Towards a Parallel Coupled Multi-Scale Model of Magnetic Reconnection

Michal Charemza   Tony Arber

centre for fusion, space and astrophysics
Centre for Scientific Computing
THE UNIVERSITY OF WARWICK

May 26, 2007

Contents

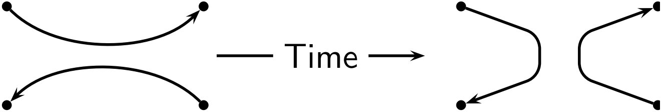
Contents

Contents

Contents

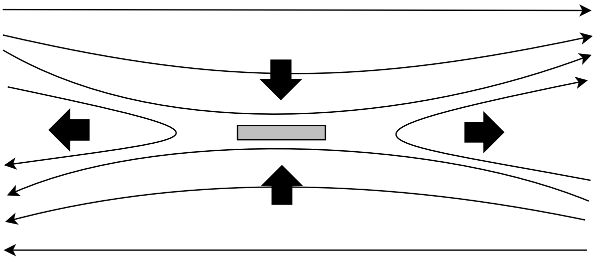
Contents

Magnetic Reconnection



- Dependent on large scale boundary conditions
- Affected by small scale non-fluid effects

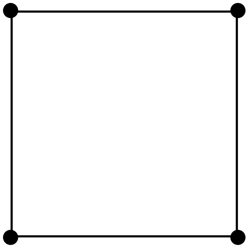
Magnetic Reconnection



- Dependent on large scale boundary conditions
- Affected by small scale non-fluid effects

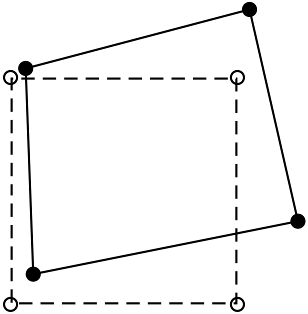
Start of time step

- At time step n, solution known on Eulerian grid
- Solution know from from previous step



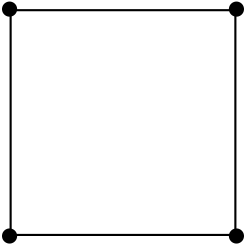
After Lagrangian Step

- At time step $n + 1$, solution known on Lagrangian grid.
- Some numerical time dependent method used.



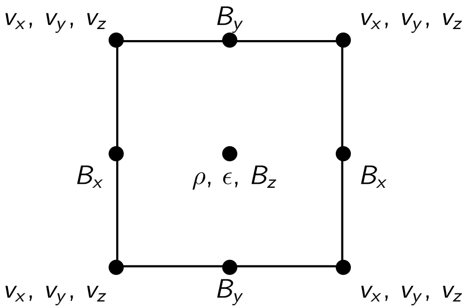
After Remap Step

- At time step $n+1$, solution known on Eulerian grid
- Geometrical method used



Lare2D

Solves resistive and Hall MHD equations



What AMR?

- Adaptive Mesh Refinement
- Technique for extending a numerical method for solving equations, using different grid resolution is different areas of the domain
- Higher grid resolution only where, and when, desired
- Higher resolution typically in regions of high change of variables

Advantages of AMR?

- Speed: Faster than equivalent non-AMR code
- Memory: Less memory used than equivalent non-AMR code.

Disadvantages of AMR

- Complex code
- Computational time used to communicate between refinement levels
- Computational time used to navigate data structures
- Can be more difficult to parallelise

Disadvantages of AMR

- Complex code
- Computational time used to communicate between refinement levels
- Computational time used to navigate data structures
- Can be more difficult to parallelise

Disadvantages of AMR

- Complex code
- Computational time used to communicate between refinement levels
- Computational time used to navigate data structures
- Can be more difficult to parallelise

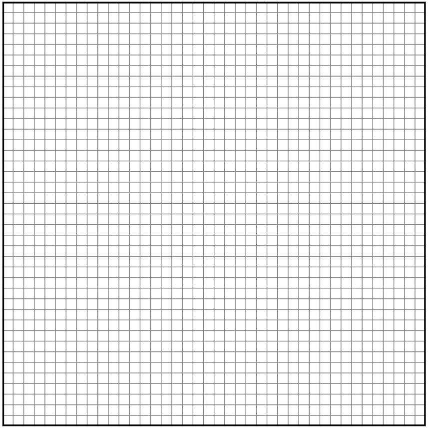
Disadvantages of AMR

- Complex code
- Computational time used to communicate between refinement levels
- Computational time used to navigate data structures
- Can be more difficult to parallelise

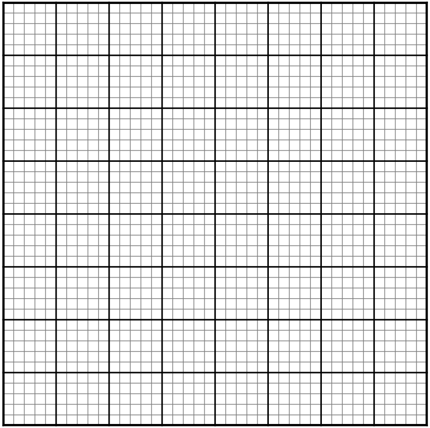
Disadvantages of AMR

- Complex code
- Computational time used to communicate between refinement levels
- Computational time used to navigate data structures
- Can be more difficult to parallelise

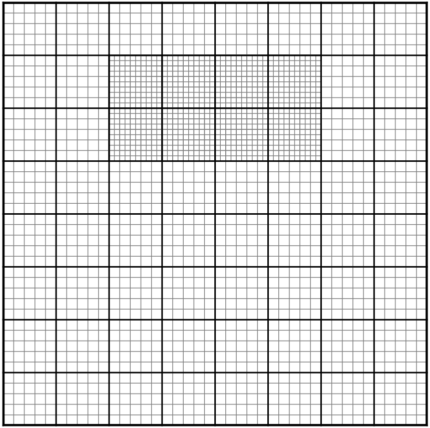
Typical Lare2D computational domain



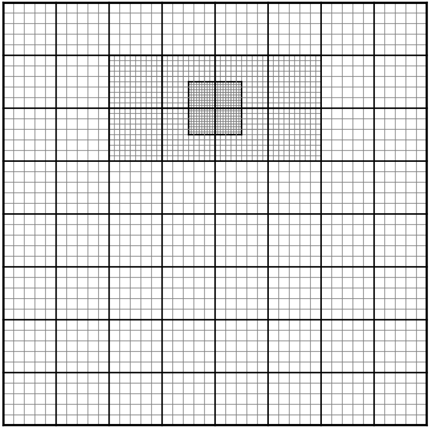
AMR Larma computational domain



AMR `Larma` computational domain



AMR Larma computational domain



Orszag-Tang Vortex

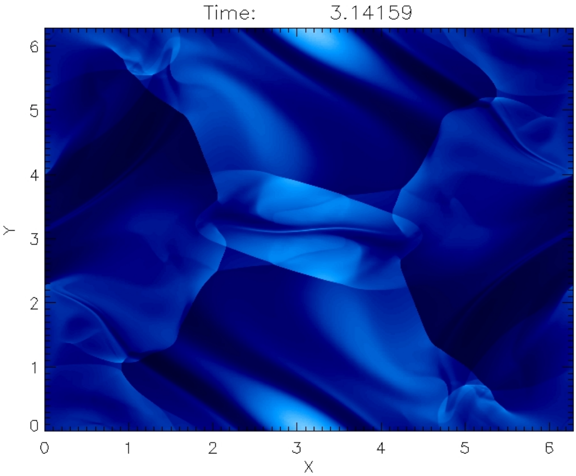
- Initial Conditions $\rho = 25/9$, $p = 5/3$

$$v_x = -\sin y \qquad B_x = -\sin y$$
$$v_y = \sin x \qquad B_y = \sin 2x$$
$$v_z = 0 \qquad B_z = 0$$

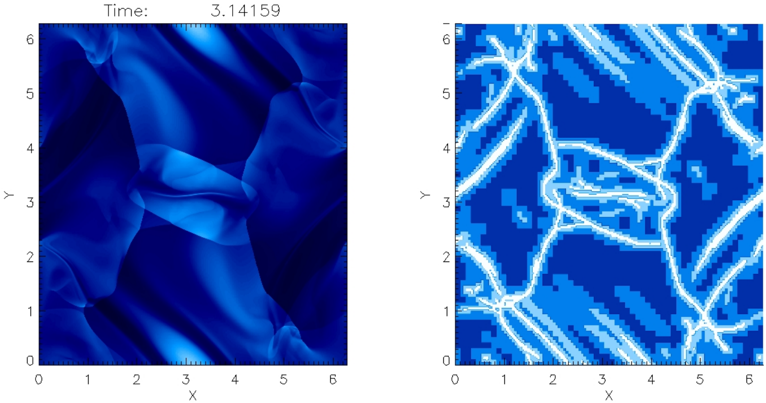
$0 \leq x, y \leq 2\pi$, time from 0 to π, periodic boundary conditions

- Simple intial conditions lead to shocks

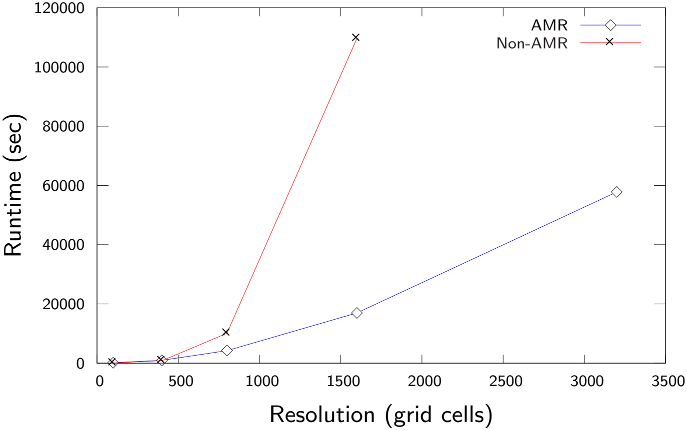
Test Results



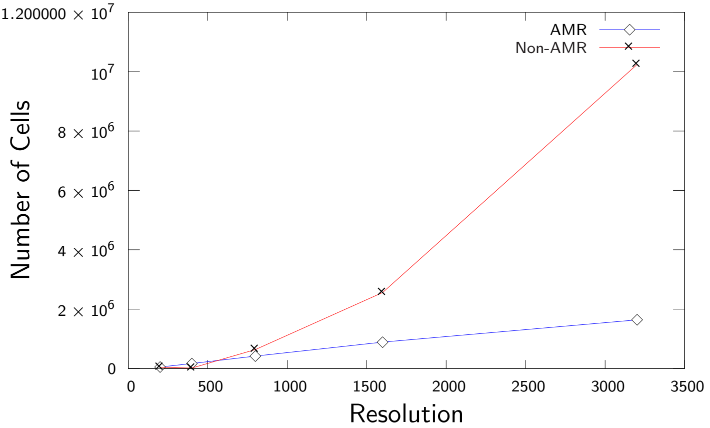
AMR Patch Placement



Speedup of Orszag-Tang Vortex



Memory usage of Orszag-Tang Vortex



Problems

- Choice of architecture (MPI)
- Load balancing
- Communication time
- What and when to communicate
- Processing the results

Problems

- Choice of architecture (MPI)
- Load balancing
- Communication time
- What and when to communicate
- Processing the results

Problems

- Choice of architecture (MPI)
- Load balancing
- Communication time
- What and when to communicate
- Processing the results

Problems

- Choice of architecture (MPI)
- Load balancing
- Communication time
- What and when to communicate
- Processing the results

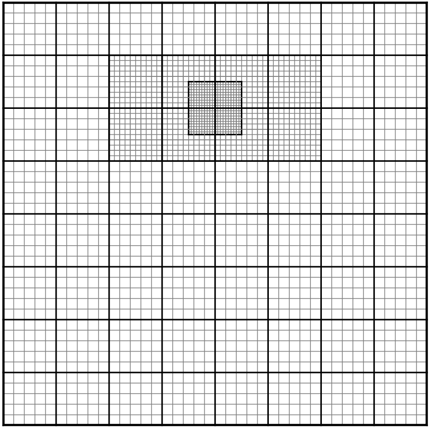
Problems

- Choice of architecture (MPI)
- Load balancing
- Communication time
- What and when to communicate
- Processing the results

Problems

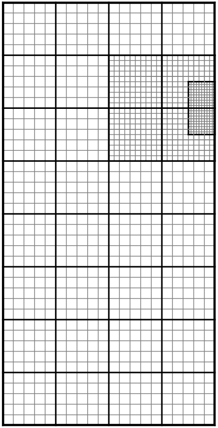
- Choice of architecture (MPI)
- Load balancing
- Communication time
- What and when to communicate
- Processing the results

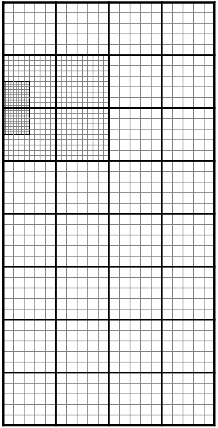
Domain decomposition with Ghost Patches



Domain decomposition with Ghost Patches

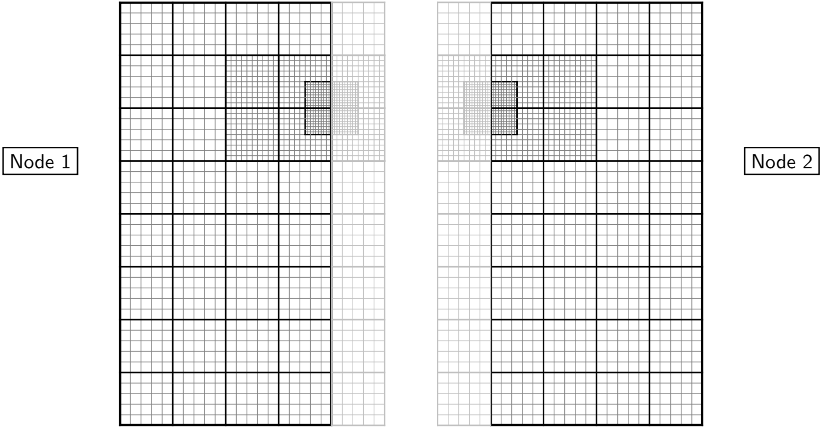
Node 1





Node 2

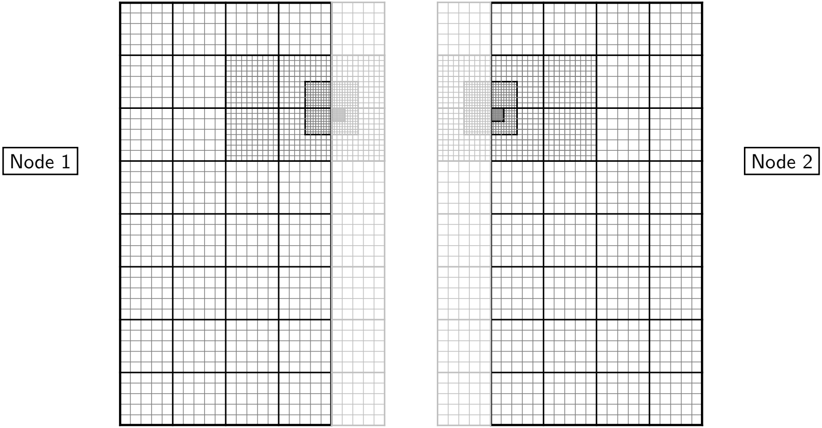
Domain decomposition with Ghost Patches



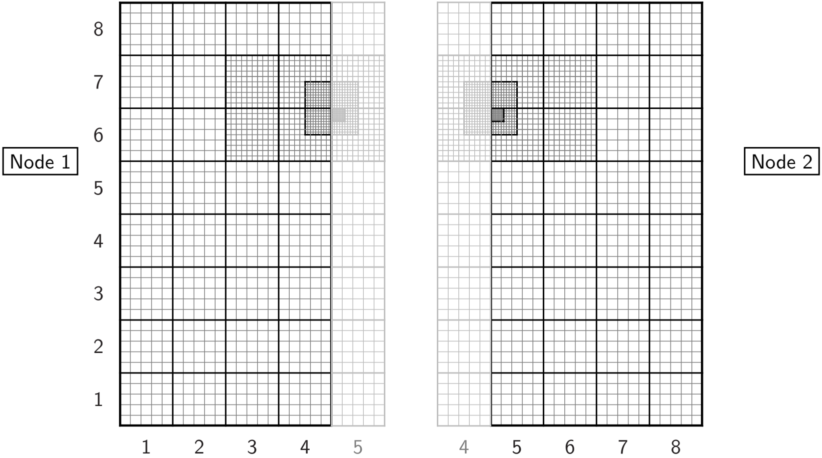
Ghost Patches: Consequences

- Majority of inter-patch communication code can be reused
- Need some way to tell for nodes to tell each other to create (and remove) patches
- Potentially many messages per time step sent to update ghost patches. High MPI latency per message $\implies$ lots of time waiting for messages to complete.

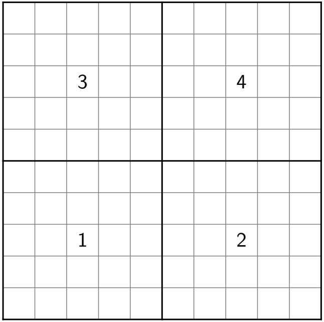
AMR Coordinates



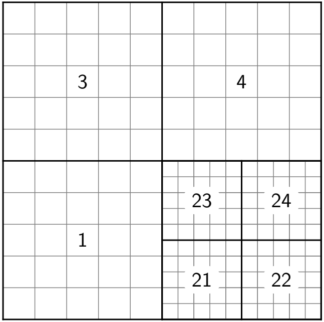
AMR Coordinates



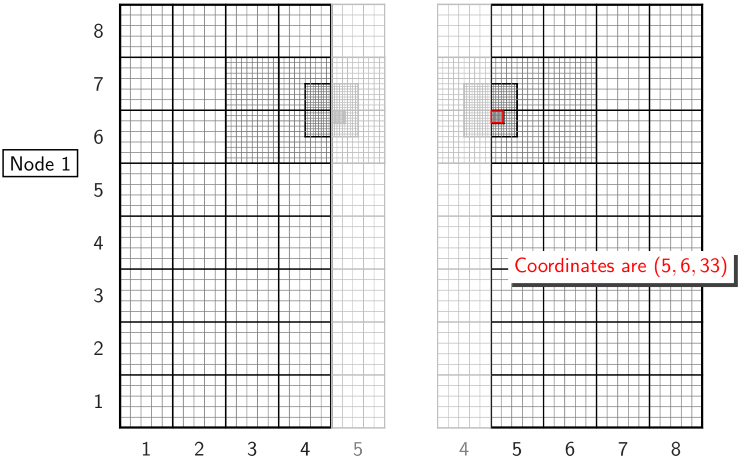
AMR Coordinates



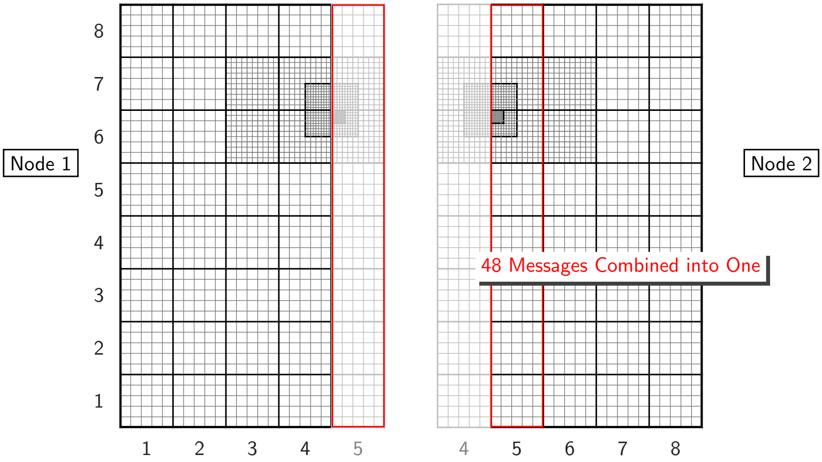
AMR Coordinates



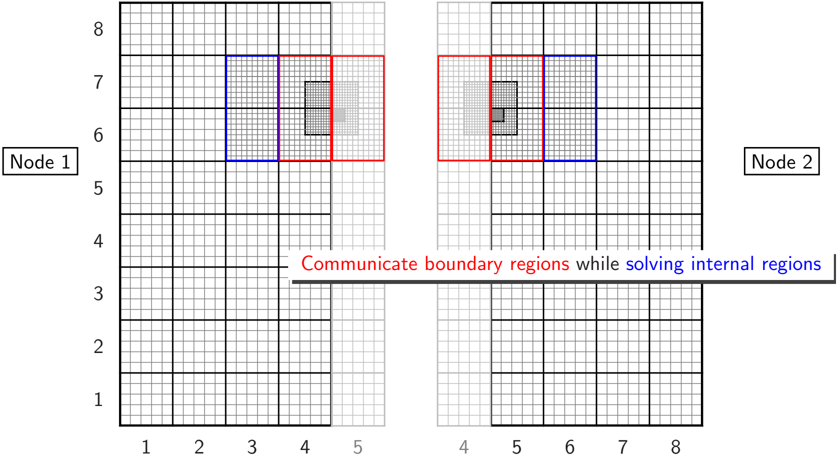
AMR Coordinates



MPI_Struct to combine messages



Non Blocking Communication



Current State

- Implemented: Ghost patches, AMR Coordinates, To do list
- Implemented: MPI Message combining
- Non blocking communication - none but patch structure should make this possible
- Load balancing - none. Different domain sizes possible. Other methods could require more of a re-write.

Current State

- Implemented: Ghost patches, AMR Coordinates, To do list
- Implemented: MPI Message combining
- Non blocking communication - none but patch structure should make this possible
- Load balancing - none. Different domain sizes possible. Other methods could require more of a re-write.

Current State

- Implemented: Ghost patches, AMR Coordinates, To do list
- Implemented: MPI Message combining
- Non blocking communication - none but patch structure should make this possible
- Load balancing - none. Different domain sizes possible. Other methods could require more of a re-write.

Current State

- Implemented: Ghost patches, AMR Coordinates, To do list
- Implemented: MPI Message combining
- Non blocking communication - none but patch structure should make this possible
- Load balancing - none. Different domain sizes possible. Other methods could require more of a re-write.

Efficiency Metric

- Efficiency of run on N processors

$$= \frac{\text{Runtime on 1 processor}}{N \times \text{ Runtime on } N \text{ processors}}$$

Load-imbalance Metric

- Pattern of communication/processing is same on *all* processes
- Time lost due to load imbalance for each computational block

$$= \max_{\text{processors } p} (\text{compute time for } p - \text{average compute time})$$

- Use timing calls in code
- Averages calculated at end of run to minimize communication

Load-imbalance Metric

- Pattern of communication/processing is same on *all* processes
- Time lost due to load imbalance for each computational block

$$= \max_{\text{processors } p} (\text{compute time for } p - \text{average compute time})$$

- Use timing calls in code
- Averages calculated at end of run to minimize communication

Load-imbalance Metric

- Pattern of communication/processing is same on *all* processes
- Time lost due to load imbalance for each computational block

$$= \max_{\text{processors } p} (\text{compute time for } p - \text{average compute time})$$

- Use timing calls in code
- Averages calculated at end of run to minimize communication

Load-imbalance Metric

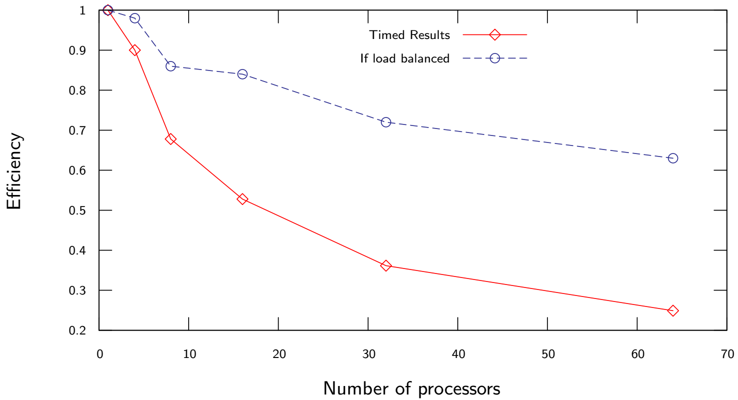
- Pattern of communication/processing is same on *all* processes
- Time lost due to load imbalance for each computational block

$$= \max_{\text{processors } p} (\text{compute time for } p - \text{average compute time})$$

- Use timing calls in code
- Averages calculated at end of run to minimize communication

Efficiency Results

Efficiency of Parallel AMR running Orszag-Tang problem, using 3 levels of refinement on mhdcluster



Communication Time

- For 64 processor run:
  runtime = 2 hours
  lost due to load imbalance = 72 min
- Using mpiP profiler
  data sent per node = 82gb
  messages sent per node = 633k
- Using SKaMPI benchmarker on mhdcluster:
  bandwidth ≈ 350mb/s
  latency ≈ 2.45 μs.
- $\implies$ Bandwith time ≈ 4m, latency time ≈ 1.55s
- If perfect scaling runtime would be 17 min: Have 25% of runtime unaccounted for.

Communication Time

- For 64 processor run:
  runtime = 2 hours
  lost due to load imbalance = 72 min
- Using mpiP profiler
  data sent per node = 82gb
  messages sent per node = 633k
- Using SKaMPI benchmarker on mhdcluster:
  bandwidth ≈ 350mb/s
  latency ≈ 2.45 μs.
- $\implies$ Bandwith time ≈ 4m, latency time ≈ 1.55s
- If perfect scaling runtime would be 17 min: Have 25% of runtime unaccounted for.

Communication Time

- For 64 processor run:
  runtime = 2 hours
  lost due to load imbalance = 72 min
- Using mpiP profiler
  data sent per node = 82gb
  messages sent per node = 633k
- Using SKaMPI benchmarker on mhdcluster:
  bandwidth ≈ 350mb/s
  latency ≈ 2.45 μs.
- $\implies$ Bandwith time ≈ 4m, latency time ≈ 1.55s
- If perfect scaling runtime would be 17 min: Have 25% of runtime unaccounted for.

Communication Time

- For 64 processor run:
  runtime = 2 hours
  lost due to load imbalance = 72 min
- Using mpiP profiler
  data sent per node = 82gb
  messages sent per node = $633\mathrm{k}$
- Using SKaMPI benchmarker on mhdcluster:
  bandwidth ≈ 350mb/s
  latency ≈ 2.45 $\mu\mathrm{s}$.
- $\implies$ Bandwith time ≈ 4m, latency time ≈ 1.55s
- If perfect scaling runtime would be 17 min: Have 25% of runtime unaccounted for.

Communication Time

- For 64 processor run:
  runtime = 2 hours
  lost due to load imbalance = 72 min
- Using mpiP profiler
  data sent per node = 82gb
  messages sent per node = 633k
- Using SKaMPI benchmarker on mhdcluster:
  bandwidth ≈ 350mb/s
  latency ≈ 2.45 μs.
- $\implies$ Bandwith time ≈ 4m, latency time ≈ 1.55s
- If perfect scaling runtime would be 17 min: Have 25% of runtime unaccounted for.

Further Work

- Find missing 25%
- Load balance Parallel AMR
- Implemenent non-blocking communication
- **Couple to parallel Vlasov**