ECSS Experience: Particle Tracing Reinvented

Carlos Rosales, Robert McLay
{carlos,mclay}@tacc.utexas.edu

XSEDE14

Thanks

- Roman S. Voronov

  New Jersey Institute of Technology

- Dimitrios Papavassiliou

  University of Oklahoma

Overview

- The ECSS Program
- Problem description
- Basic parallel implementation
- Scalability improvements
  - Parallel IO
  - Work load balance
  - Spatial buffering
  - Temporal Buffering
- Results and future work

ECSS Program

- Extended Collaborative Support Service
- Aimed to provide technical support to research groups using XSEDE resources
- Extended support (up to a year, renewable)
- Many areas
  - Performance analysis
  - IO optimization
  - Visualization
  - Community Gateways
  - Data Flows

Original ECSS Project

- Observed lack of performance in tracing step
- Velocity field generated by Lattice Boltzmann Method (one file per task, thousands of tasks)
- Profile indicates most time spent on MPI data exchanges
- PI concerned with how to improve MPI performance
- After investigating code it seems that there are severe limitations to the implementation used

Problem Description

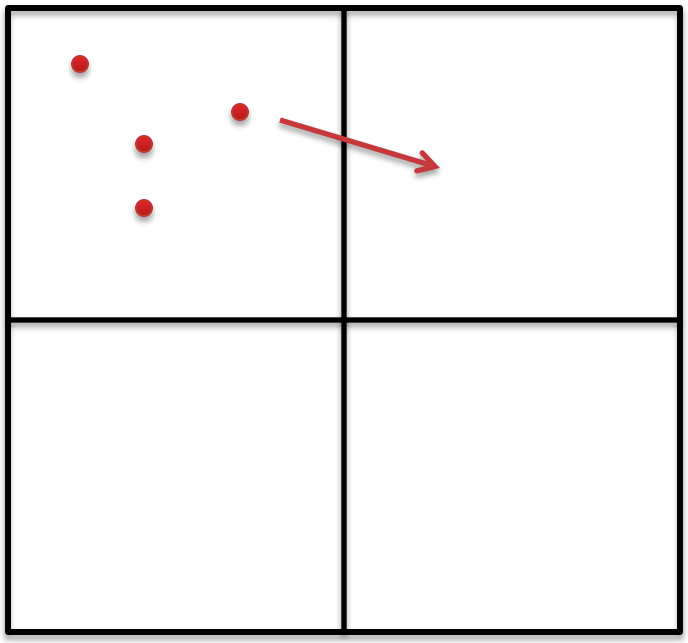
- Given a fixed velocity field, trace a number N of massless particles for a number of timesteps T
- Simple problem, present in many simulation workflows
- Single thread solution is trivial
- Significant complications at scale

Basic Parallel Implementation

- Domain decomposition using MPI
- Each MPI task
  - Reads a file with its velocity data
  - Tracks particles within its volume
    - Interpolates velocity
    - Calculates new position
    - Applies boundary conditions if necessary
  - Exchanges data with all its neighbors at every time step
- Simple to implement but…

Basic Implementation Issues

- Tracing forced to occur in as many nodes as the simulation
  - Inefficient, as tracing is a much more lightweight calculation
  - Often makes the tracing network-bound
- Task load only balanced for homogeneous particle distribution
  - Many simulations have long residence time volumes
  - Often leaves many tasks idle



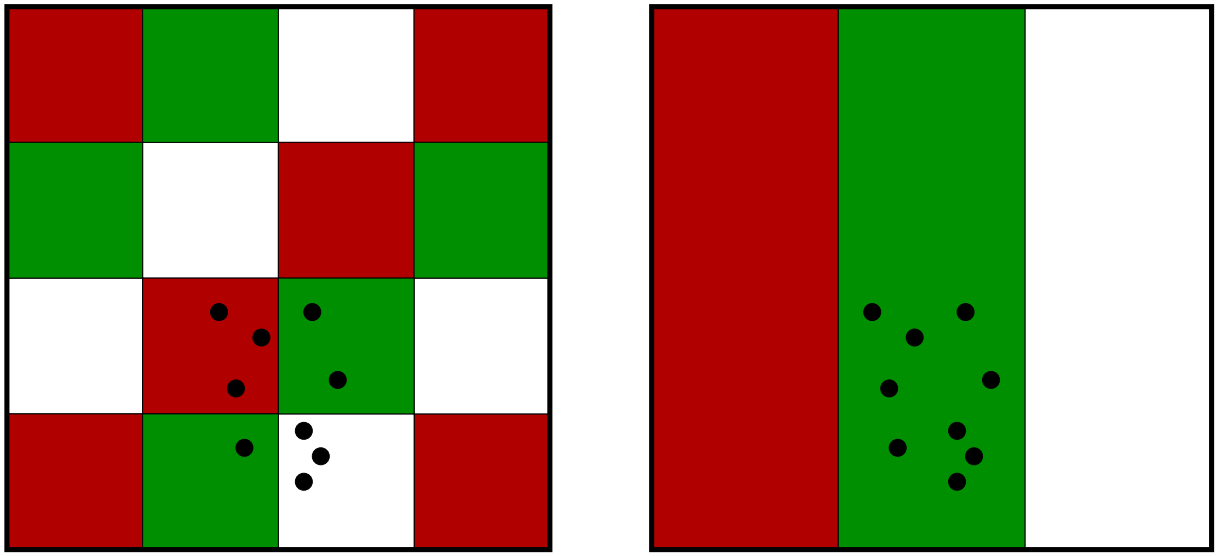
Introducing Parallel IO

- Parallel HDF5 introduced to
  - Allow different number of nodes for simulation and tracing steps
  - Improve velocity field reading times
- This in itself would represent a large performance improvement
  - In case study simulation run of 1k-2k cores but only 1k-10k particles were traced
  - **Tracing step can be run more efficiently on a few nodes only!**

Introducing Static Load Balance

- Use multiple disjoint subdomains per MPI task
- Even with heterogeneous distribution of particles load balance should be acceptable
- This reduces the number of idle tasks at any given time
- Subdomain size and number can be adjusted to deal with difficult cases
- Extreme cases may still require dynamic, measurement-based, adjustment

Multiple Subdomain Example



3 MPI Task, 9 Particles, 5 subdomains per task

66% resources wasted with basic implementation

Multiple Subdomain Issues

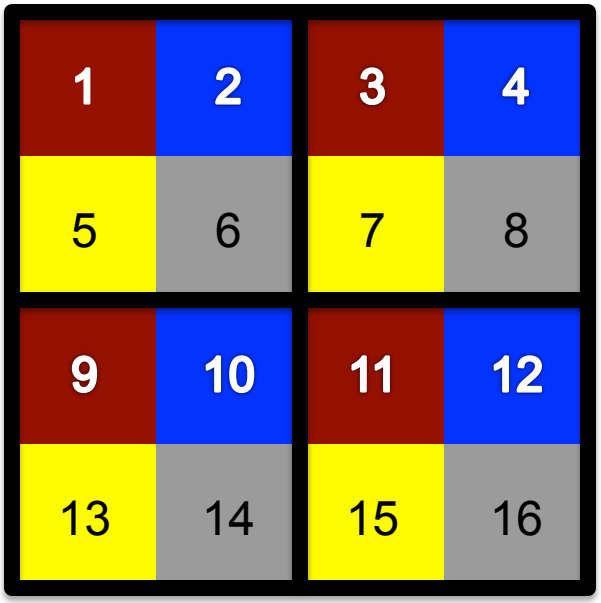
- **Handling input**
  - Many small blocks of data
  - Low IO rates
  - Need to aggregate reads somehow
- **Increased number of MPI exchanges**
  - Need to add spatial buffering to be effective
- Additional checks for subdomain boundary penetration
  - Checks performed in very short loops (not too expensive compared to basic implementation)

Handling the Input

- Determine *Big Block* to *Small Block* mapping
  - A *Big Block* is a set of complete subdomains
  - A *Small Block* is a subdomain
- Each MPI Task reads a single *Big Block* using PHDF5
  - Large, contiguous block of data
- *Small Blocks* are distributed to task neighbors using MPI
- Another option would be to use slab unions in HDF5

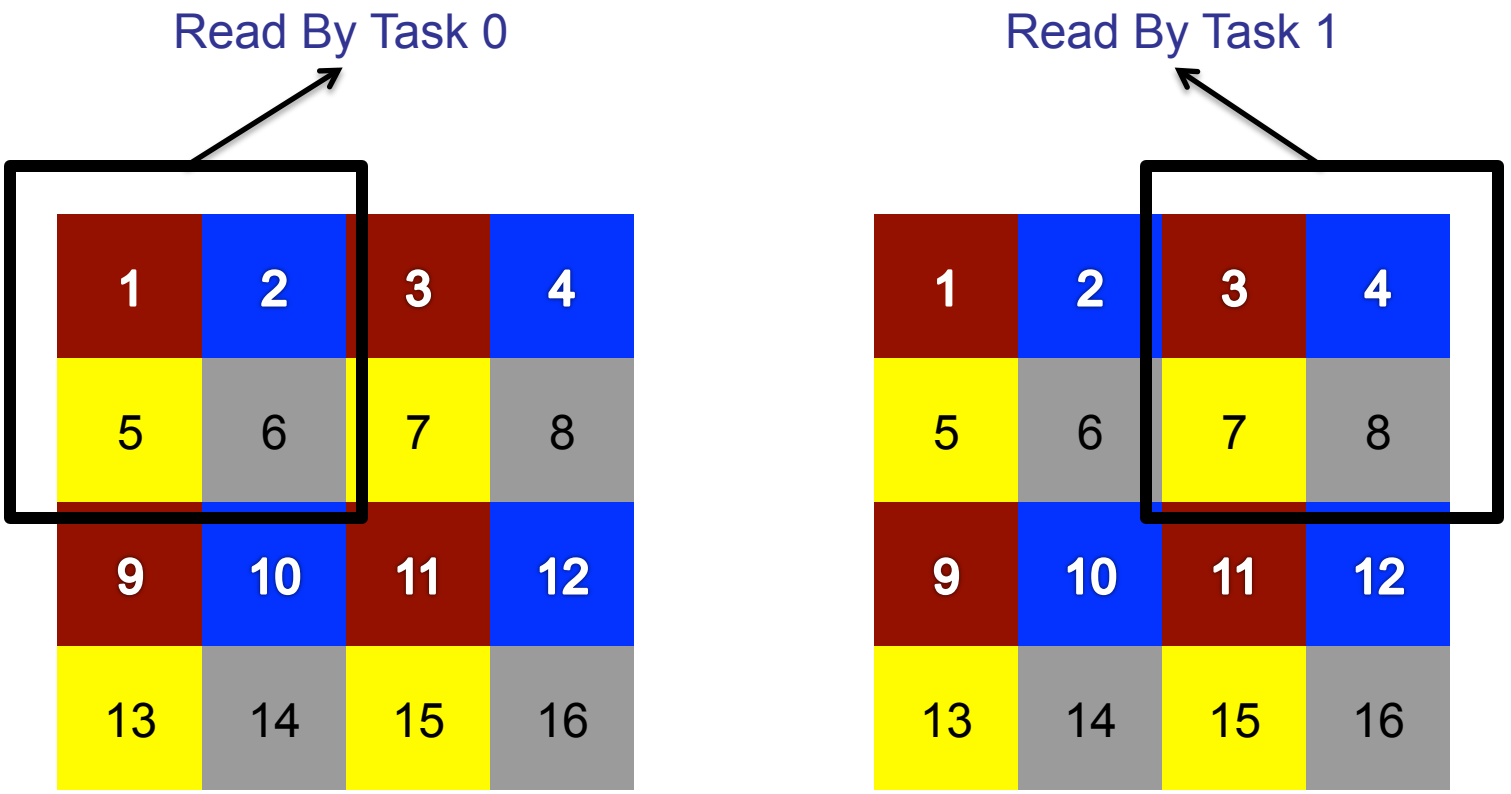
Big Blocks vs Small Blocks (I)

2D partition, 4 MPI tasks
16 subdomains



Task	Primary Domains
0	1, 3, 9, 11
1	2, 4, 10, 12
2	5, 7, 13, 15
3	6, 8, 14, 16

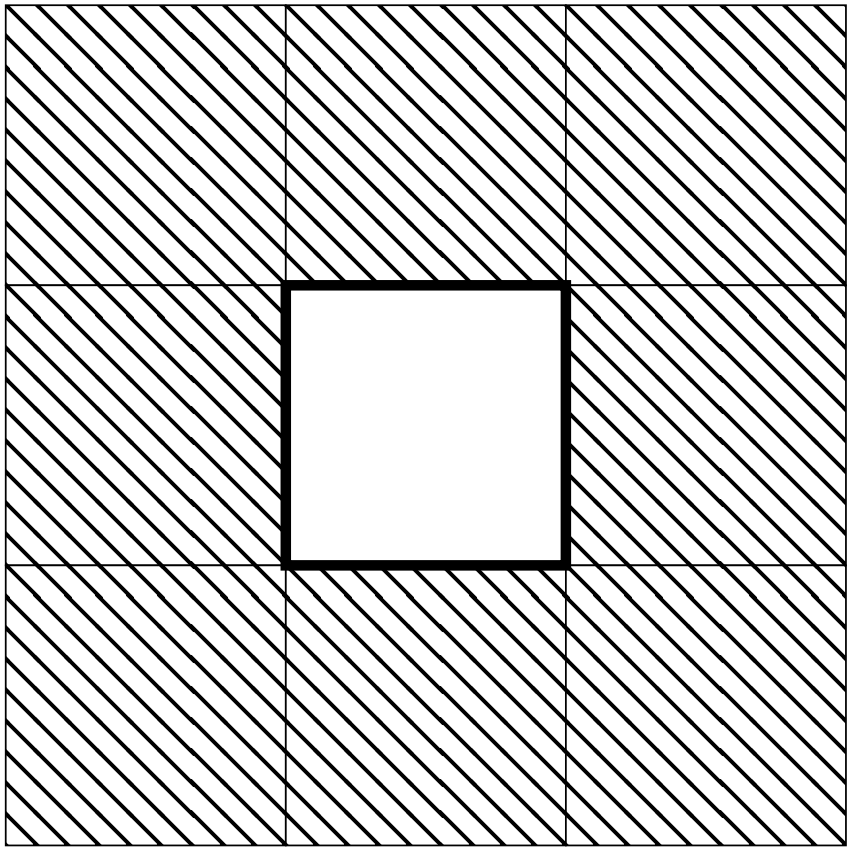
Big Blocks vs Small Blocks (II)



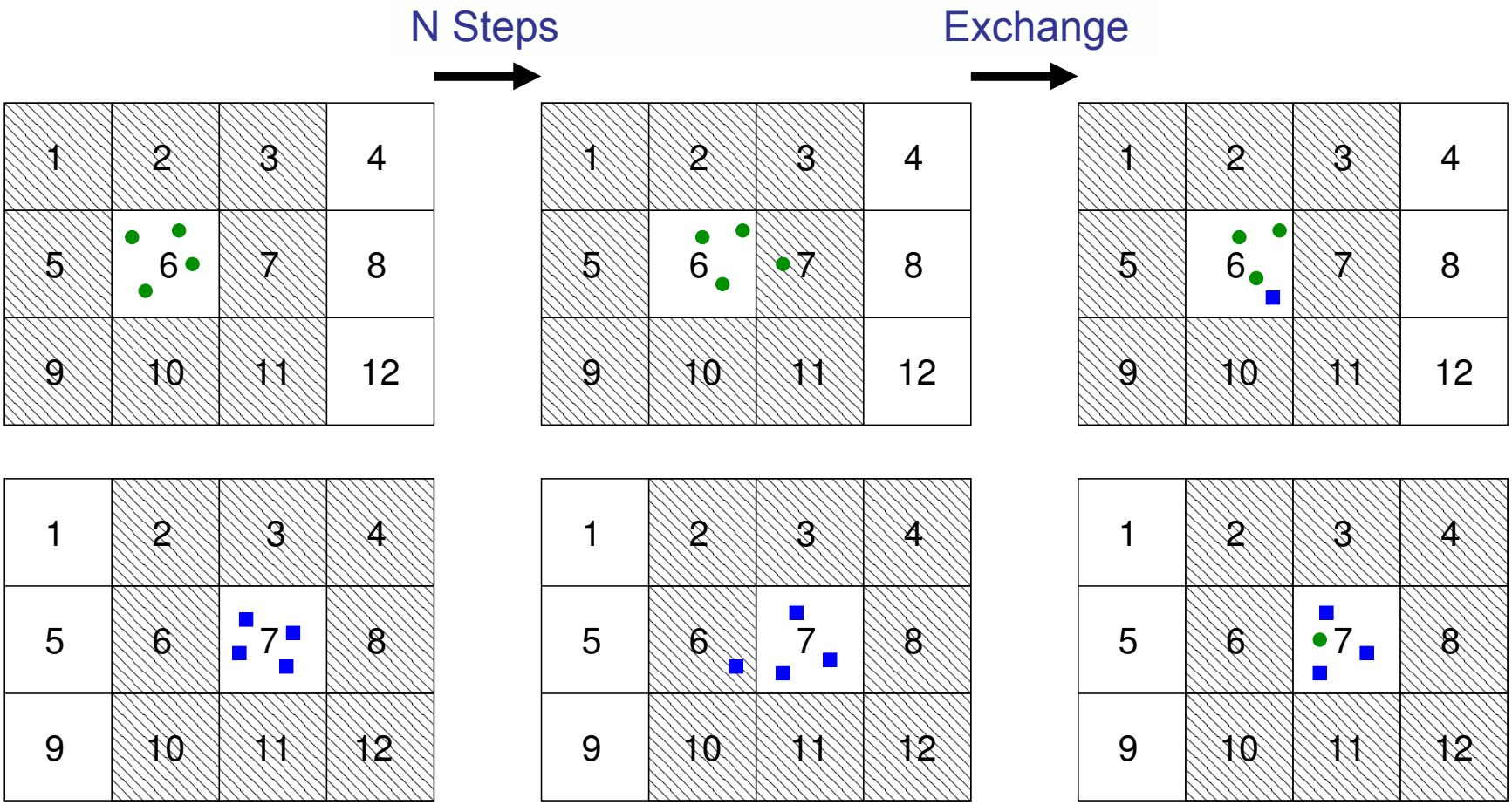
After the read each task sends small block information to its neighbors

Spatial Buffering

- Each task buffers all subdomains neighboring its assigned subdomains
- No duplicated storage for overlapping buffered volumes
- Increases memory usage
- Reduces data exchange rate between MPI tasks
- Particle can be traced inside buffered area
- Particle will only be reassigned to new MPI task in next exchange



Buffered Tracer Exchange



Temporal Buffering

- Maximum local velocity determined from input for each MPI task
- Maximum distance a tracer is allowed to move before an MPI exchange occurs is the subdomain size

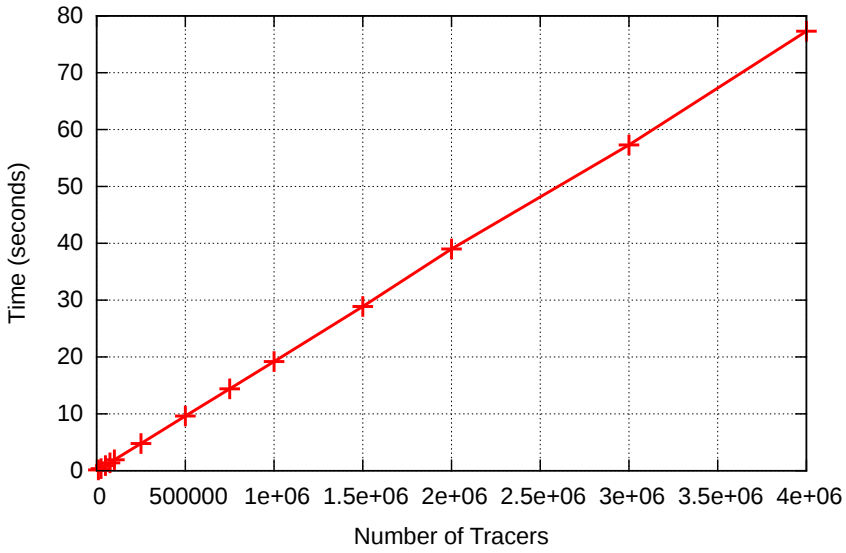
$$N_{max}=L/(\delta t\ U_{max})$$

- For Lattice Boltzmann Method solvers the velocity is small compared to the mesh size, so this number can be several thousand time steps
- Using smaller subdomains (for improved static load balance) reduces Nmax
- Could be dynamically adjusted

Results for a Simple Geometry

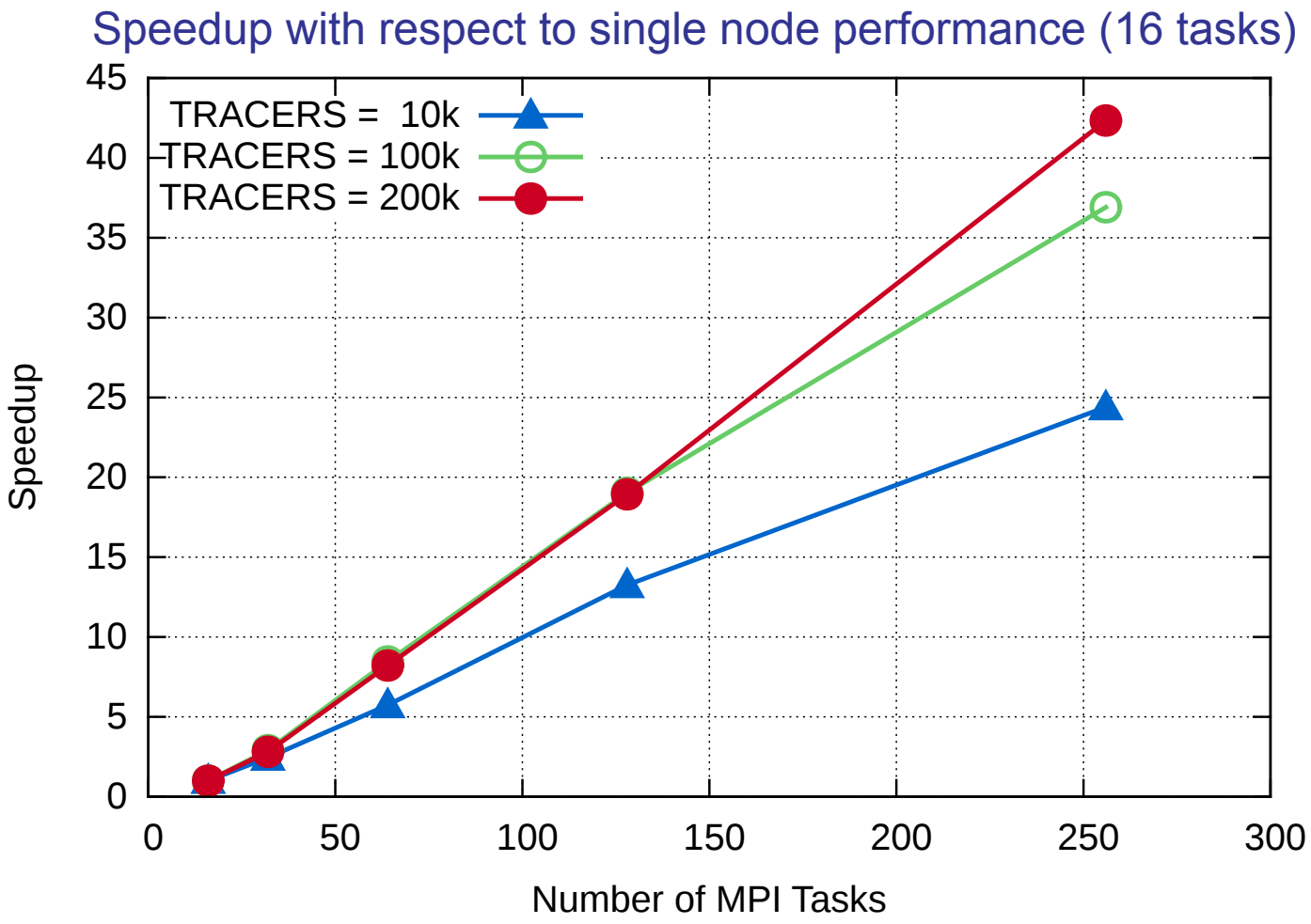
- Simplest case we could think of
- Straight pipe
- Parabolic velocity flow profile
- Particles released at inlet
- Particles that hit walls bounce forward
- Particles that reach outlet eliminated from tracking

Scaling with Tracers (16 tasks)



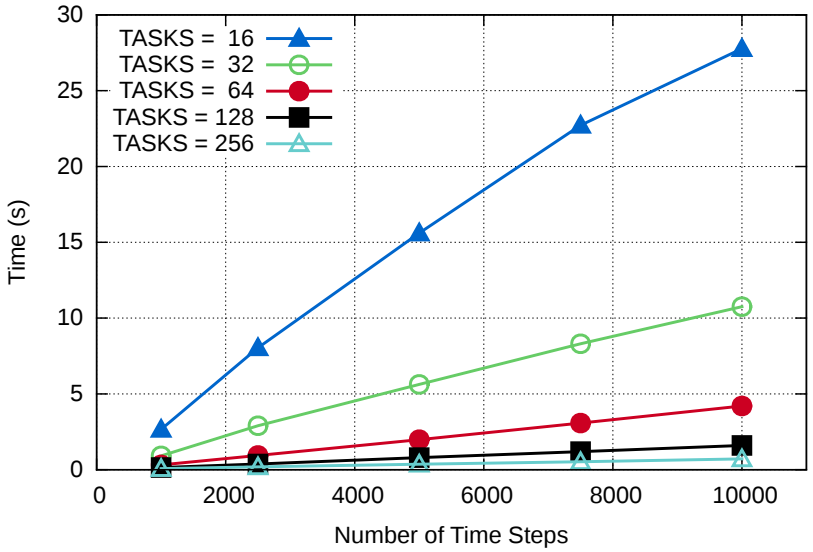
- No obvious issues with scaling in tracer number
- Easy to trace millions of particles
- However a single node is too limited a test

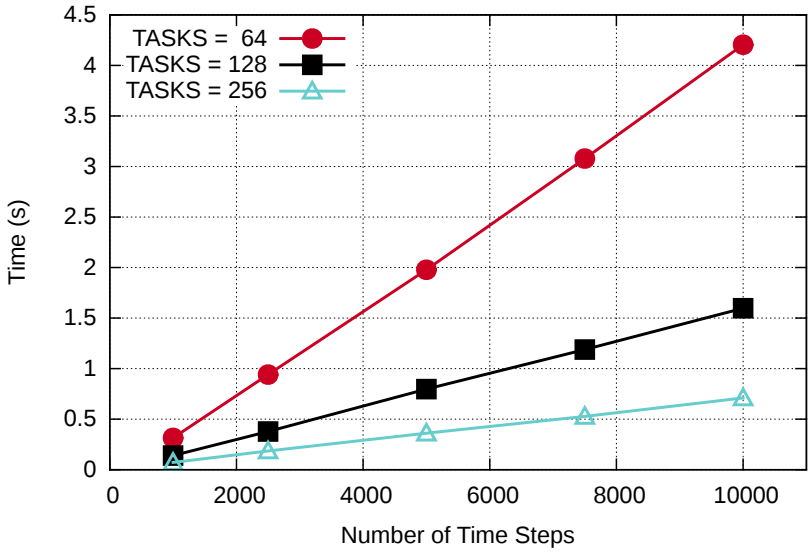
Scaling with Tracers



Scaling with Tracing Length

10k Particles in a 256x256x256 Domain



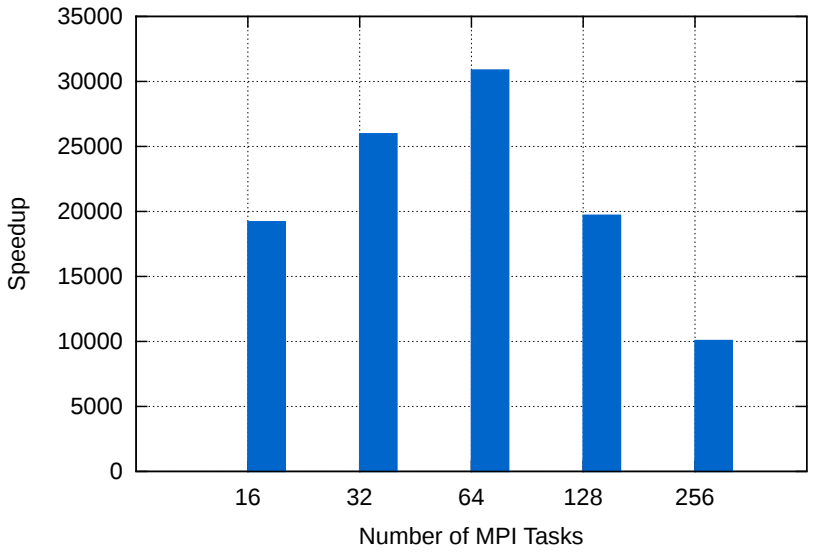


No surprises, as expected the execution time is mostly linear with the number of tracing steps

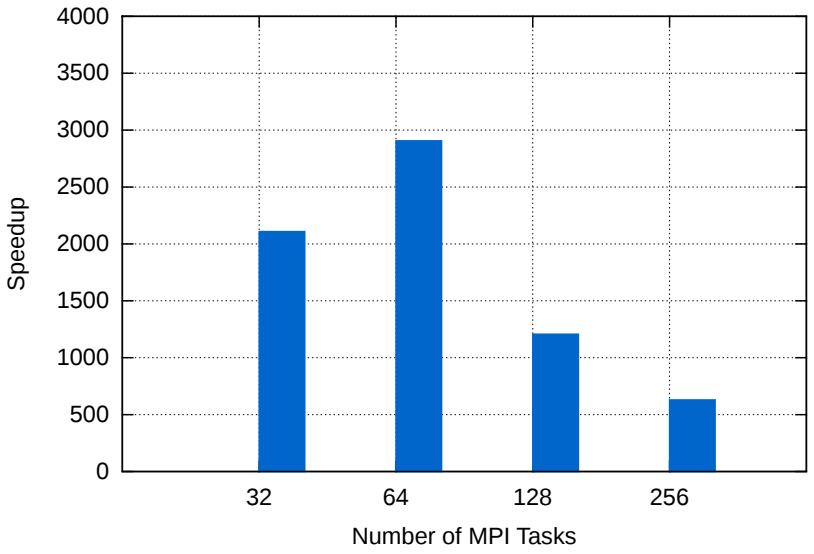
TACC

Speedup vs Basic (256)

Tracing Only



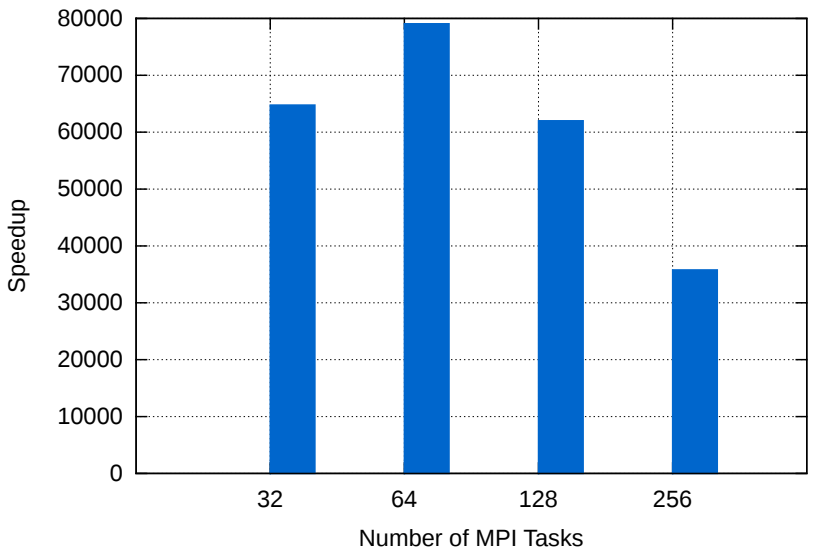
Setup + Tracing



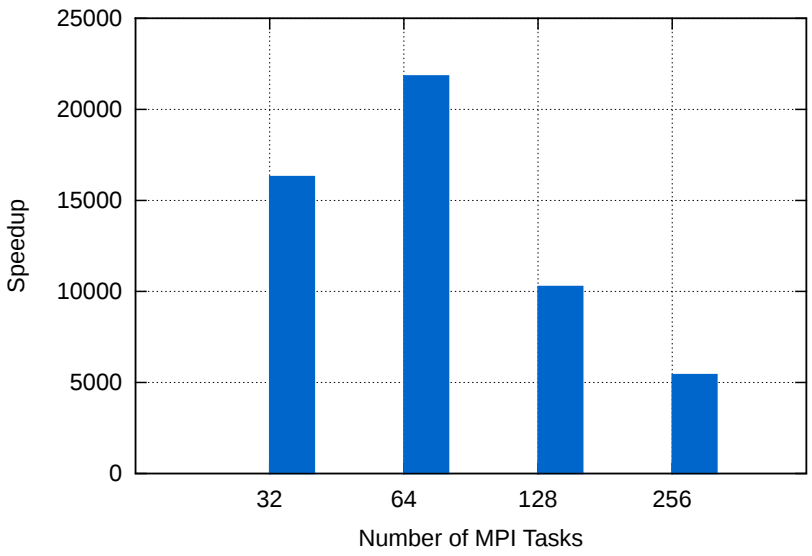
10k Particles, 256x256x256 domain, 1k steps

Speedup vs Basic (512)

Tracing Only



Setup + Tracing



10k Particles, 512x512x512 domain, 1k steps

Conclusions & Future work

- Very large speedup in tracing obtained
- Tracing step no longer bottleneck
- Traditional techniques used
  - Static load balancing
  - Parallel IO
- Buffering required to achieve high performance
- No black magic, just sensible coding (and lots of colored pens)

- Future Work
  - Open code repository to public
  - Implement dynamic workload rebalancing for exceptionally tough cases
  - Implement automatic memory reallocation for temporary arrays when needed
  - Reduce setup times

Thanks!