

# Beyond MPI\_Send: What I learned implementing MPI for halo exchange

Ed Bennett  
@QuantumofEd



SA2C Tech Chat, 2018-07-27

Slides: [git.io/fNH3t](https://git.io/fNH3t)

# Background

- Lattice field theory code

# Background

- Lattice field theory code
  - Originally in FORTRAN IV/77

# Background

- Lattice field theory code
  - Originally in FORTRAN IV/77
  - Using `-parallel` scaled well to 4 threads

# Background

- Lattice field theory code
  - Originally in FORTRAN IV/77
  - Using `-parallel` scaled well to 4 threads
  - Refactored to Fortran 90ish

# Background

- Lattice field theory code
  - Originally in FORTRAN IV/77
  - Using `-parallel` scaled well to 4 threads
  - Refactored to Fortran 90ish
  - Indirection replaced with explicit indexing

# Background

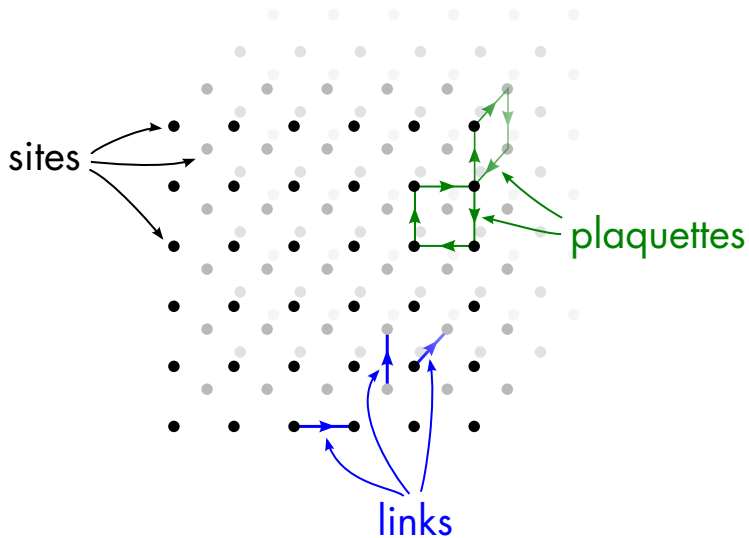
- Lattice field theory code
  - Originally in FORTRAN IV/77
  - Using `-parallel` scaled well to 4 threads
  - Refactored to Fortran 90ish
  - Indirection replaced with explicit indexing
  - All arrays have static sizes

# Background

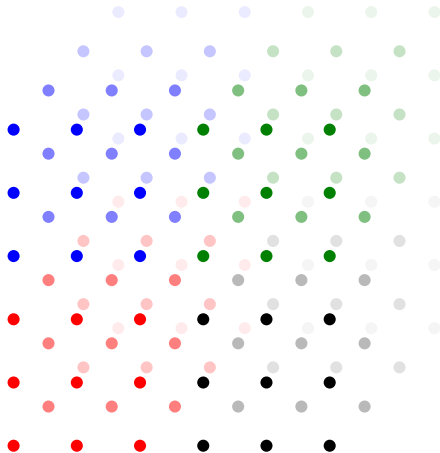
- Lattice field theory code
  - Originally in FORTRAN IV/77
  - Using `-parallel` scaled well to 4 threads
  - Refactored to Fortran 90ish
  - Indirection replaced with explicit indexing
  - All arrays have static sizes
- Three-dimensional problem; 1-3 additional d.o.f.s



# 3D lattice



# Partitioning a 3D lattice



# A quick refresher

- MPI: **M**essage **P**assing **I**nterface

# A quick refresher

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1

# A quick refresher

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model

# A quick refresher

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)

# A quick refresher

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - Communication is explicit

# A quick refresher

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - Communication is explicit
- Initialise program with `MPI_Init`, finish with `MPI_Finalize`



# A quick refresher

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - Communication is explicit
- Initialise program with `MPI_Init`, finish with `MPI_Finalize`
- Point-to-point send and receive with `MPI_Send`, `MPI_Recv`

# A quick refresher

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - Communication is explicit
- Initialise program with `MPI_Init`, finish with `MPI_Finalize`
- Point-to-point send and receive with `MPI_Send`, `MPI_Recv`
- Immediate (non-blocking) versions: `MPI_Isend`, `MPI_Irecv`

# A quick refresher

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - Communication is explicit
- Initialise program with `MPI_Init`, finish with `MPI_Finalize`
- Point-to-point send and receive with `MPI_Send`, `MPI_Recv`
- Immediate (non-blocking) versions: `MPI_Isend`, `MPI_Irecv`
- Point-to-all operations with `MPI_Bcast`

# A quick refresher

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - Communication is explicit
- Initialise program with `MPI_Init`, finish with `MPI_Finalize`
- Point-to-point send and receive with `MPI_Send`, `MPI_Recv`
- Immediate (non-blocking) versions: `MPI_Isend`, `MPI_Irecv`
- Point-to-all operations with `MPI_Bcast`
- Collectives, e.g. `MPI_Reduce`

## TIL #3: use mpi\_f08

- Makes the ierr return variable optional

## TIL #3: use mpi\_f08

- Makes the `ierr` return variable optional
- Everything isn't an integer any more!

## TIL #3: use mpi\_f08

- Makes the ierr return variable optional
- Everything isn't an integer any more!
  - E.g. arguments of MPI\_Wait are now type(MPI\_Request) and type(MPI\_Status)

## TIL #3: use mpi\_f08

- Makes the ierr return variable optional
- Everything isn't an integer any more!
  - E.g. arguments of MPI\_Wait are now type(MPI\_Request) and type(MPI\_Status)
- In-place reductions
  - call MPI\_AllReduce(MPI\_In\_Place, vel2, 1, MPI\_Real, &MPI\_Sum, comm)



## TIL #2: Subarray types

```
subroutine init_single_halo_type_4(direction, position, size4, &
                                datatype, typetarget)
    integer, intent(in) :: direction, position, size4
    type(MPI_Datatype), intent(in) :: datatype
    type(MPI_Datatype), intent(out) :: typetarget
    integer, dimension(4) :: sizes, subsizes, starts

    sizes = (/ ksize_x_l + 2, ksize_y_l + 2, ksize_z_l + 2, size4 /)
    subsizes = (/ ksize_x_l, ksize_y_l, ksize_z_l, size4 /)
    subsizes(direction+1) = 1
    starts = (/ 1, 1, 1, 0 /)
    starts(direction+1) = position

    call MPI_Type_Create_Subarray(4, sizes, subsizes, starts, &
                                MPI_Order_Fortran, datatype, typetarget)
    call MPI_Type_Commit(typetarget)
    return
end subroutine init_single_halo_type_4
```

# TIL #1: MPI-IO

- Avoid channelling all I/O through a single rank/node

# TIL #1: MPI-IO

- Avoid channelling all I/O through a single rank/node
- Higher performance for reading and writing data

# TIL #1: MPI-IO

- Avoid channelling all I/O through a single rank/node
- Higher performance for reading and writing data
- Works really well with subarray types

# TIL #1: MPI-IO

- Avoid channelling all I/O through a single rank/node
- Higher performance for reading and writing data
- Works really well with subarray types

# TIL #1: MPI-IO

- Avoid channelling all I/O through a single rank/node
- Higher performance for reading and writing data
- Works really well with subarray types

```
call MPI_File_Open(comm, 'con', MPI_Mode_Rdonly, &
                    MPI_Info_Null, mpi_fh)
call MPI_File_Set_View(mpi_fh, 0_8, MPI_Real, mpiio_type, &
                        "native", MPI_Info_Null)
call MPI_File_Read_All(mpi_fh, theta, &
                        3 * ksize_x_l * ksize_y_l * ksize_t_l, &
                        MPI_Real, status)
call MPI_File_Close(mpi_fh)
```

## TIL #4: Cartesian communicators

- Give some structure to the set of processes

## TIL #4: Cartesian communicators

- Give some structure to the set of processes
- Can be created with periodic boundaries

```
call MPI_cart_create(MPI_COMM_WORLD, 3, &  
                    (/ NP_X, NP_Y, NP_T /), &  
                    (/ .true., .true., .true. /), &  
                    .true., comm)
```



## TIL #4: Cartesian communicators

- Give some structure to the set of processes
- Can be created with periodic boundaries

```
call MPI_cart_create(MPI_COMM_WORLD, 3, &  
                    (/ NP_X, NP_Y, NP_T /), &  
                    (/ .true., .true., .true. /), &  
                    .true., comm)
```

- Can access processes relative to current one:

```
call MPI_Cart_Shift(comm, 2, 1, ip_tdn, ip_tup)
```

## TIL #4: Cartesian communicators

- Give some structure to the set of processes
- Can be created with periodic boundaries

```
call MPI_cart_create(MPI_COMM_WORLD, 3, &  
                    (/ NP_X, NP_Y, NP_T /), &  
                    (/ .true., .true., .true. /), &  
                    .true., comm)
```

- Can access processes relative to current one:

```
call MPI_Cart_Shift(comm, 2, 1, ip_tdn, ip_tup)
```

- Gives index of processes in both directions

## TIL #5: Persistent MPI communications

- Each MPI\_Send/MPI\_Recv pair has an overhead

## TIL #5: Persistent MPI communications

- Each MPI\_Send/MPI\_Recv pair has an overhead
- For a tight loop, this wastes time

## TIL #5: Persistent MPI communications

- Each `MPI_Send/MPI_Recv` pair has an overhead
- For a tight loop, this wastes time
- Instead, use `MPI_Send_Init/MPI_Recv_Init` outside the loop

## TIL #5: Persistent MPI communications

- Each MPI\_Send/MPI\_Recv pair has an overhead
- For a tight loop, this wastes time
- Instead, use MPI\_Send\_Init/MPI\_Recv\_Init outside the loop
- MPI\_Start/MPI\_StartAll inside

## TIL #5: Persistent MPI communications

- Each MPI\_Send/MPI\_Recv pair has an overhead
- For a tight loop, this wastes time
- Instead, use MPI\_Send\_Init/MPI\_Recv\_Init outside the loop
- MPI\_Start/MPI\_StartAll inside
- Also need MPI\_Wait/MPI\_WaitAll

## TIL #5: Persistent MPI communications

- Each MPI\_Send/MPI\_Recv pair has an overhead
- For a tight loop, this wastes time
- Instead, use MPI\_Send\_Init/MPI\_Recv\_Init outside the loop
- MPI\_Start/MPI\_StartAll inside
- Also need MPI\_Wait/MPI\_WaitAll
- Collectives planned for MPI 3.2, e.g. MPI\_AllReduce\_Init



# Step 1: Halos

- Add a 1-site border around all three dimensions

# Step 1: Halos

- Add a 1-site border around all three dimensions
  - Only if  $\phi_{i+1,j,k}$  is needed

# Step 1: Halos

- Add a 1-site border around all three dimensions
  - Only if  $\phi_{i+1,j,k}$  is needed
- Store contents of  $\phi_{1,j,k}$  in  $\phi_{n_x+1,j,k}$ ,  $\phi_{n_x,j,k}$  in  $\phi_{0,j,k}$ , etc.

# Step 1: Halos

- Add a 1-site border around all three dimensions
  - Only if  $\phi_{i+1,j,k}$  is needed
- Store contents of  $\phi_{1,j,k}$  in  $\phi_{n_x+1,j,k}$ ,  $\phi_{n_x,j,k}$  in  $\phi_{0,j,k}$ , etc.
- Functions to update halos

# Step 1: Halos

- Add a 1-site border around all three dimensions
  - Only if  $\phi_{i+1,j,k}$  is needed
- Store contents of  $\phi_{1,j,k}$  in  $\phi_{n_x+1,j,k}$ ,  $\phi_{n_x,j,k}$  in  $\phi_{0,j,k}$ , etc.
- Functions to update halos
  - Use a module – `comms` – for this

# Step 1: Halos

- Add a 1-site border around all three dimensions
  - Only if  $\phi_{i+1,j,k}$  is needed
- Store contents of  $\phi_{1,j,k}$  in  $\phi_{n_x+1,j,k}$ ,  $\phi_{n_x,j,k}$  in  $\phi_{0,j,k}$ , etc.
- Functions to update halos
  - Use a module – `comms` – for this
  - Can then swap out for MPI version later

# Step 1: Halos

- Add a 1-site border around all three dimensions
  - Only if  $\phi_{i+1,j,k}$  is needed
- Store contents of  $\phi_{1,j,k}$  in  $\phi_{n_x+1,j,k}$ ,  $\phi_{n_x,j,k}$  in  $\phi_{0,j,k}$ , etc.
- Functions to update halos
  - Use a module – `comms` – for this
  - Can then swap out for MPI version later
- Test each function still gives same results as previously

## Step 2: I/O

- Reimplement configuration read and write using MPI-IO



## Step 2: I/O

- Reimplement configuration read and write using MPI-IO
- Existing implementation saved RNG state in configuration

## Step 2: I/O

- Reimplement configuration read and write using MPI-IO
- Existing implementation saved RNG state in configuration
  - Store state from rank 0

## Step 2: I/O

- Reimplement configuration read and write using MPI-IO
- Existing implementation saved RNG state in configuration
  - Store state from rank 0
  - Reseed other ranks as `state + rank`

## Step 2: I/O

- Reimplement configuration read and write using MPI-IO
- Existing implementation saved RNG state in configuration
  - Store state from rank 0
  - Reseed other ranks as `state + rank`
- Check that read and write still give same results

## Step 3: Subarray type initialisation

- Add MPI initialisation function to set up MPI subarray types

## Step 3: Subarray type initialisation

- Add MPI initialisation function to set up MPI subarray types
- Separate types for each array shape, each boundary, each direction, each datatype

## Step 3: Subarray type initialisation

- Add MPI initialisation function to set up MPI subarray types
- Separate types for each array shape, each boundary, each direction, each datatype
  - 4D (real and complex), 5D, 6D

## Step 3: Subarray type initialisation

- Add MPI initialisation function to set up MPI subarray types
- Separate types for each array shape, each boundary, each direction, each datatype
  - 4D (real and complex), 5D, 6D
  - $x, y, t$ ; up, down; send, receive



## Step 3: Subarray type initialisation

- Add MPI initialisation function to set up MPI subarray types
- Separate types for each array shape, each boundary, each direction, each datatype
  - 4D (real and complex), 5D, 6D
  - $x, y, t$ ; up, down; send, receive
  - Varying sizes of 4th, 5th, 6th dimensions

## Step 3: Subarray type initialisation

- Add MPI initialisation function to set up MPI subarray types
- Separate types for each array shape, each boundary, each direction, each datatype
  - 4D (real and complex), 5D, 6D
  - $x, y, t$ ; up, down; send, receive
  - Varying sizes of 4th, 5th, 6th dimensions
  - Use arrays for this

## Step 3: Subarray type initialisation

- Add MPI initialisation function to set up MPI subarray types
- Separate types for each array shape, each boundary, each direction, each datatype
  - 4D (real and complex), 5D, 6D
  - $x, y, t$ ; up, down; send, receive
  - Varying sizes of 4th, 5th, 6th dimensions
  - Use arrays for this
  - Populate only required array elements

## Step 3: Subarray type initialisation

- Add MPI initialisation function to set up MPI subarray types
- Separate types for each array shape, each boundary, each direction, each datatype
  - 4D (real and complex), 5D, 6D
  - $x, y, t$ ; up, down; send, receive
  - Varying sizes of 4th, 5th, 6th dimensions
  - Use arrays for this
  - Populate only required array elements
  - e.g. size6 needs to be 1, 12, and 25; create `halo_6_xdn_send(4:4, 1:25)`; populate elements 1, 12, 25

## Step 3: Subarray type initialisation

- Add MPI initialisation function to set up MPI subarray types
- Separate types for each array shape, each boundary, each direction, each datatype
  - 4D (real and complex), 5D, 6D
  - $x, y, t$ ; up, down; send, receive
  - Varying sizes of 4th, 5th, 6th dimensions
  - Use arrays for this
  - Populate only required array elements
  - e.g. size6 needs to be 1, 12, and 25; create `halo_6_xdn_send(4:4, 1:25)`; populate elements 1, 12, 25
- Include the type for MPI-IO here

## Step 4: Halo communication

- For each dimensionality, a function to do both `MPI_Isend` and `MPI_Irecv`

## Step 4: Halo communication

- For each dimensionality, a function to do both `MPI_Isend` and `MPI_Irecv`
- Also takes in an array of `MPI_Requests` and fills it

## Step 4: Halo communication

- For each dimensionality, a function to do both `MPI_Isend` and `MPI_Irecv`
- Also takes in an array of `MPI_Requests` and fills it
- A separate function to complete updates in any dimensionality



## Step 4: Halo communication

- For each dimensionality, a function to do both `MPI_Isend` and `MPI_Irecv`
- Also takes in an array of `MPI_Requests` and fills it
- A separate function to complete updates in any dimensionality
  - Only takes in an array of 12 `MPI_Request` objects

## Step 4: Halo communication

- For each dimensionality, a function to do both `MPI_Isend` and `MPI_Irecv`
- Also takes in an array of `MPI_Request`s and fills it
- A separate function to complete updates in any dimensionality
  - Only takes in an array of 12 `MPI_Request` objects
- Unit test

## Step 5: Parallelise most functions

- Replace global with local dimensions

## Step 5: Parallelise most functions

- Replace global with local dimensions
- Before any function call relying on halos:

## Step 5: Parallelise most functions

- Replace global with local dimensions
- Before any function call relying on halos:
  - Work out where data last edited

## Step 5: Parallelise most functions

- Replace global with local dimensions
- Before any function call relying on halos:
  - Work out where data last edited
  - Add a halo update start at that point

## Step 5: Parallelise most functions

- Replace global with local dimensions
- Before any function call relying on halos:
  - Work out where data last edited
  - Add a halo update start at that point
  - Complete update in time for its use

## Step 5: Parallelise most functions

- Replace global with local dimensions
- Before any function call relying on halos:
  - Work out where data last edited
  - Add a halo update start at that point
  - Complete update in time for its use
- Any collective operation gets an `MPI\_AllReduce`



## Step 5: Parallelise most functions

- Replace global with local dimensions
- Before any function call relying on halos:
  - Work out where data last edited
  - Add a halo update start at that point
  - Complete update in time for its use
- Any collective operation gets an `MPI\_AllReduce`
- All MPI calls are wrapped with `#ifdef MPI`

## Step 5: Parallelise most functions

- Replace global with local dimensions
- Before any function call relying on halos:
  - Work out where data last edited
  - Add a halo update start at that point
  - Complete update in time for its use
- Any collective operation gets an `MPI\_AllReduce`
- All MPI calls are wrapped with `#ifdef MPI`
- Check regression tests

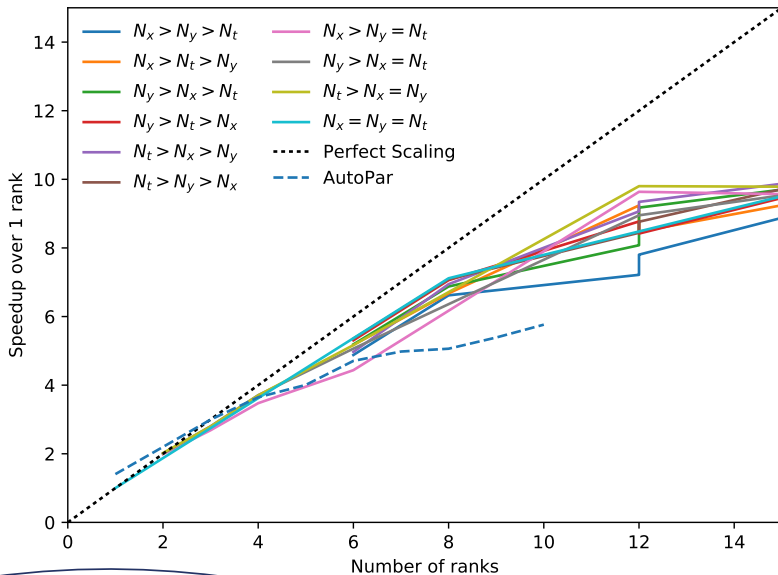
## Step 6: Loose ends

- Correlation functions: lots of extra bookkeeping

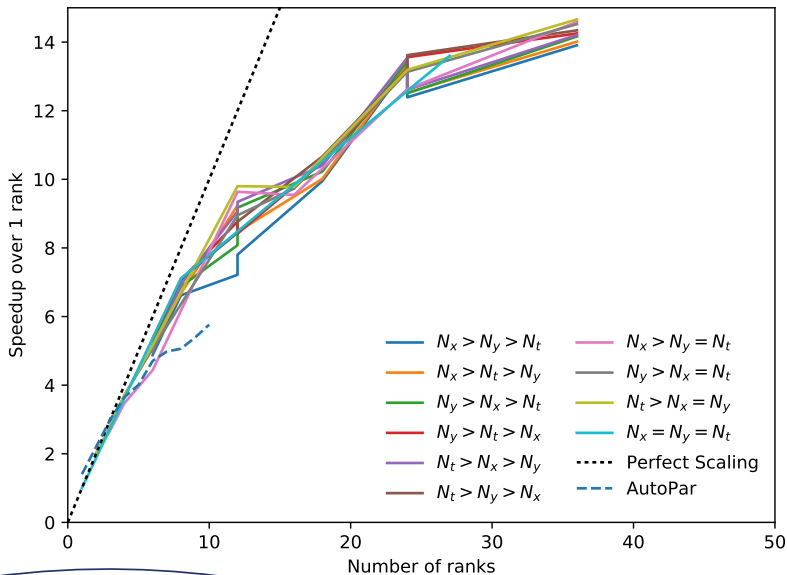
## Step 6: Loose ends

- Correlation functions: lots of extra bookkeeping
- Reading input parameters: do on rank 0 and broadcast

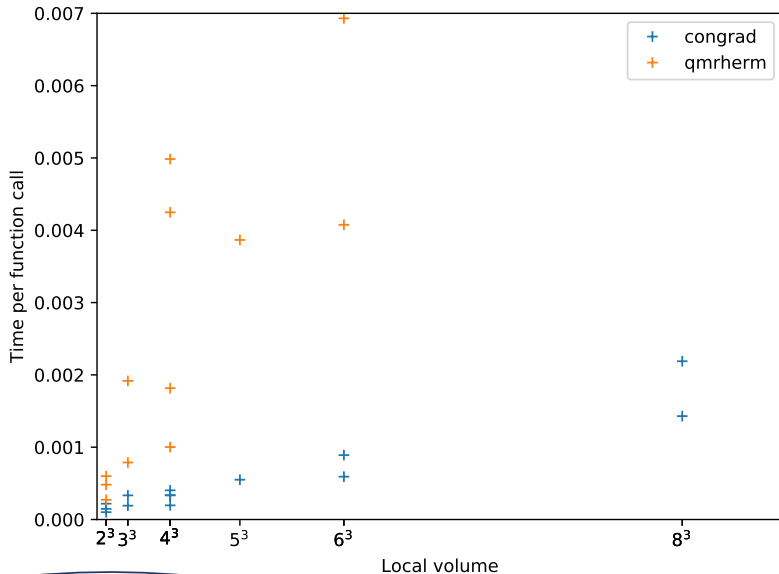
# Performance on Hawk



# Performance on Hawk



# Weak and strong scaling of a single operation



# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?



# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...
- Noncontiguous send/receive regions?

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...
- Noncontiguous send/receive regions?
  - ITAC shows MPI\_Isend takes time, which is weird

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...
- Noncontiguous send/receive regions?
  - ITAC shows MPI\_Isend takes time, which is weird
  - Possibly due to data rearrangement

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...
- Noncontiguous send/receive regions?
  - ITAC shows MPI\_Isend takes time, which is weird
  - Possibly due to data rearrangement
  - Need to reintroduce indirection to fix this

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...
- Noncontiguous send/receive regions?
  - ITAC shows MPI\_Isend takes time, which is weird
  - Possibly due to data rearrangement
  - Need to reintroduce indirection to fix this
- Insufficient hiding of communication



# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...
- Noncontiguous send/receive regions?
  - ITAC shows MPI\_Isend takes time, which is weird
  - Possibly due to data rearrangement
  - Need to reintroduce indirection to fix this
- Insufficient hiding of communication
  - Delay calls to MPI\_Wait until relevant work is reached

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...
- Noncontiguous send/receive regions?
  - ITAC shows MPI\_Isend takes time, which is weird
  - Possibly due to data rearrangement
  - Need to reintroduce indirection to fix this
- Insufficient hiding of communication
  - Delay calls to MPI\_Wait until relevant work is reached
  - Initial tests not promising - but only 10% of work hidden

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...
- Noncontiguous send/receive regions?
  - ITAC shows MPI\_Isend takes time, which is weird
  - Possibly due to data rearrangement
  - Need to reintroduce indirection to fix this
- Insufficient hiding of communication
  - Delay calls to MPI\_Wait until relevant work is reached
  - Initial tests not promising - but only 10% of work hidden
  - Hiding more requires restructuring loops

# Why doesn't it scale well past 8–12 processes?

- Try persistent send/receive?
  - Up to 20% speedup on Broadwell+OPA
  - No improvement on Skylake+Mellanox
  - Persistent AllReduce has to wait...
- Noncontiguous send/receive regions?
  - ITAC shows MPI\_Isend takes time, which is weird
  - Possibly due to data rearrangement
  - Need to reintroduce indirection to fix this
- Insufficient hiding of communication
  - Delay calls to MPI\_Wait until relevant work is reached
  - Initial tests not promising - but only 10% of work hidden
  - Hiding more requires restructuring loops
  - Significantly more mess

# Thanks for listening!



Slides: [git.io/fNH3t](https://git.io/fNH3t)