNL

PGAS Applications Workshop, SC16

Salt Lake City, Utah, Monday, Nov 14th, 2016

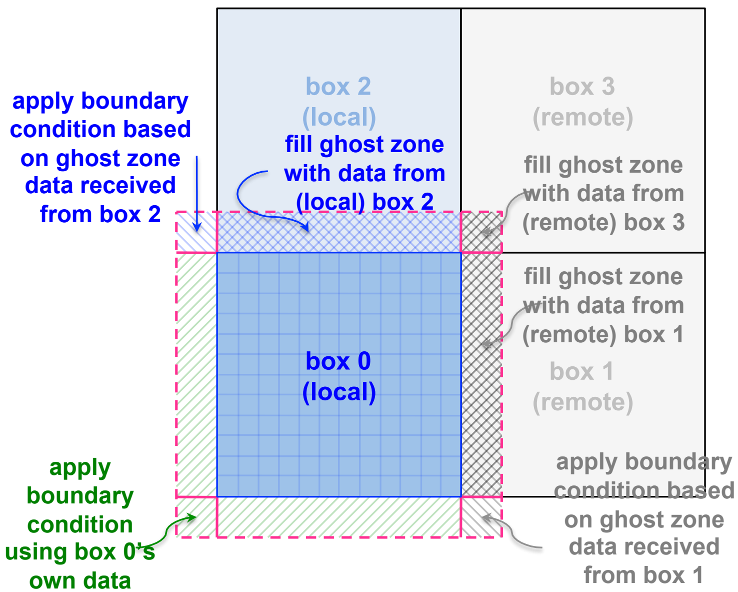
PRINCETON UNIVERSITY

PPPL PRINCETON PLASMA PHYSICS LABORATORY

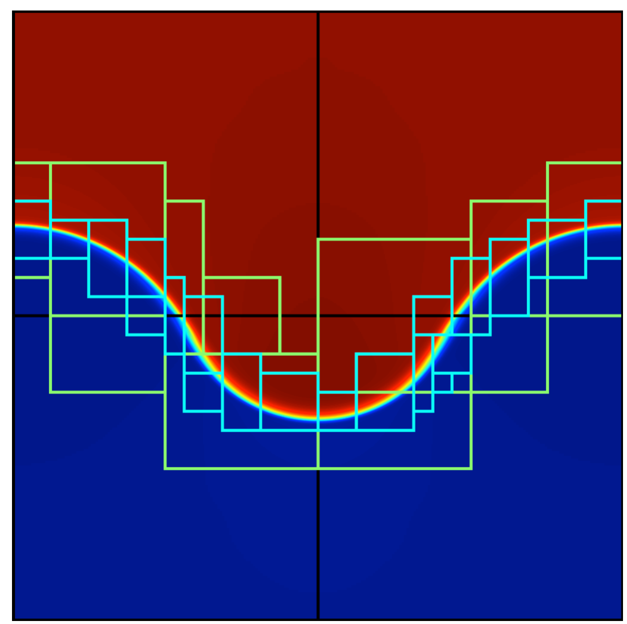
NERSC

Nearest-neighbor Communication

- One of the most important communication patterns, appearing in many scientific applications, such as HPGMG, Boxlib, GTCP, LAMMPS, CoMD, MILc, Luesh, etc.



HPGMG



Boxlib

Typical MPI Implementation

- Using point-to-point communication
- Data are aggregated so at most one message between a pair of processes for better performance
  - Packing/Unpacking
- Often implemented with:
  - Non-blocking functions
  - Overlap packing / unpacking and local computation with data communication
- Fit well with two-sided MPI messages

Questions

- With the arrival of MPI3 RMA, can MPI one-sided outperform the commonly used two-sided implementations ?
- How about UPC++, which also implements the one-sided message and supports Partitioned Global Address Space (PGAS) and Active Messages (AM) ?

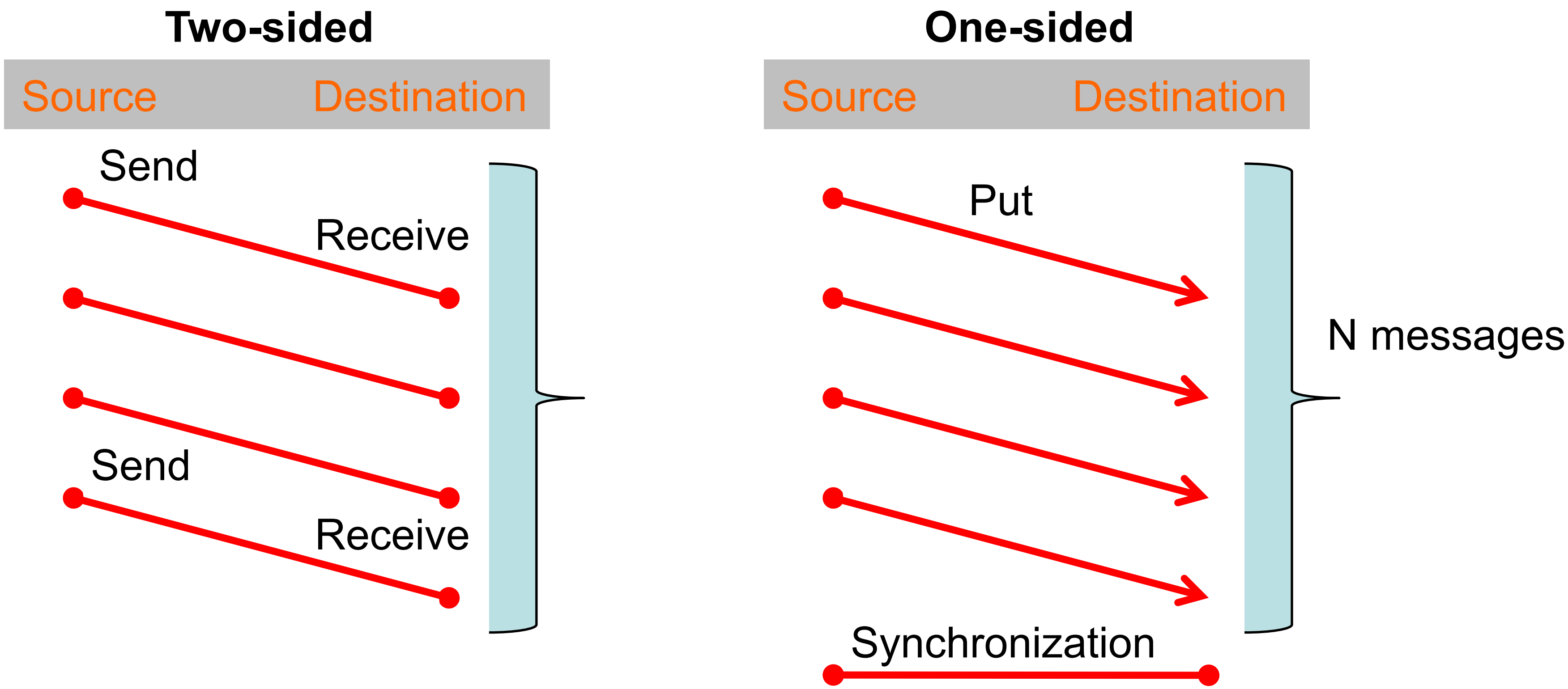
Outline

- Differences between two-sided messages and one-sided messages
- Differences between MPI one-sided and UPC++
- Application performance
  - GTC-P
  - Boxlib

MPI Two-sided vs. MPI One-sided

- **Message Information**
  - Two-sided: sender knows source information, receiver knows destination information
  - One-sided: message initiator knows both source and destination information
- **Synchronization**
  - Two-sided: Implicit synchronization, one sender matching one receiver
  - One-sided: Separates data transfer and synchronization, explicit synchronization

One-sided Advantage



- **Receiver has to match N sends with N receives**

- **Receiver only needs to be involved at synchronization point**
- **Sender can control the message sizes**

MPI One-sided API (subset)

- **Create a window object**
  - MPI_Win_create(base, size, …, win)
  - MPI_Win_create_dynamic (win)
- **Data Transfer:**
  - **MPI_Put : non-blocking**
  - MPI_Put (origin_addr, origin_count, origin_type
    target, target_disp, target_count, target_type, win)
- **Synchronization (win based)**
  - **MPI_Win_fence**
  - **MPI_PSCW (Post + Start + Complete + Wait)**
  - **MPI_Win_lock/MPI_in_unlock**

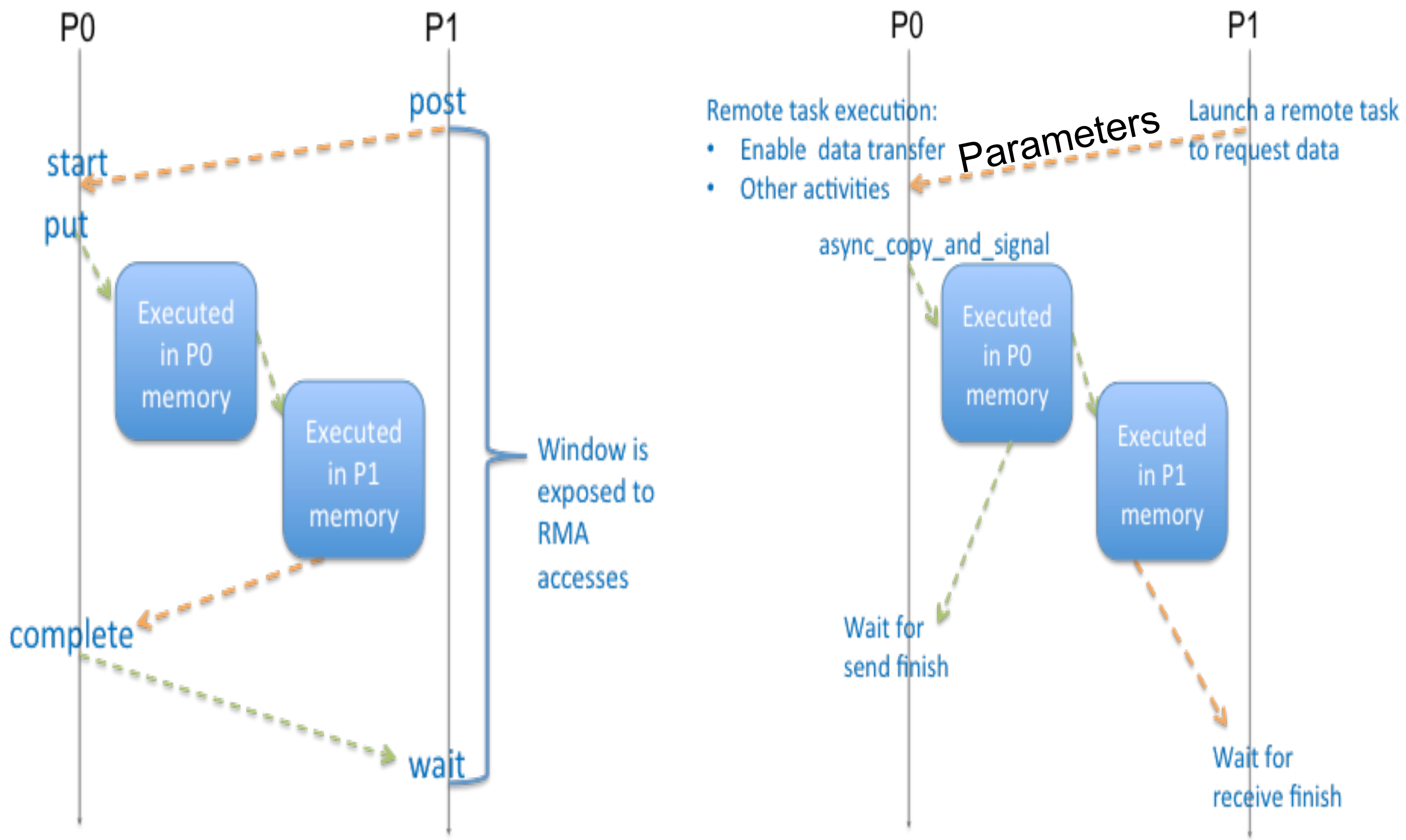
UPC++

- Supports Partitioned Global Address Space (PGAS)
  – Global pointer : global_ptr<double> gp
- Data transfer
  – async_copy(src, dest, count)
  – async_copy_and_signal(src, dest, count, event…)
- Synchronization
  – Global data (spin-waiting)
  – remote task execution
    - async(remote, *event)(func, args)
  – Barrier

CRD

MPI (PSCW) vs. UPC++

MPI (PSCW) Remote Task



Applications

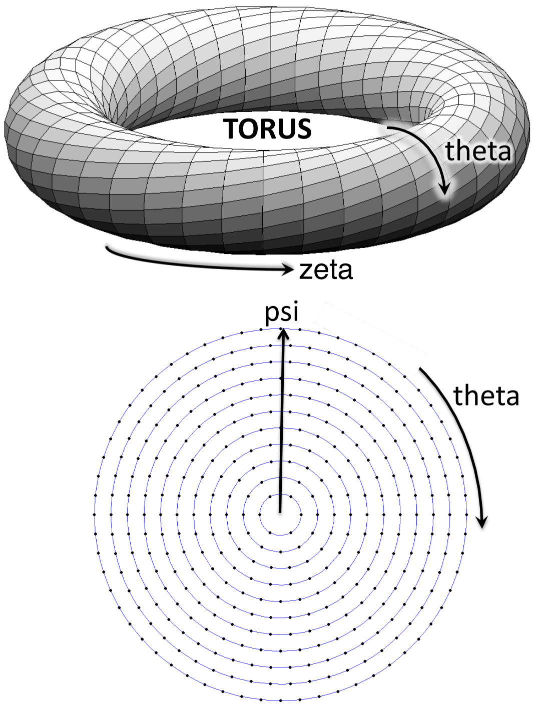
- GTC-P
  - Communicate with left and right neighbors
  - Large message size
- Boxlib
  - Dynamic nearest neighbor communication
  - All message initiators have to collect destination info before the data transfer

Approach

- Focus on the communication only
  - GTC-P shift
  - Boxlib FillBoundary
- Replace MPI two sided calls with MPI one-sided or UPC++ API, optimize where applicable

Gyrokinetic Toroidal Code (GTC-P)

- A particle-in-cell (PIC) code that solves the five-dimensional (5D) gyrokinetic Vlasov-Poisson equation in full, global torus geometry to address turbulence issues in tokamaks

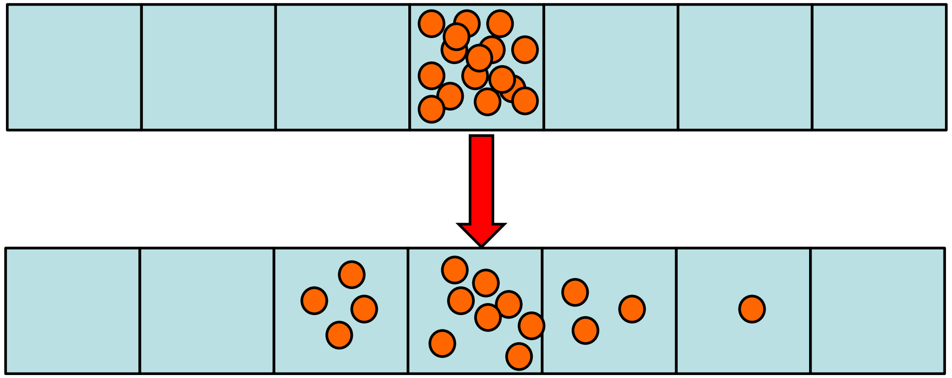


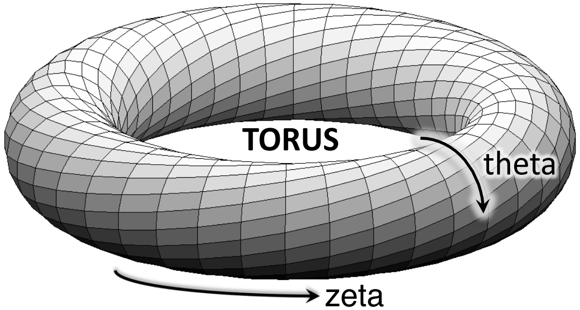
2D "Poloidal plane"
mgrid = total number of points

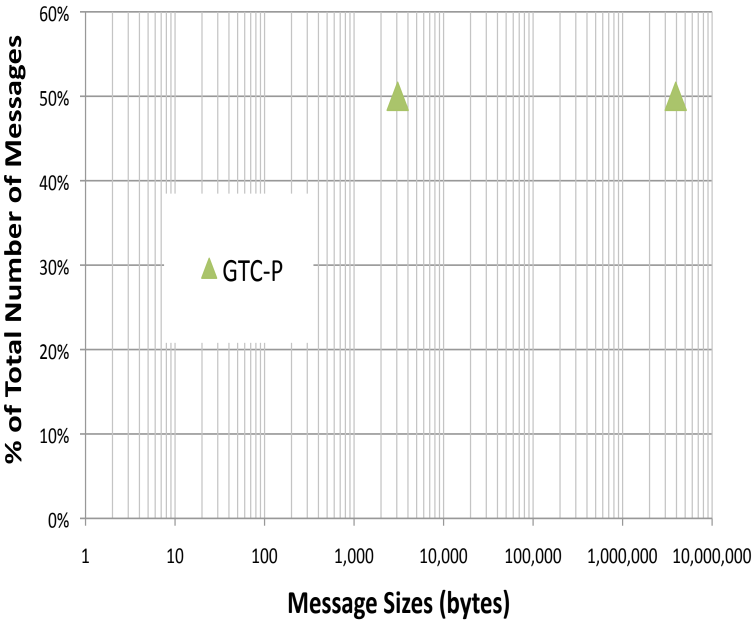
- Domain partitioned in both zeta and psi direction
- Particles (electrons and ions) following the same partition in zeta direction
- Network performance becomes increasingly important for the overall performance

Communication Pattern (shift)

- Focus on particle movement, mainly in zeta direction and only across a few subdomains
- Communicate with left and right neighbors only



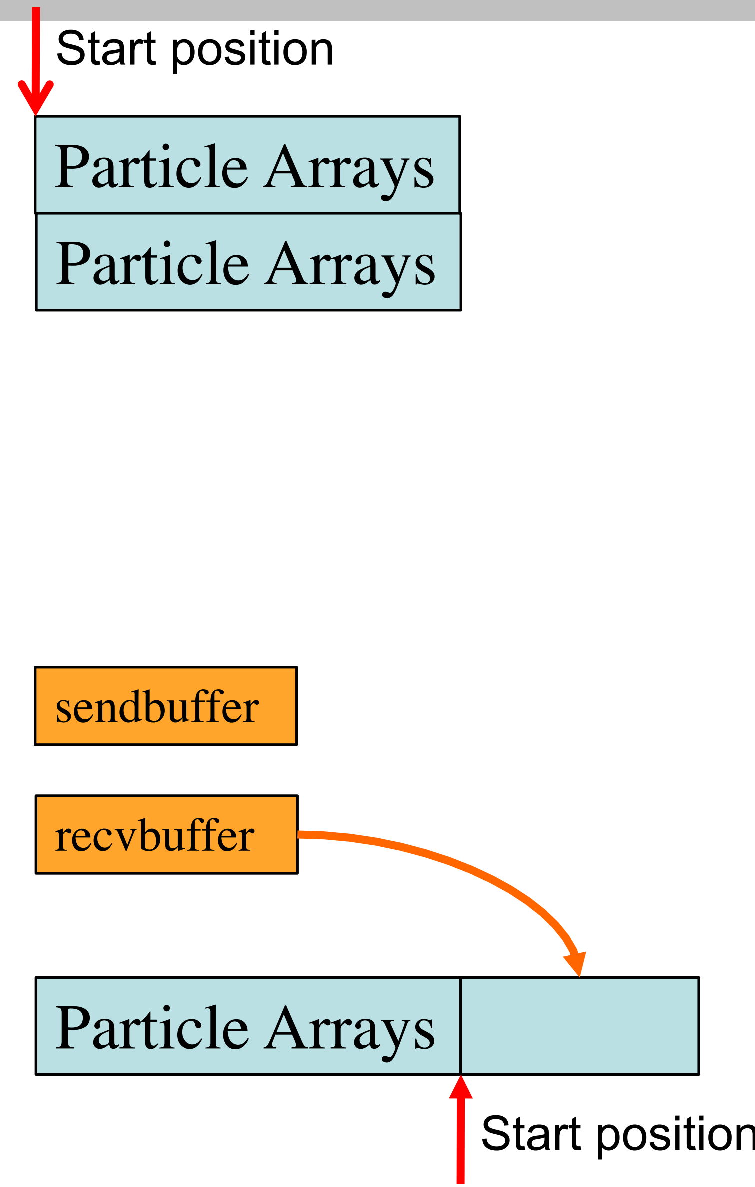




CRD

MPI Two-sided Implementation

1. **Scan particle arrays from position start to compute "*msendleft*" and "*msendright*"**
2. **Exchange *msendleft* and *msendright* with left and right neighbors using MPI_Sendrecv**
3. **Pack particles into *sendbuffer***
4. **Call MPI_Sendrecv to send data into *recvbuffer***
5. **Unpack data from recvbuffer to the end of particle array**



MPI One-sided Implementation

MPI Two-sided	MPI One-sided

1. Scan particle arrays from position start to compute "*msendleft*" and "*msendright*"
2. Exchange *msendleft* and *msendright* with left and right neighbors using MPI_Sendrecv
3. Pack particles into *sendbuffer*
4. Call MPI_Sendrecv to send data into *recvbuffer*
5. Unpack data from recvbuffer to the end of particle array

1. MPI_Win_create(recvbuffer, ….., win)
2. Exchange displacements
3. Create send (sgroup) and receive (rgroup)

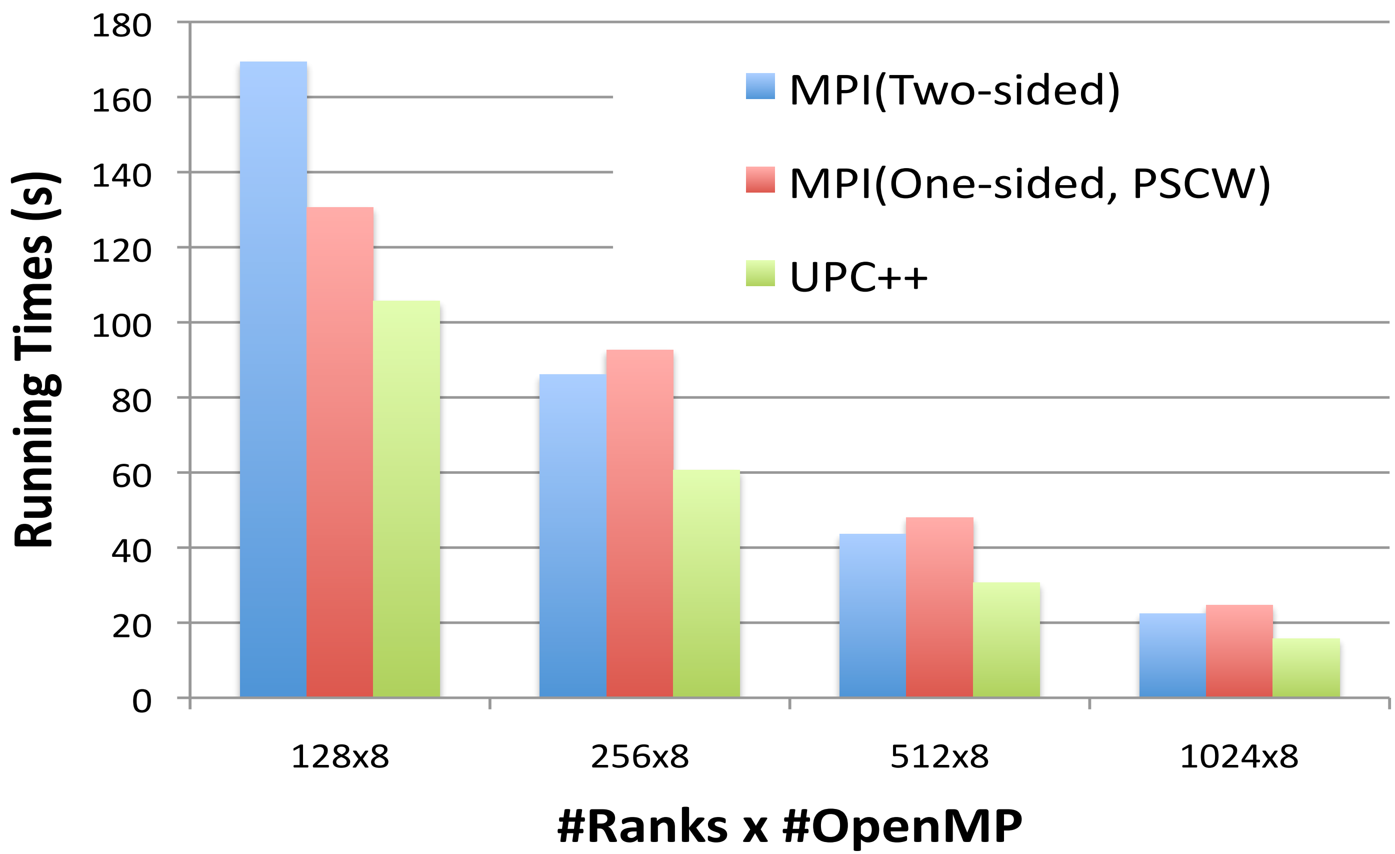
1. MPI_Win_post(rgroup, win)
2. MPI_Win_start(sgroup, win)
3. MPI_Put(left, disp))
4. MPI_Put(right, disp)
5. MPI_Win_complete(win)
6. MPI_Win_wait(win)

UPC++ Implementation

MPI One-sided	UPC++
1. MPI_Win_create(recvbuffer, …., win) 2. Exchange displacements 3. Create send (sgroup) and receive (rgroup)	1. Allocate sendbuffer, recvbuffer in global address space 2. Store shared_ptr in global variables
1. MPI_Win_post(rgroup, win) 2. MPI_Win_start(sgroup, win) 3. MPI_Put(left, disp)) 4. MPI_Put(right, disp) 5. MPI_Win_complete(win) 6. MPI_Win_wait(win)	1. Async(left)(func) 2. Async(right)(func) 3. Async_copy(src, leftbuf, count) 4. Advance() 5. Async_copy(src, rightbuf, count) 6. Advance() 7. Async_wait()

CRD

GTC-P Shift Performance on Cori



- **Run with MPI+OpenMP**
- **MPI One-sided scales worse than MPI two-sided**
- **UPC++ scales best**

Time Breakdown for Case 256*8

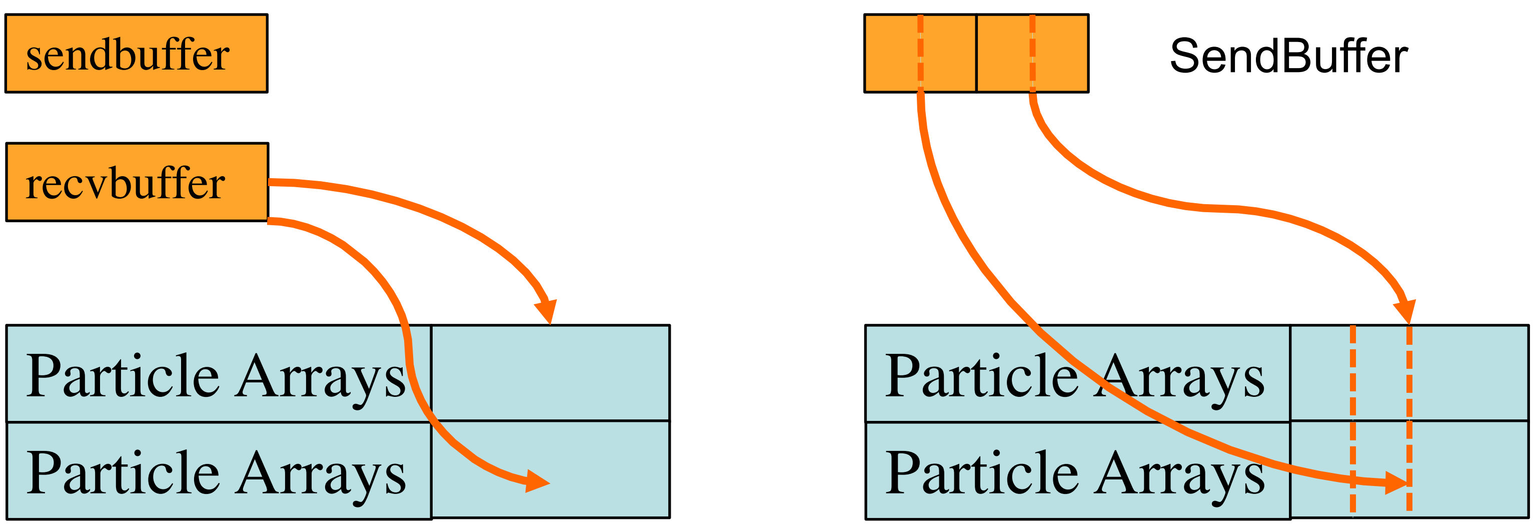
	Packing (s)	Comm (s)	Unpacking (s)	Imbalance (s)	Total (s)
MPI (Two-sided)	17.2	39.0	4.9	24.2	85.3
MPI (One-sided)	17.2	54.3	4.9	16.0	92.5
UPC++	18.8	18.4	7.2	15.1	59.4

- **Main difference lies in communication time**
- **In UPC++, after the data transfer, advance() is called to push the data onto network immediately**

CRD

One-sided Optimization

- **Directly send data from source to destination**
- **Apply message pipelining to overlap packing and data transfer**



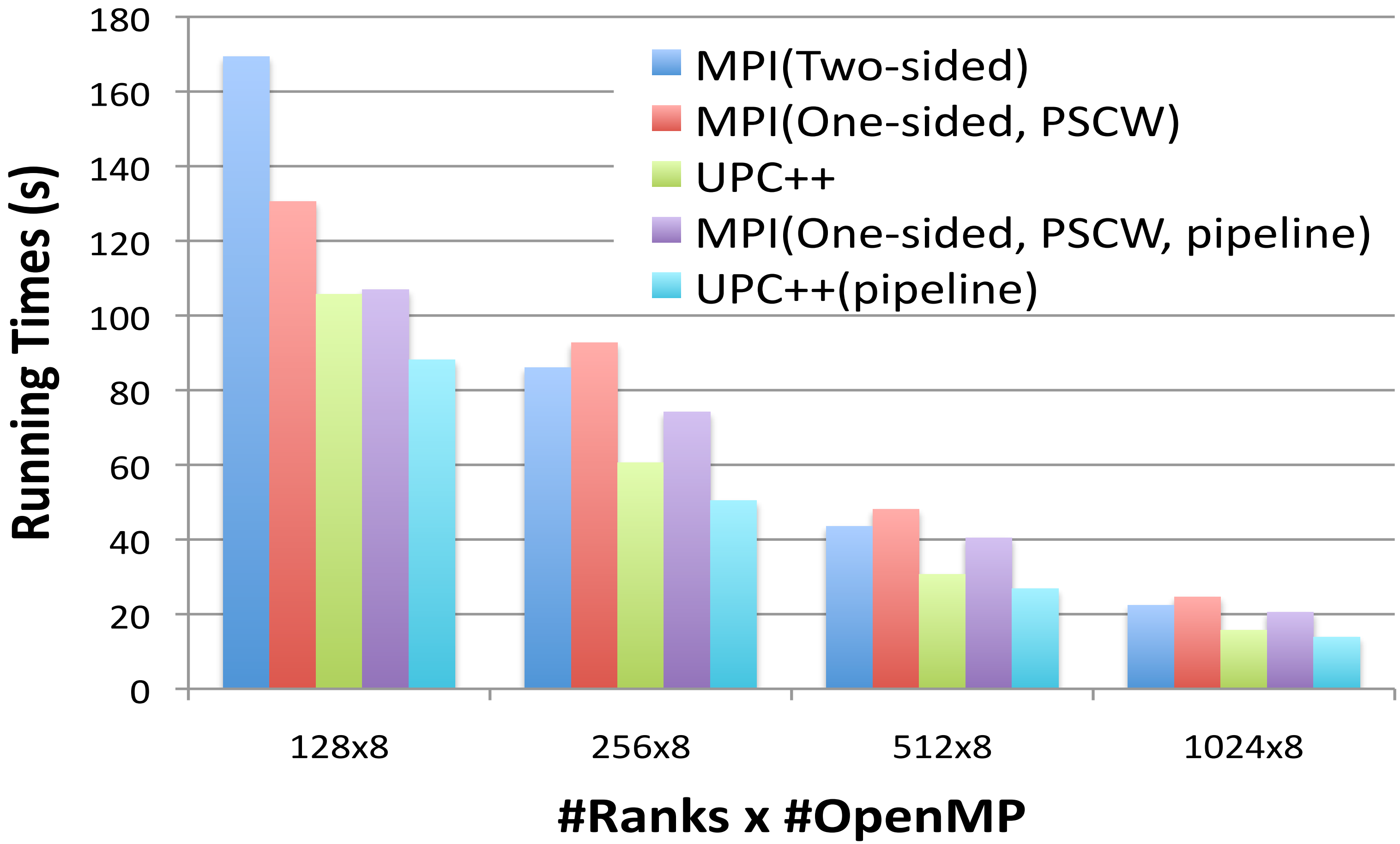
CRD

Time Breakdown (GTC-P, 256x8)

	Packing (s)	Comm (s)	Unpacking (s)	Imbalance (s)	Total (s)
MPI (Two-sided)	17.2	39.0	4.9	24.2	85.3
MPI (One-sided)	17.2	54.3	4.9	16.0	92.5
UPC++	18.8	18.4	7.2	15.1	59.4

	Packing + Comm (s)	Unpacking (s)	Imbalance (s)	Total (s)
MPI (One-sided, pipeline)	58.2	0	15.6	73.7
UPC++ (pipeline)	34.8	0	14.8	49.6

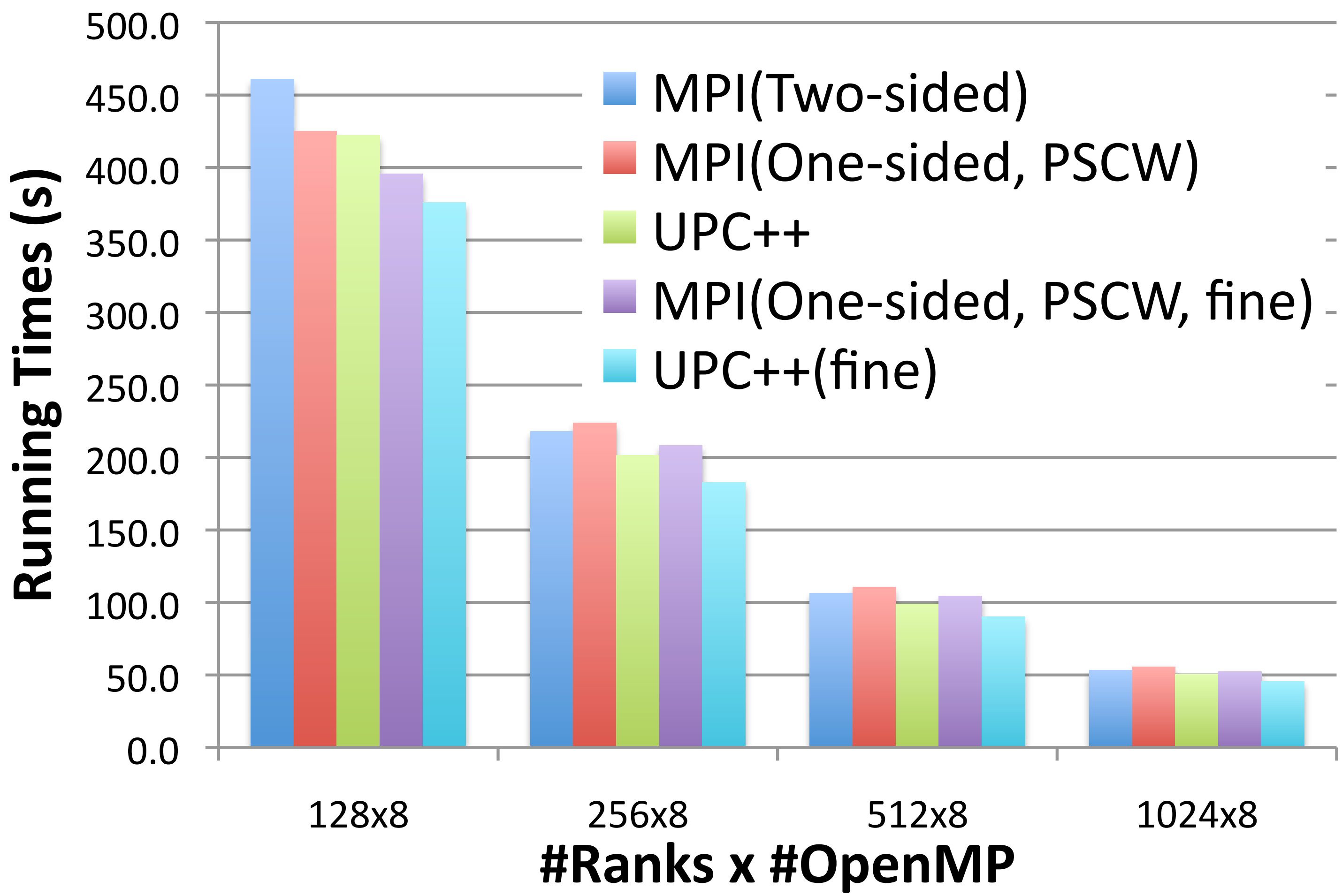
CRD Improved Performance (shift)



- **With message pipelining and no unpacking, MPI one-sided now outperforms MPI two-sided**
- **UPC++ still performs best, 1.6-1.9X better than two-sided**

CRD

Improved Performance (Total)



- UPC++ performs about 20% better than MPI two-sided for total running times

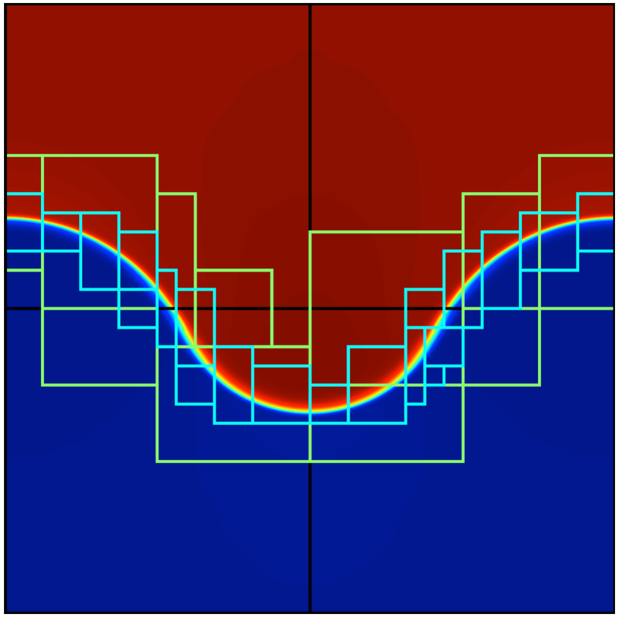
Boxlib

- An adaptive mesh refinement software framework for solving hyperbolic, parabolic and elliptic PDEs on a hierarchy of block-structured grids.
- Finer level composed of union of regular subgrids but the union may be irregular
- Total together more than 300,000 lines of C++, Fortran, and C.
- The main communication routines are packaged in a single file, enabling incremental changes

CRD

Communication Pattern (FillBoundary)

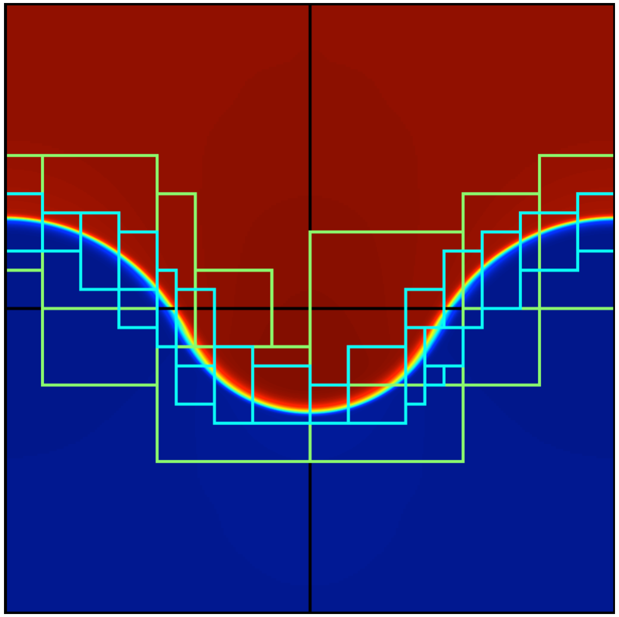
- **Nearest-neighbor Communication pattern needed:**
  - Between levels
  - Neighbors within the same level



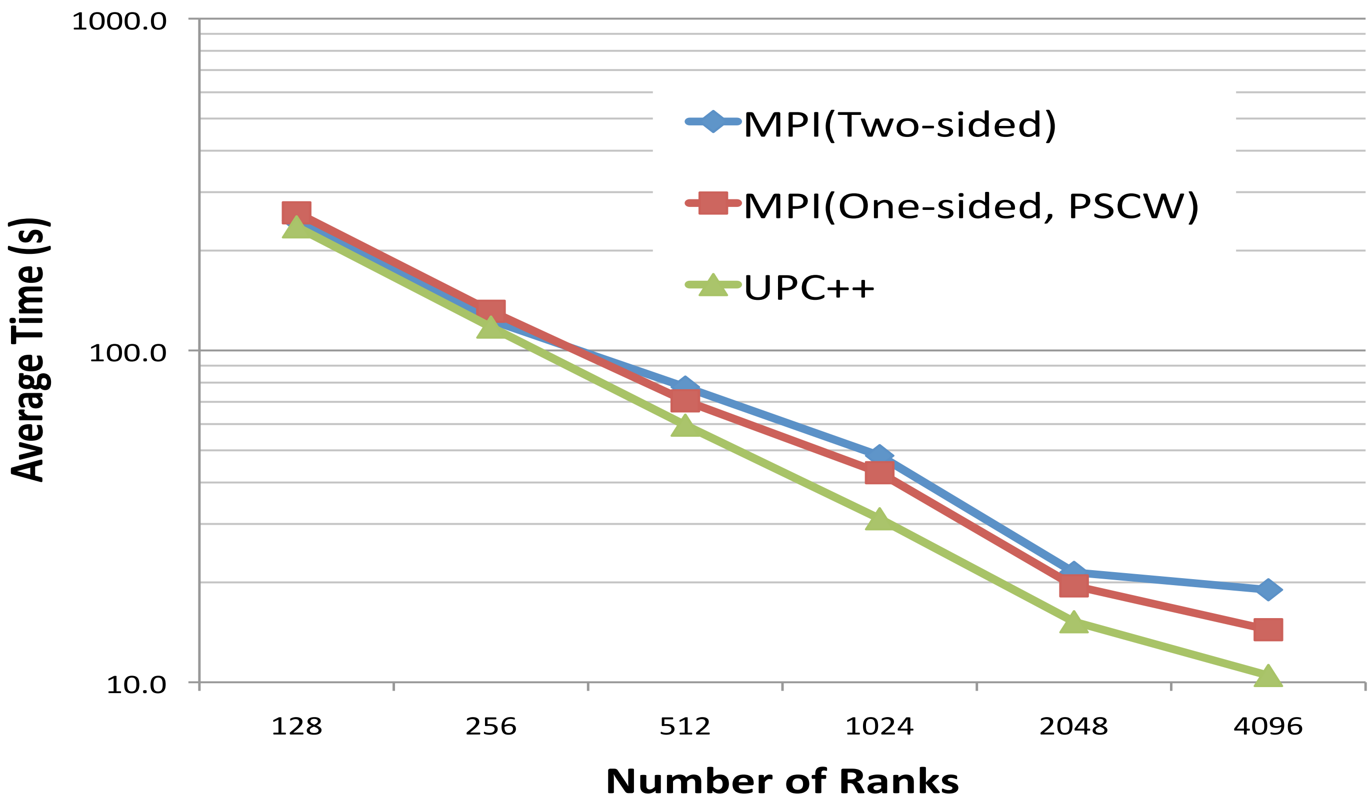
- Calculate the receiving data size and allocate the receive buffer
- For each receiving neighbor, call MPI_Irecv
- For each sending neighbor, pack the data into sending buffer and call MPI_Isend
- Perform local work
- Call MPI_Waitall()
- Unpack the receiver buffer

One-sided Implementation

- Dynamic communication buffer
- MPI Two-sided: not a problem
- MPI One-sided:
  - Using mpi two-sided to exchange info
  - MPI_Put
  - Using MPI_Win_PSCW synchronization
- UPC++
  - Using remote task execution to synchronize
  - Target address can be carried as func parameters
  - Async_copy_and_signal



Performance On Cori (Boxlib)



- **MPI One-sided performs better than two-sided**
- **UPC++ performs best**

CRD

Performance Characteristics (Boxlib)

	Packing	Sending	Waiting	Local	Unpacking	Exchange	Total (s)
MPI (Two-sided)	0.77	0.20	14.29	0.48	1.57	N/A	19.4
MPI (One-sided)	0.93	2.93	7.50	0.55	0.62	2.80	14.40
UPC++	0.69	3.56	3.60	0.49	0.73	N/A	10.50

- MPI Two-sided: dominating by “Waiting”
- MPI One-sided: higher sending time, mainly due to exchange
- UPC++: higher sending time but best waiting time

Summary

- Applied MPI one-sided messages and UPC++ two production codes, GTC-P an Boxlib for which Nearest-Neighbor communication is dominated.
- MPI one-sided delivers close or better (Boxlib, GTC-P with message pipelining) performance than MPI two-sided
- UPC++ delivers best performance
  – Synchronization (using remote task)
  – Better overlap data transfer with local operations