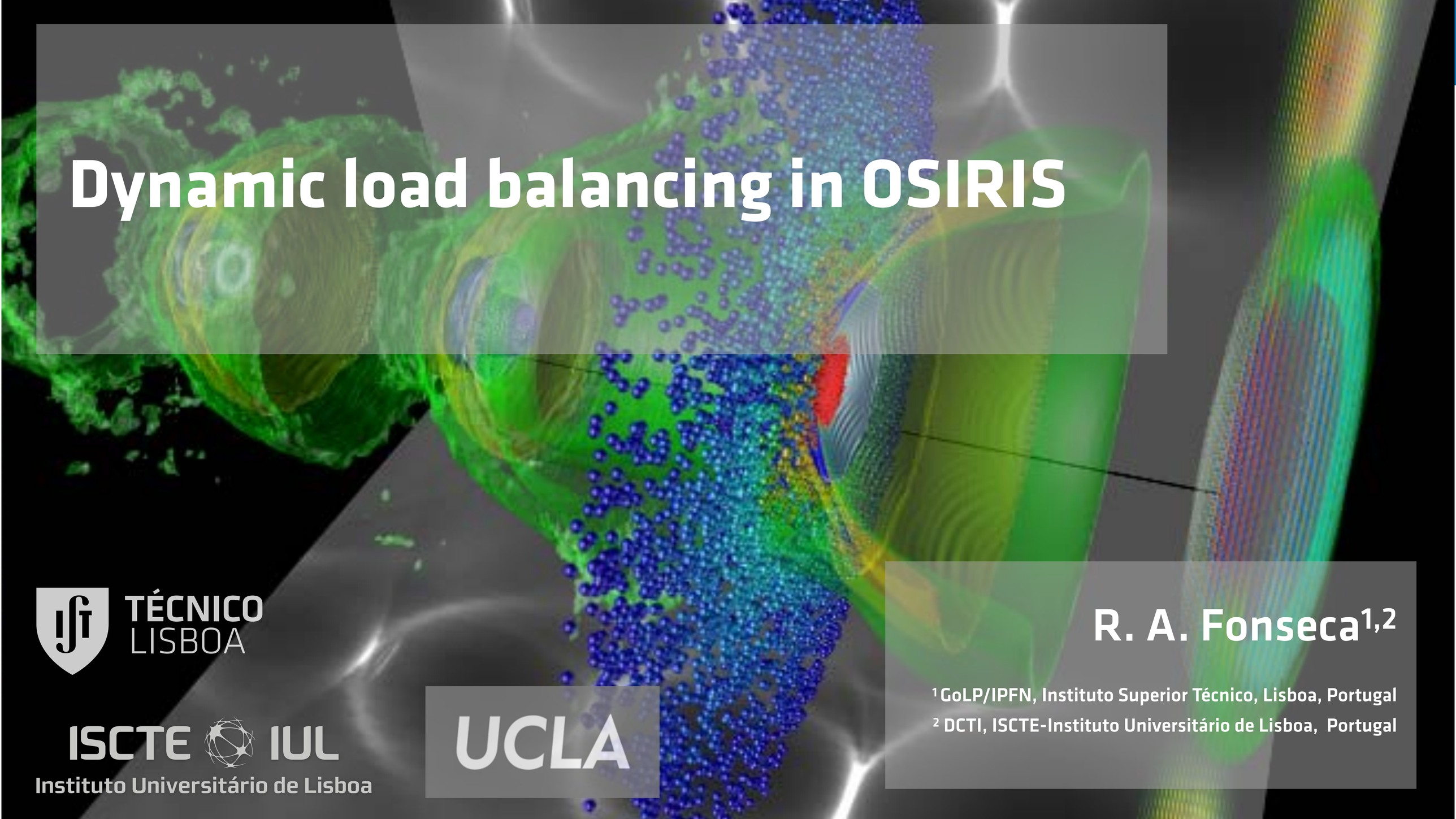
Dynamic load balancing in OSIRIS



TÉCNICO LISBOA

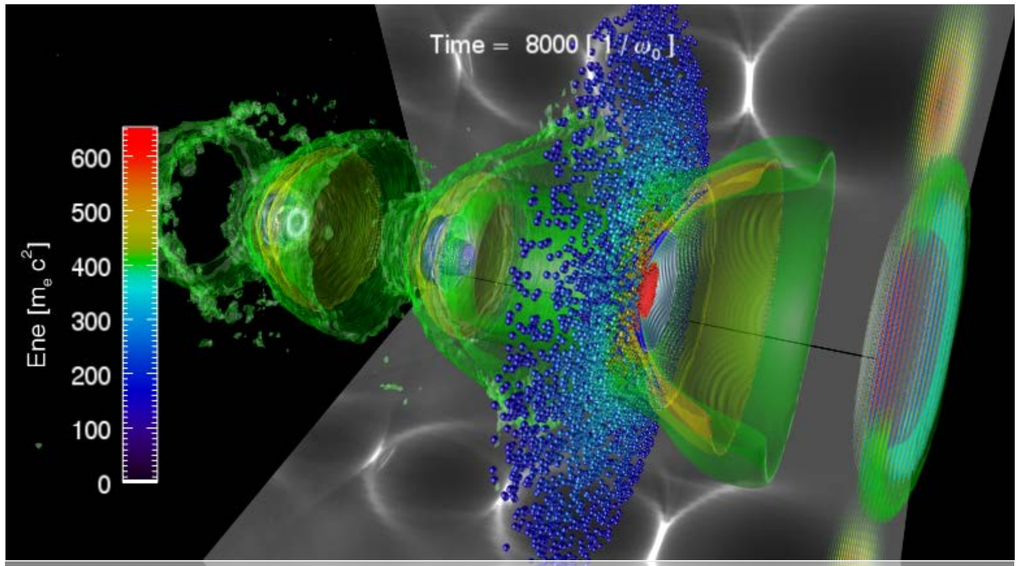
ISCTE IUL
Instituto Universitário de Lisboa

UCLA

R. A. Fonseca[1,2]

[1] GoLP/IPFN, Instituto Superior Técnico, Lisboa, Portugal
[2] DCTI, ISCTE-Instituto Universitário de Lisboa, Portugal

Maintaining parallel load balance is crucial



LWFA Simulation

Parallel Partition

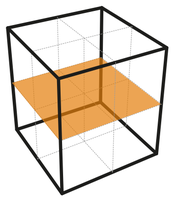
- 94×24×24 = 55k cores

Load Imbalance (max/avg load)

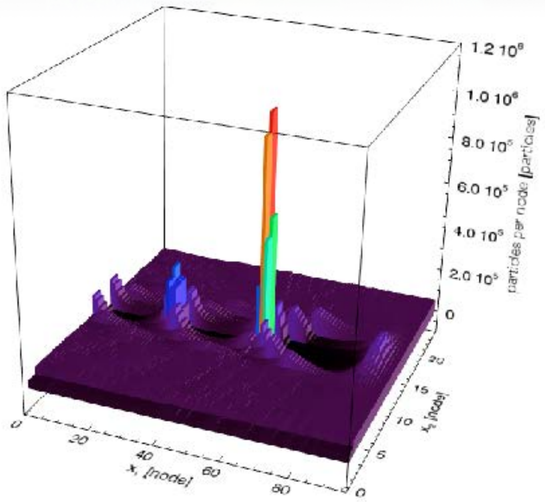
- 9.04×

Average Performance

- ~12% peak



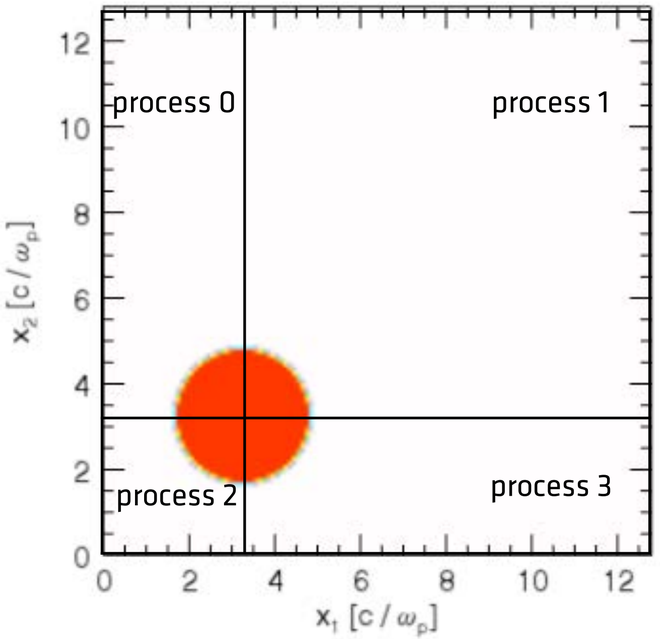
Particles/node, i_z = 12



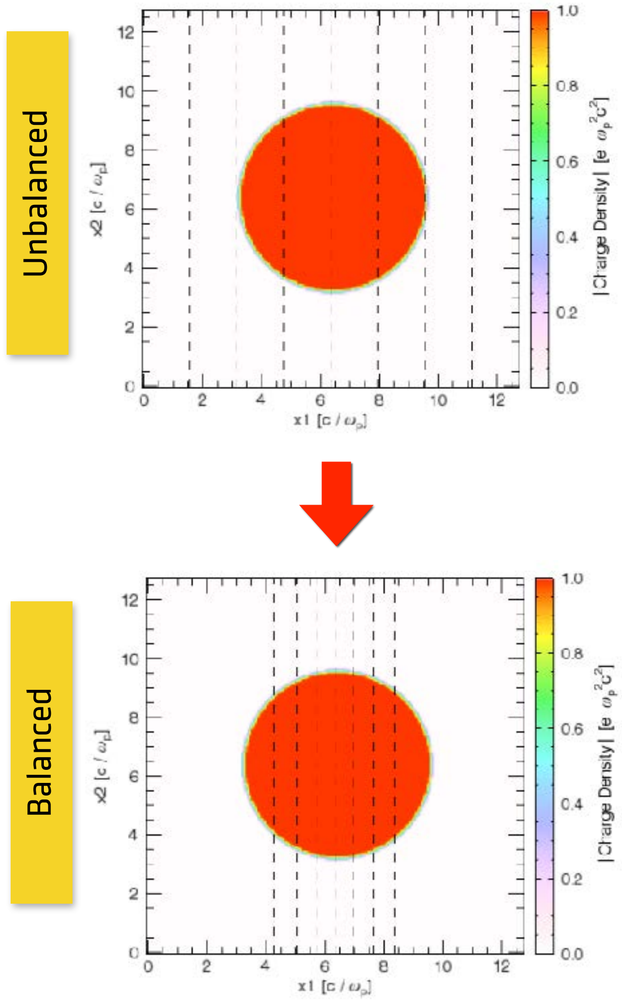
- **Full scale 3D modeling of relevant scenarios requires scalability to large number of cores**
  - Code performance must be sustained throughout the entire simulation
- **The overall performance will be limited by the slowest node**
  - Simulation time is dominated by particle calculations
  - Some nodes may have more particles than other
  - If the distribution of particles remains approximately constant throughout the simulation we could tackle this at initialization
    - Static load balancing
  - However this will usually depend on the dynamics of the simulation
- **Shared memory parallelism can help mitigate the problem**
  - Use a “particle domain” decomposition inside shared memory region
  - Smear out localized computational load peaks
- **Dynamic load balancing required for optimal efficiency**

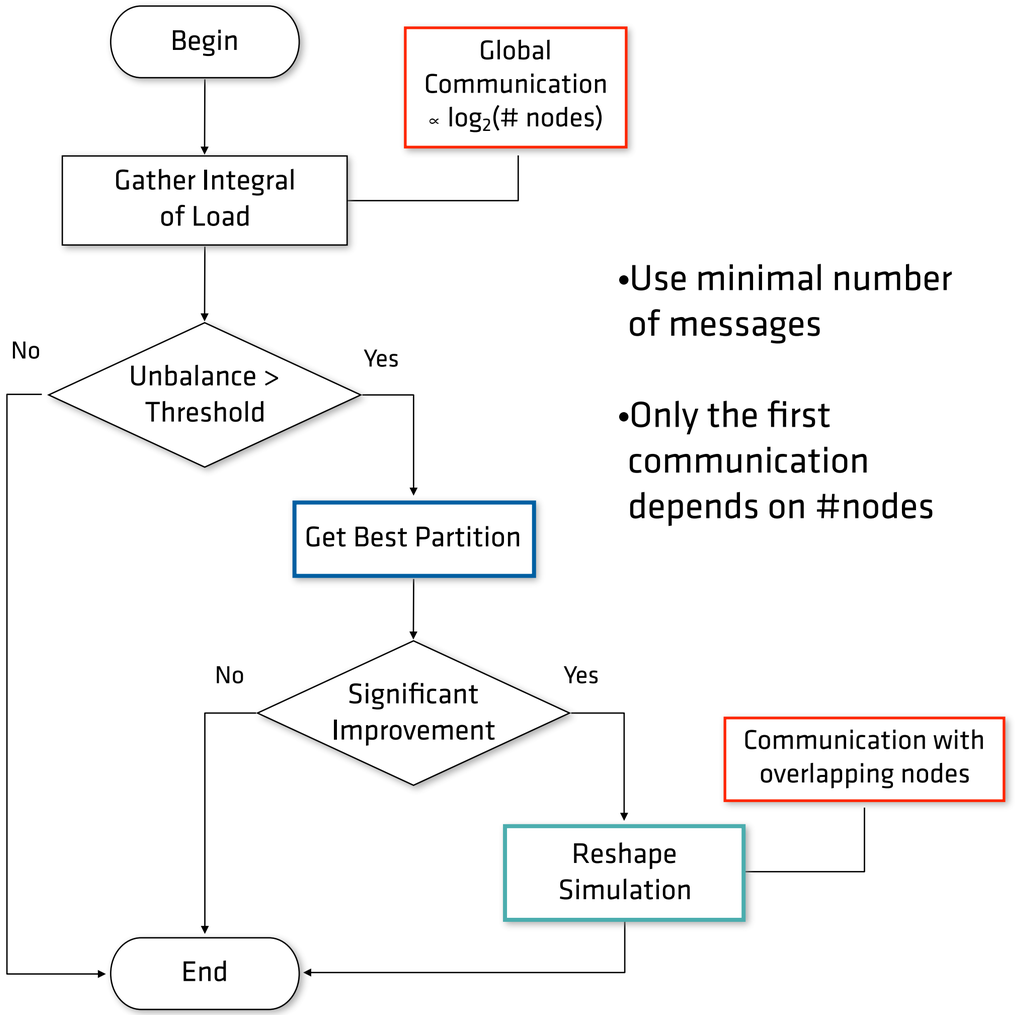
Adjusting processor load dynamically

- The code can change node boundaries dynamically to attempt to maintain a even load across nodes:
  - Determine best possible partition from current particle distribution
  - Reshape the simulation



Even particle load across processes 4 cores

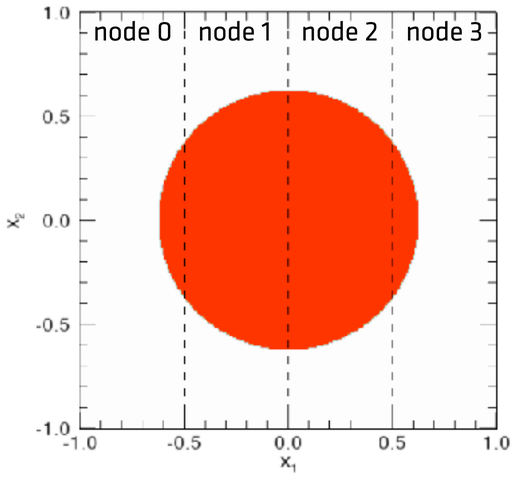




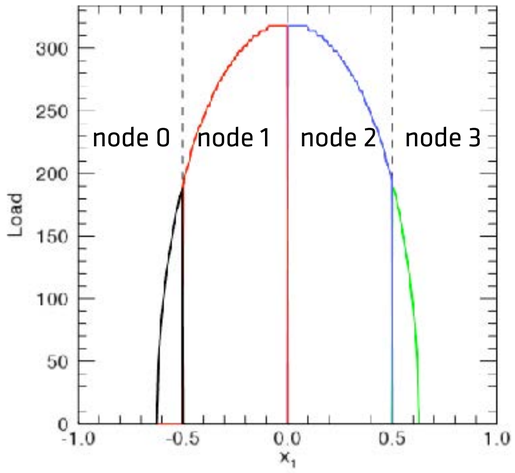
- Use minimal number of messages
- Only the first communication depends on #nodes

Calculate cumulative load along load balance direction

- **The algorithm for calculating the best partition requires the cumulative computational load**
  - Essentially a scan operation
  - Implemented as a reduce all operation followed by local calculations
- **All nodes know the result**
  - Simplifies best partition calculations



Get Load along load balancing direction on each node

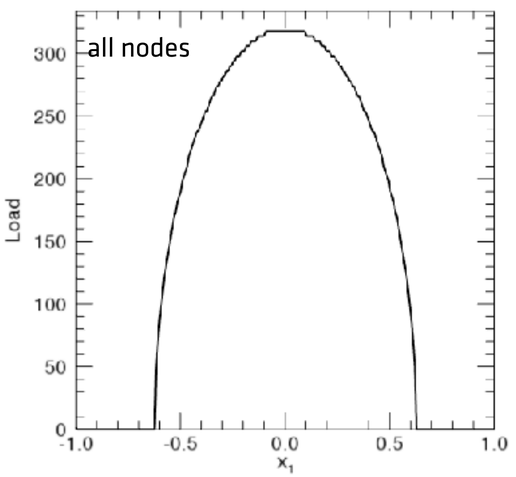


Main overhead (~90%)
Scales well for large number of nodes
$\propto \log_2$(# nodes)

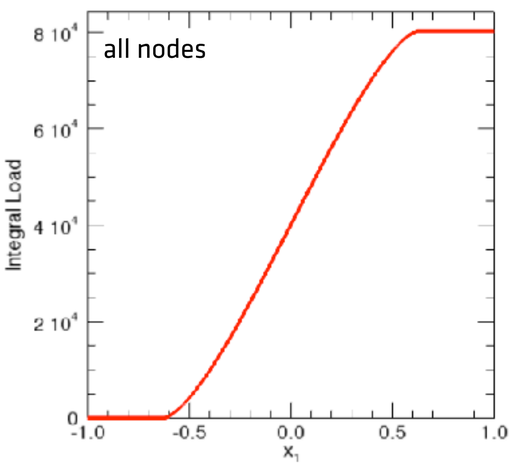
Add load from all nodes
+
Distribute result to all nodes

MPI_REDUCEALL(few kB)

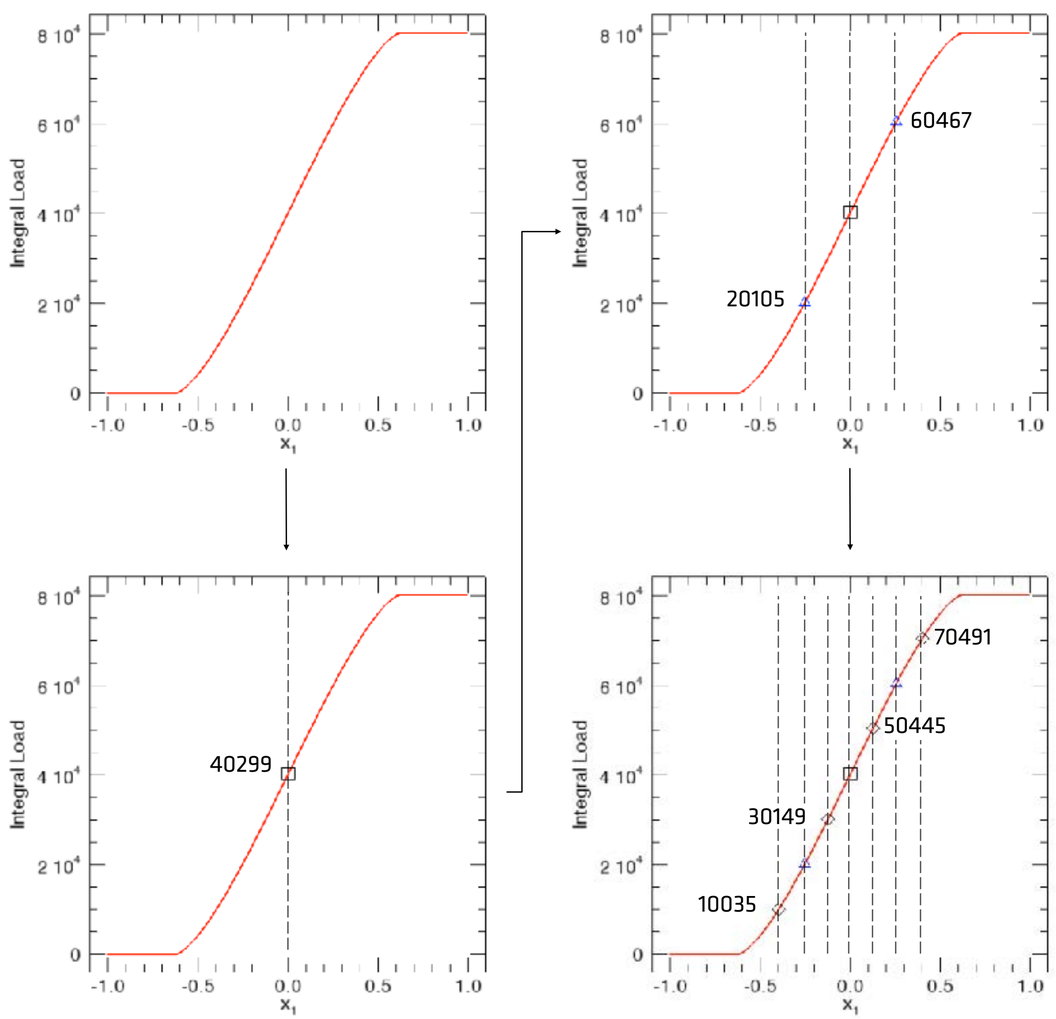
Global Integral Load is known by all nodes



Integrate Load along load balancing direction



Finding the best partition

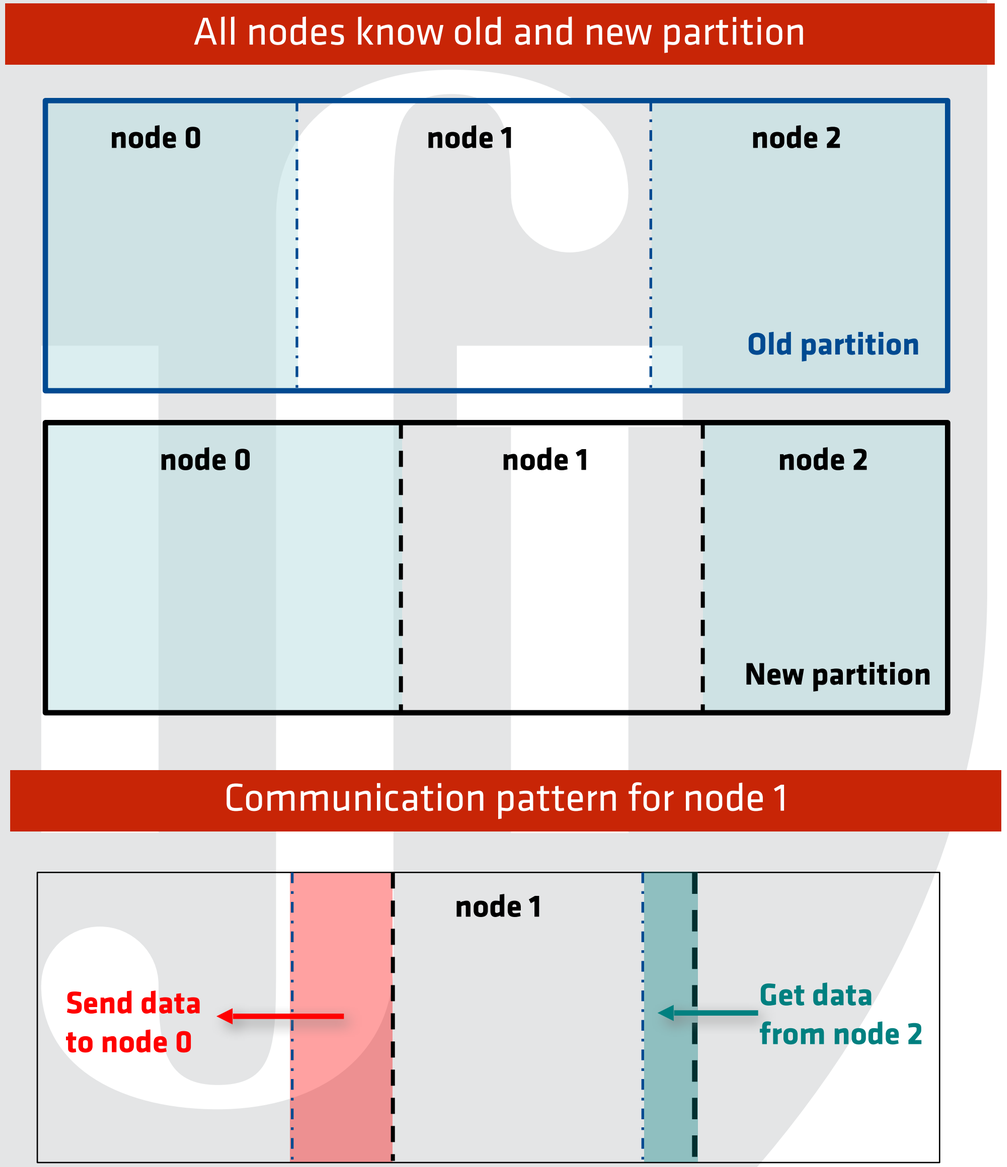


Binary Search Algorithm

- **Algorithm for 1D partitions**
  - Only 1D integral load required (less communication)
- **Find point where integral load is half between endpoints**
- **Repeat with two resulting intervals until all node partitions are calculated**
- **Same calculation is done by all nodes**
  - No need for extra communication
- **Fast algorithm**
- **Well balanced distribution**
- **Handles “impossible” scenarios well**
  - i.e. when it is impossible to find a load balanced situation it finds the best solution
- **Can handle extra constraints:**
  - minimum number of cells per node
  - Cells per node must be an integer multiple of some quantity

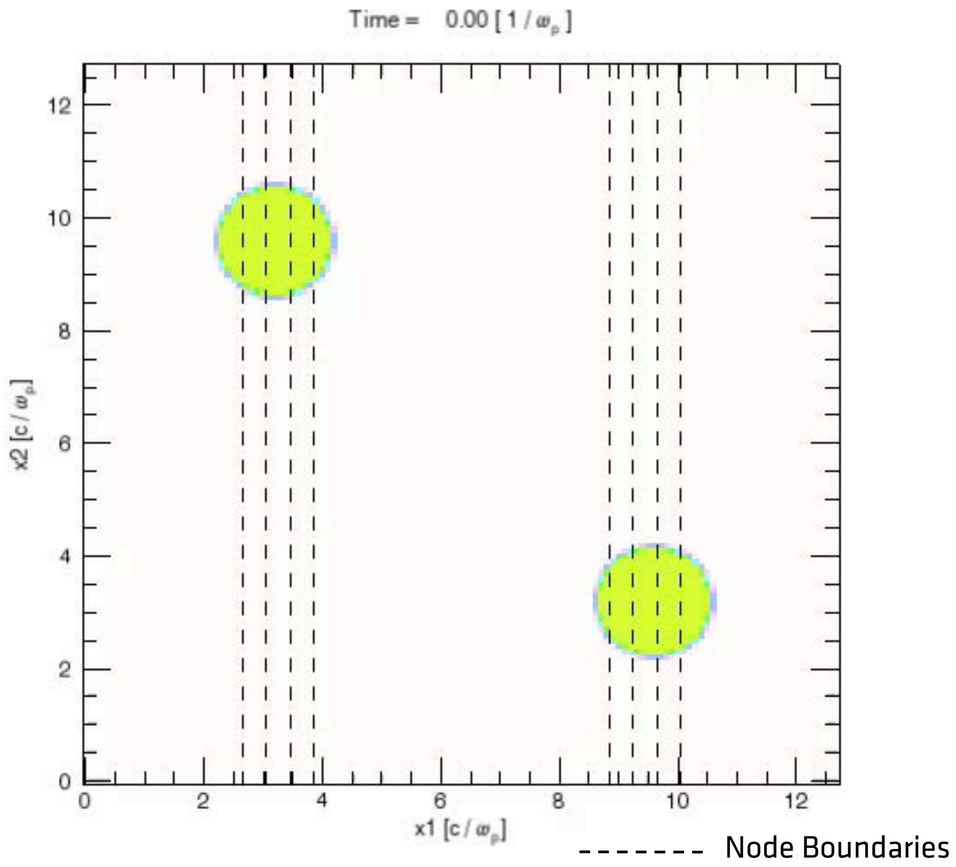
Partition reshaping

- **All nodes know previous and new partition**
- **For each node**
  - Detect overlap between local node on old partition and other nodes on new partition
  - Detect overlap between local node on new partition and other nodes on old partition
  - Send/receive data to proper nodes/from old nodes
- **For particles**
  - Remove/add to particle buffer
- **For grids**
  - Send/get data to/from other nodes
  - Allocate new grid with new shape
  - Copy in values from old grid and data from other nodes
  - Free old grid



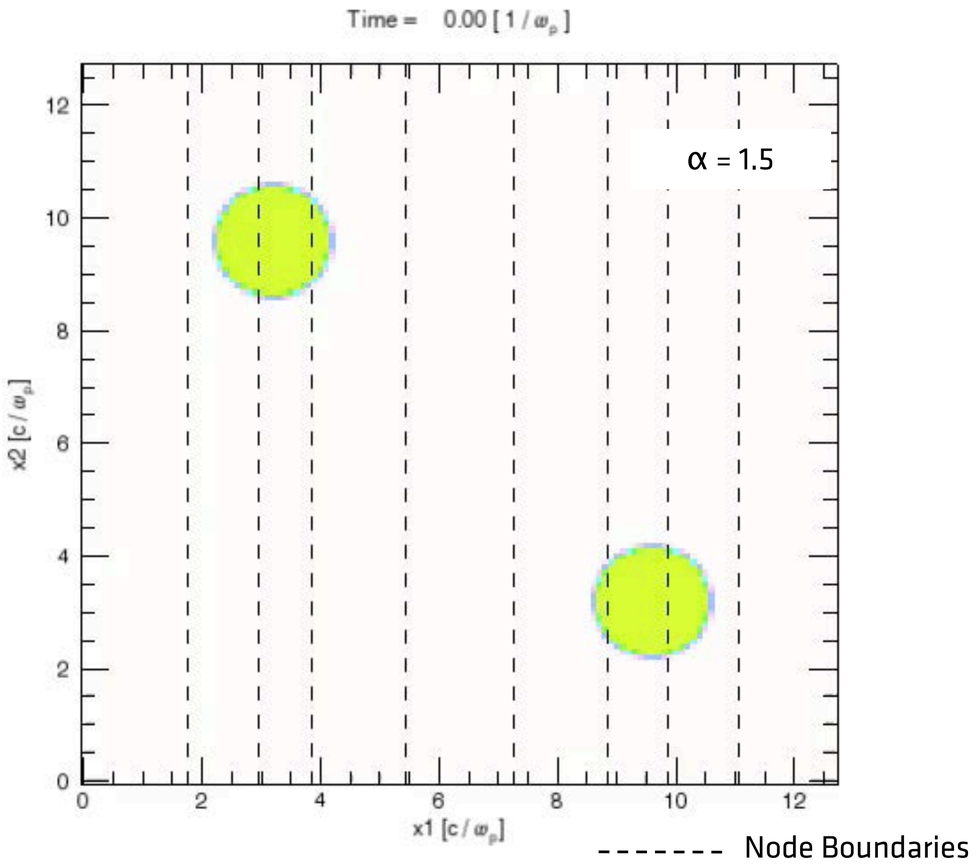
Including cells in the computational load

Particles only



- Load estimate considering only particles per vertical slice

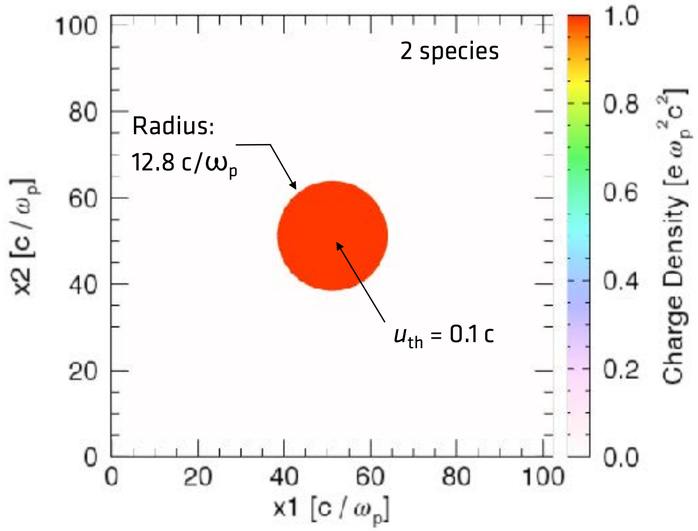
Particles + cells



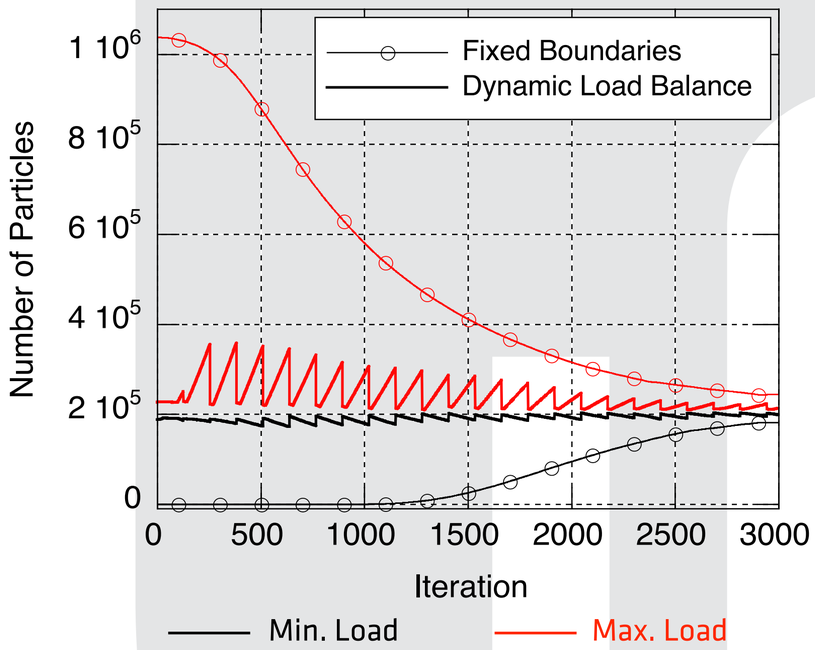
- Cell calculations (field solve + grid filtering) can be considered by including an effective cell weight
  - In this example each cell is considered to have the same computational load as 1.5 particles

Hydrodynamic nano plasma explosion

Simulation Setup

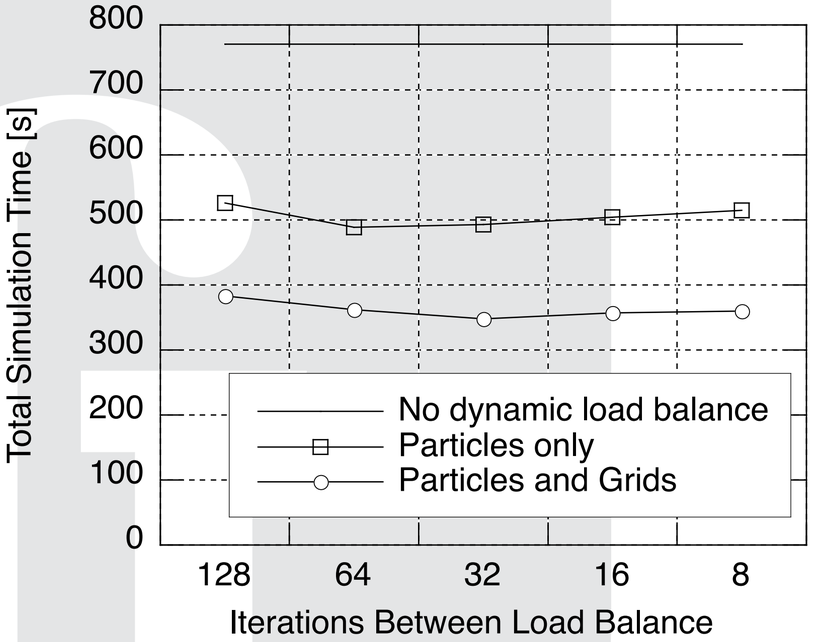


- 2D Cartesian geometry
- 3000 iterations
- Grid
  - 1024^2 cells
  - 102.4^2 $(c/\omega_p)^2$
- Particles
  - electrons + positrons
  - 8^2 particle/species/cell
  - Generalized thermal velocity 0.1 c



Load

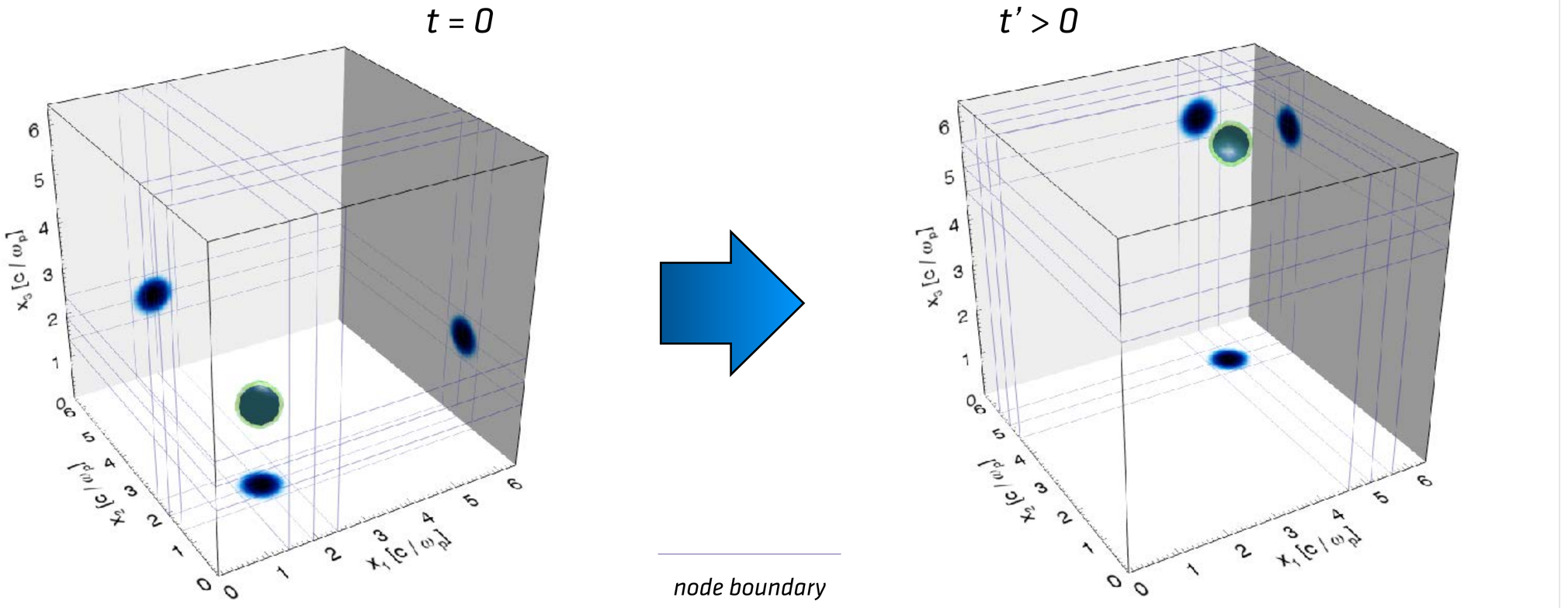
- Algorithm maintains the best possible load balance
- Repartitioning the simulation is a time consuming task
  - Don't do it too often!



Performance

- Particles & grids ~50% better than just particles
- $n_{\text{loadbalance}}$ ~64 iterations yields good results
- Performance boost ≥ 2
  - Other scenarios can be as high as 4

Multi-dimensional dynamic load balancing



Best Partition

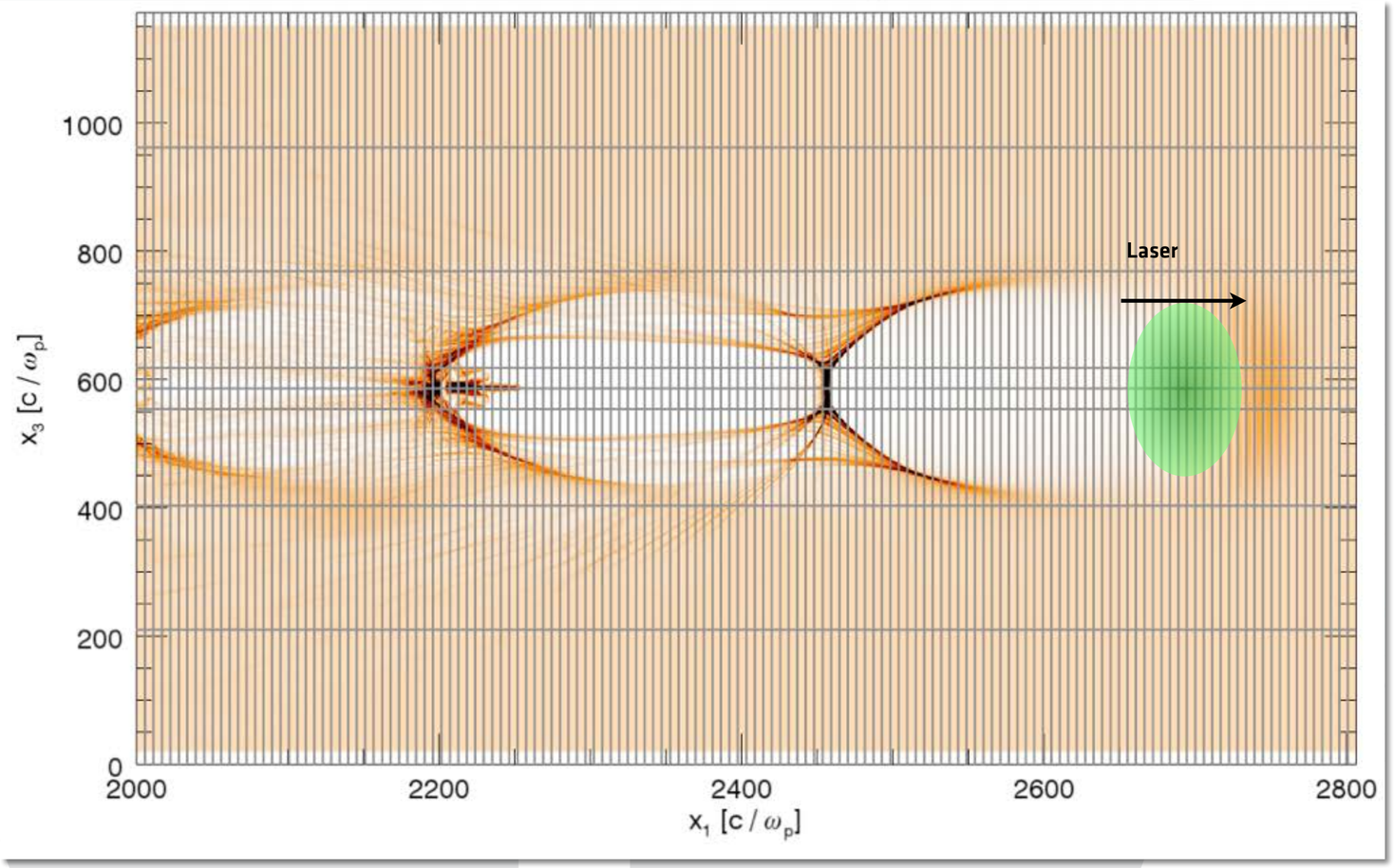
- Difficult task:
  - Single solution exists only for 1D parallel partitions
  - Improving load balance in one direction might result in worse results in another direction

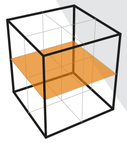
Multidimensional Partition

- Assume separable load function (i.e. load is a product of $fx(x) \times fy(y) \times fz(z)$)
  - Each partition direction becomes independent
- Accounting for transverse directions
  - Use max / sum value across perpendicular partition

Large scale LWFA run: Close but no cigar

- **The ASCR problems are very difficult to load balance**
  - Very small problem size per node
  - When choosing partitions with less nodes along propagation direction imbalance degrades significantly
- **Not enough room to dynamic load balance along propagation direction**
- **Dynamic load balancing in the transverse directions does not yield big improvements**
  - Very different distributions from slice to slice
  - Dynamic load balance in just 1 direction does not improve imbalance
  - Using max node load helps to highlight the hot spots for the algorithm





x_1-x_2 slice at box center
similar partition along x_3
> 30% improvement in inbalance

No overall speedup

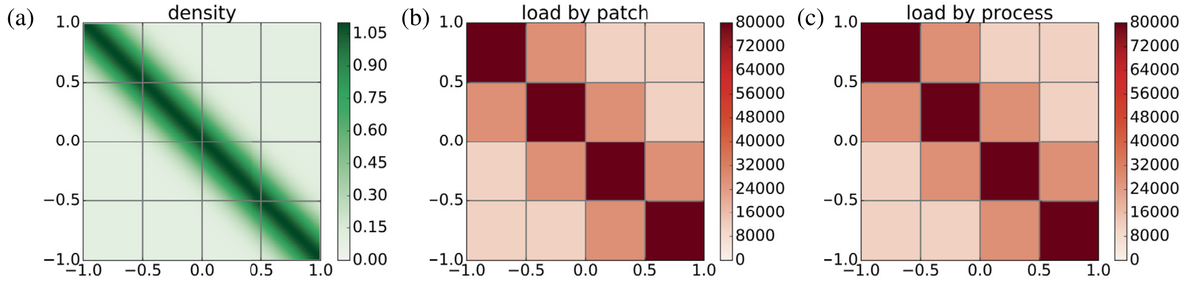
- Best result:
  - Dynamic load balance along x_2 and x_3
  - Use max load
  - 30% improvement in imbalance but…
  - Lead to a 5% overall slowdown!

Alternative: Patch based load balancing

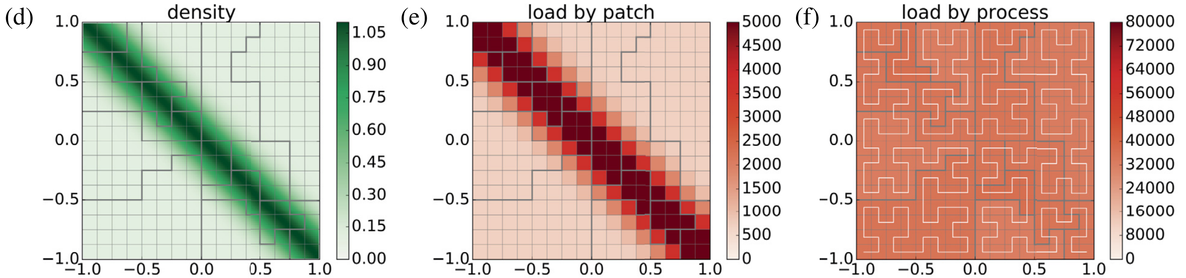
Implemented in PSC / SMILEI

- Partition the space into (10-100x) more domains (patches) than processing elements (PE)
- Dynamically assign patches to PE
  - Assign similar load to PEs
  - Attempt to maintain neighboring patches in the same PE

Standard MPI partition

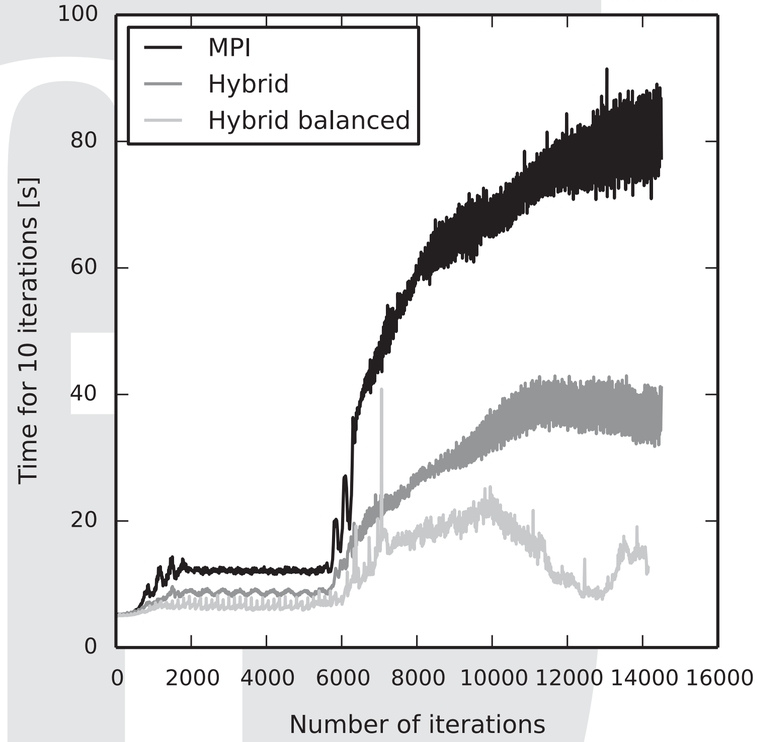


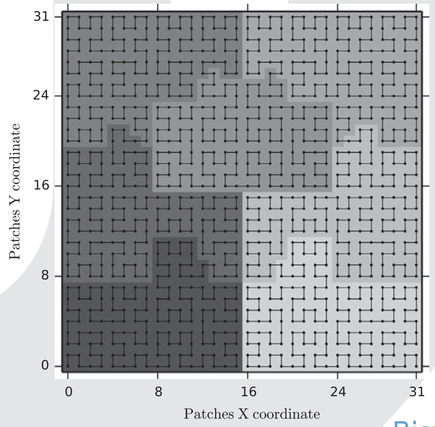
Patch based partitioning



Germaschewski et al., JCP **318** 305 (2016); Beck et al., NIMPR-A **829** 418 (2016)

LWFA simulation





Patch assignment

7 PE example

Overview

- **Maintaining load balancing is crucial for large scale simulations**
  - Lowest performing node dominates
  - Particle load per PE evolves dynamically along the simulation
  - Share memory parallelism helps smear out load spikes but does not solve the problem

- **Dynamic load balancing is implemented by moving node boundaries**
  - Works well for many problems
  - Difficult to expand to multi-dimensional partitions
  - Unable to handle difficult LWFA scenarios

- **Patch based dynamic load balancing appears to a viable solution**
  - Requires moving to a tiling scheme to organize the data
    - Lots of work, but has other advantages, e.g. improved data locality, similarity with GPU code
  - Current implementations rely on MPI_THREAD_MULTIPLE