Beyond `MPI_Send`: What I learned implementing MPI for halo exchange

Ed Bennett
@QuantumofEd

Swansea University
Prifysgol Abertawe
@SwanseaUni

SUPERCOMPUTING WALES
UWCHGYFRIFIADURA CYMRU
@SuperCompWales

SA2C
@sa2c_swansea

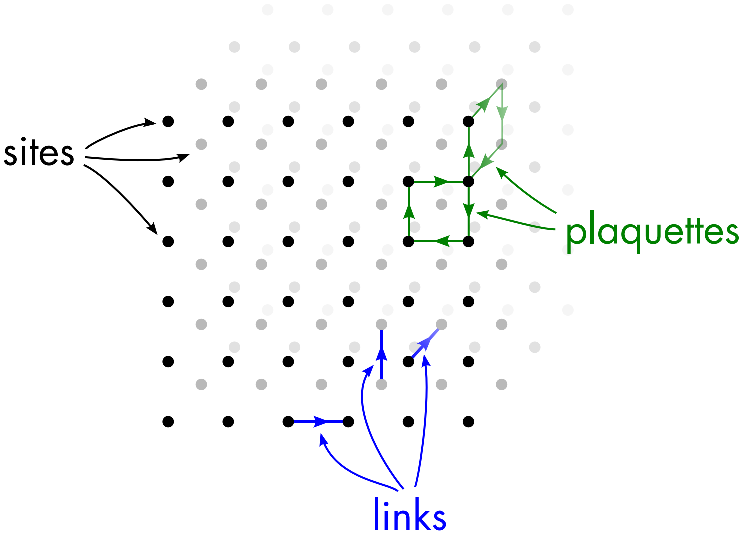
SA2C Tech Chat, 2018-07-27

Slides: git.io/fNH3t

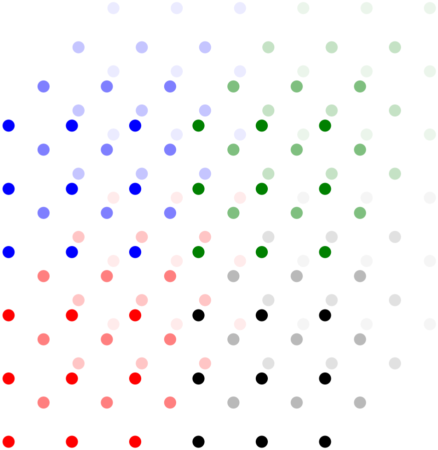
Background

- 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
- 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`
- **I**mmediate (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`
- 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 = (/ ksizex_l + 2, ksizey_l + 2, ksizet_l + 2, size4 /)
  subsizes = (/ ksizex_l, ksizey_l, ksizet_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
- 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 * ksizex_l * ksizey_l * ksizet_l, &
                       MPI_Real, status)
call MPI_File_Close(mpi_fh)
```

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
- 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
  - 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
- 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
- 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`
- 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
- Unit test

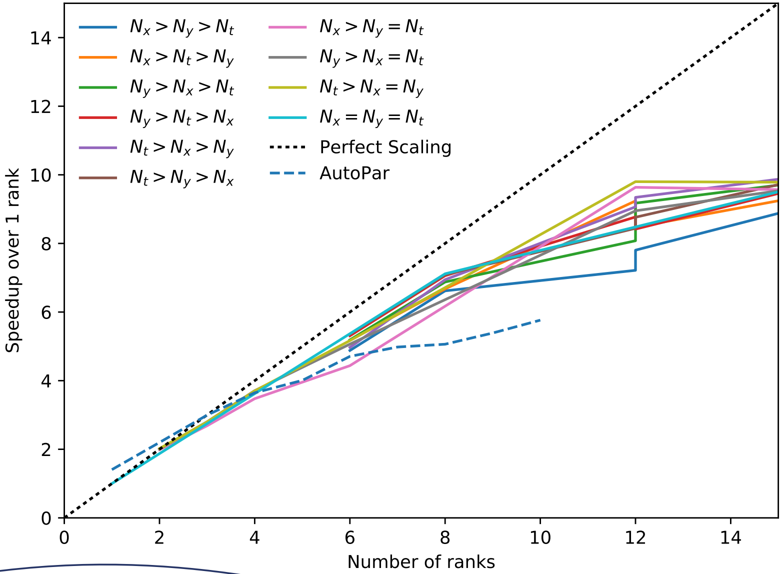
Step 5: Parallelise most functions

- 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

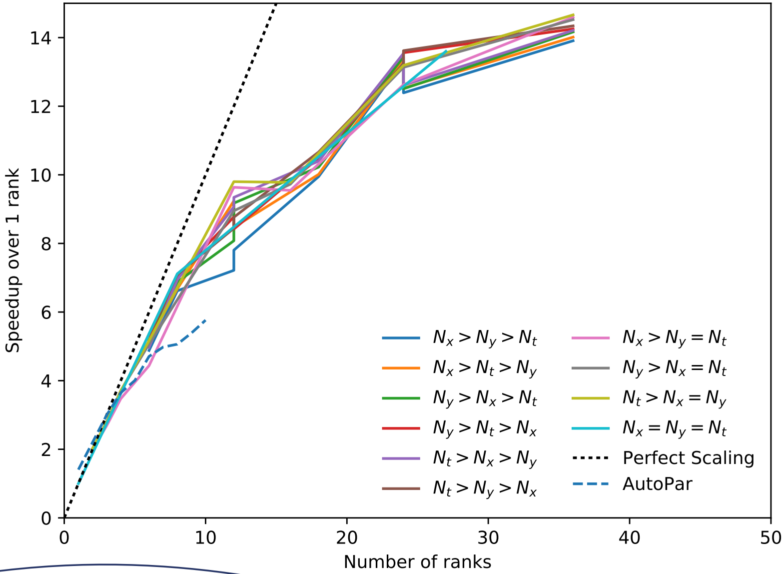
- Correlation functions: lots of extra bookkeeping
- Reading input parameters: do on rank 0 and broadcast

Performance on Hawk

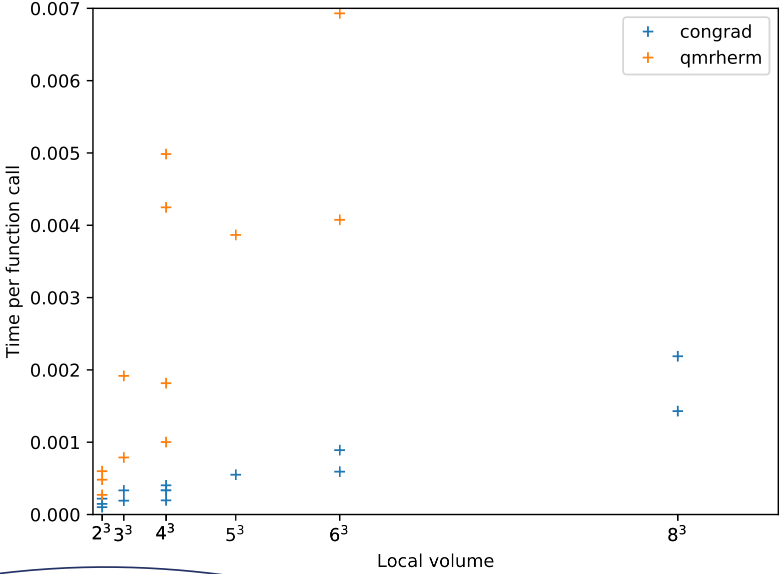


Slides: git.io/fNH3t

Performance on Hawk



Weak and strong scaling of a single operation



Slides: git.io/fNH3t

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