# An Overview of the DiRAC Benchmark Suite

Richard Rollins, Sidharth Kashyap, Angus Lepper, Peter Boyle

DiRAC / University of Edinburgh

September 8, 2016

# The DiRAC–3 Benchmark Suite

*A cross–section of particle and astrophysics research software to guide purchasing for the next generation of DiRAC systems.*

Breakdown Across Three Services:

- Extreme–Scaling: Grid, QPhiX, MILCmk
- Memory–Intensive: Gadget3-Eagle, Swift, CloverLeaf3D
- Data–Intensive: IOR, Walls

Research Goals:

- Expose hardware–dependent performance of research software
- Realise opportunities for code optimisation

DiRAC-benchmarks/DiRAC3-testsuite

# Solving PDEs on Structured Grids

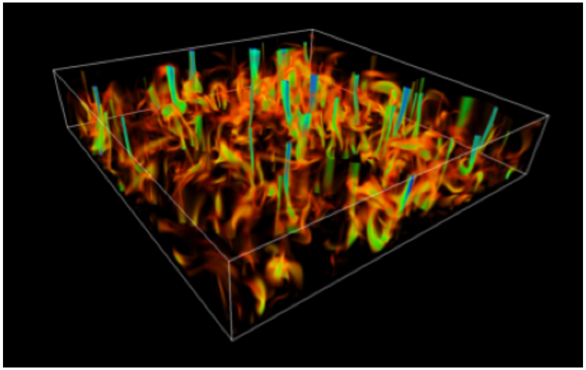
Used in many DiRAC projects spanning a range of different domains:

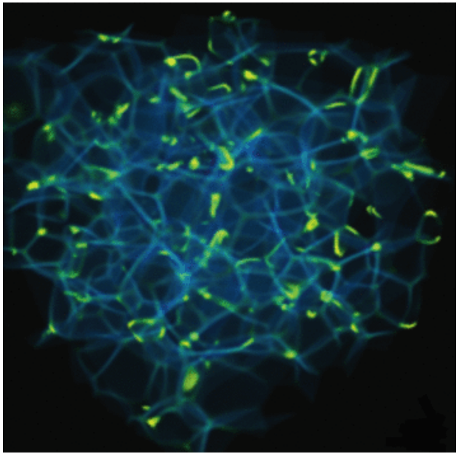
- Lattice QCD (Grid, MILC, QPhiX)
- Magnetohydrodynamics (CloverLeaf3D, Lare3D)
- Cosmological phase transitions (Walls)

Derivatives discretised on a structured grid e.g:

$$\nabla^2 \phi^{\mathrm{n}}_{\mathrm{ij}} = \phi^{\mathrm{n}}_{\mathrm{i+1j}} + \phi^{\mathrm{n}}_{\mathrm{i-1j}} + \phi^{\mathrm{n}}_{\mathrm{ij+1}} + \phi^{\mathrm{n}}_{\mathrm{ij-1}} - 4\phi^{\mathrm{n}}_{\mathrm{ij}}$$

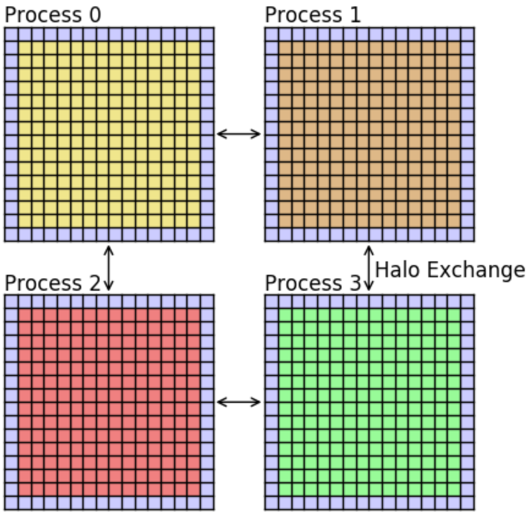
Distribute grid over MPI for parallelism





# Walls: Adding MPI to an OpenMP Code

- OpenMP only code run on COSMOS SGI UV2000 using shared memory to achieve massive parallelism
- Add MPI to halo exchange routine to achieve portability to distributed–memory systems without loosing performance.



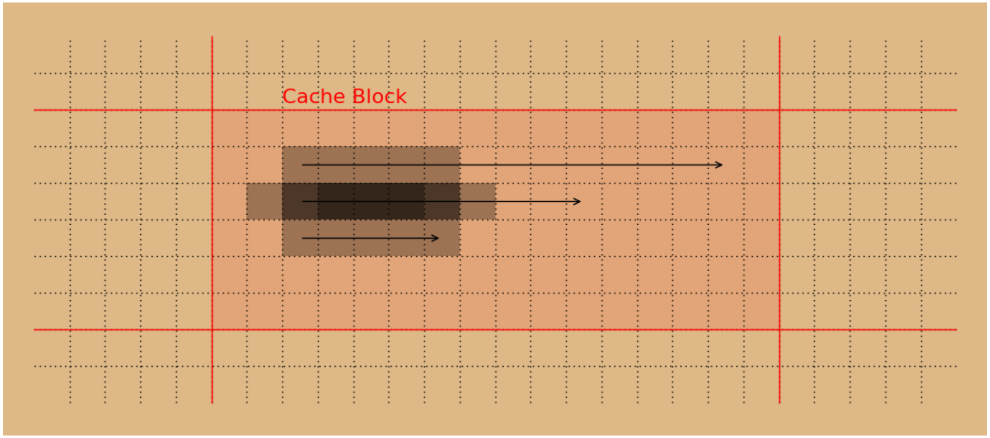

```
int jj,left,right,stat,err;
double send_left[ysize],send_right[ysize];
double recv_left[ysize],recv_right[ysize];

// Copy halo data to send buffers
for(jj=0; jj<ysize; jj++)
{ send_left[jj]  = phi[1][jj+1];
  send_right[jj] = phi[xsize][jj+1]; }

// Send and receive data left and right
MPI_Cartcoords(Cart_Comm,0,1,&left,&right);
MPI_Sendrecv(send_right,ysize,MPI_DOUBLE,
  right,rank,recv_left,ysize,MPI_DOUBLE,
  left,rank,Cart_Comm,stat,err);
MPI_Sendrecv(send_left,ysize,MPI_DOUBLE,
  left,rank,recv_right,ysize,MPI_DOUBLE,
  right,rank,Cart_Comm,stat,err);

// Copy received data to halo cells
for(jj=0; jj<ysize; jj++)
{ phi[0][jj+1]       = recv_left[jj];
  phi[xsize+1][jj+1] = recv_right[jj]; }
```

# Cache Blocking Optimization

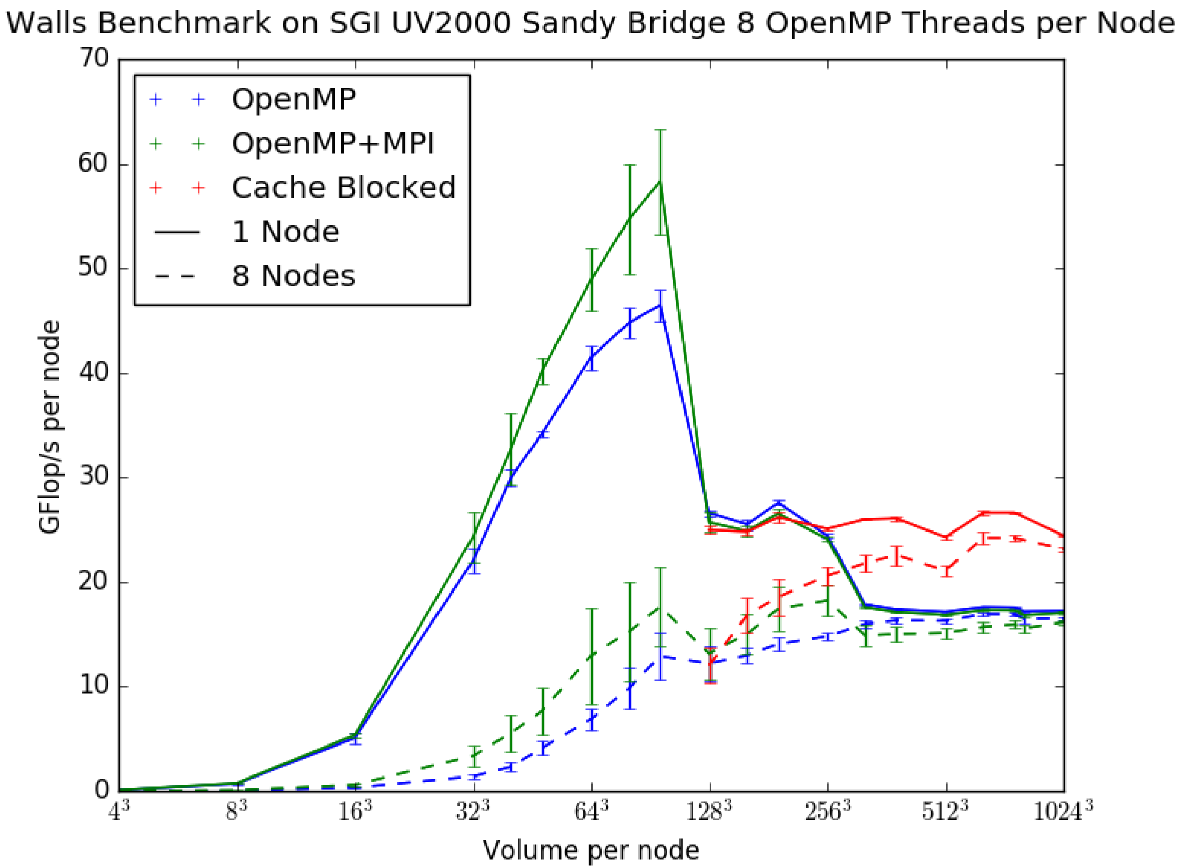



```
// New loops: Iterate over cache blocks
for(bi=0; bi<isize; bi+=bisize) {
for(bj=0; bj<jsize; bj+=bjsize) {

// Modified loops: Iterate over elements
#pragma omp for
for(i=bi; i<MIN(bi+bisize,isize); i++) {
for(j=bj; j<MIN(bj+bjsize,jsize); j++) {

// Unchanged: Loop Body
Lphi[i][j] = - 4.0*phi[i][j]
       + phi[i+1][j] + phi[i][j+1]
       + phi[i-1][j] + phi[i][j-1]; }}}}
```

# Walls: Cache Blocking Performance



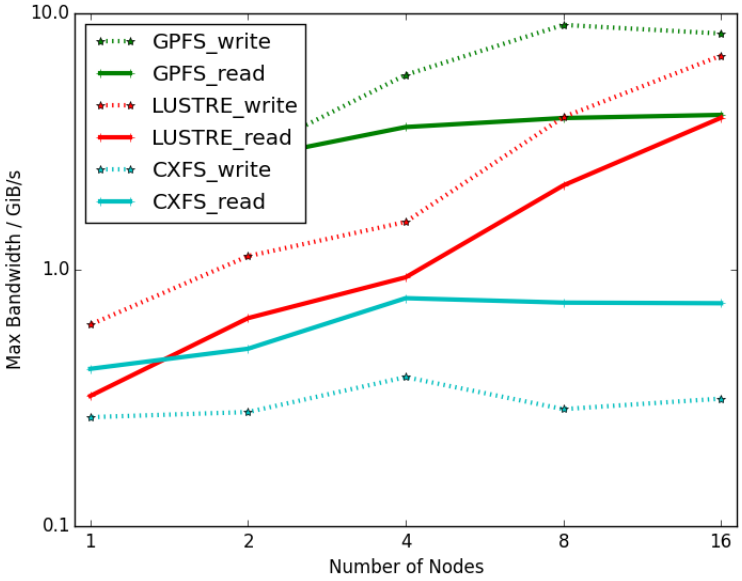
# LARE3D: Cache Blocking

- An MPI only Lagrangian–remap code for solving the equations of magnetohydrodynamics e.g. Solar Physics.
- Follow optimisations in CloverLeaf3D benchmark to add OpenMP parallelism and cache blocking.
- Many more stencils than Walls, predominant kernels are the Lagrangian stepping and remapping.
- Auto tune a single cache block size for the given memory hierarchy leading to 15% speedup for single– and multi–node runs on COSMA

# IOR Parallel File System Benchmark

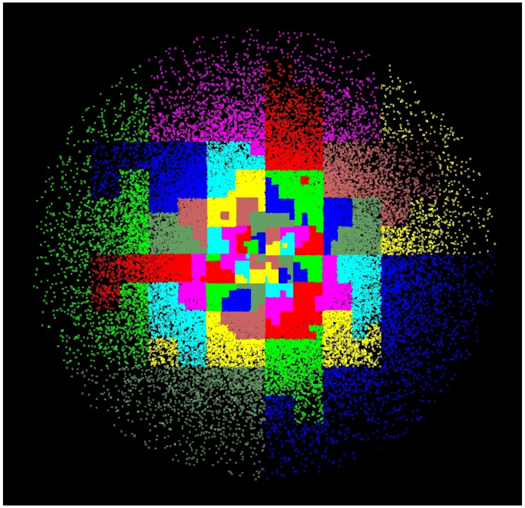
Read/write performance to disc for varied configurations:

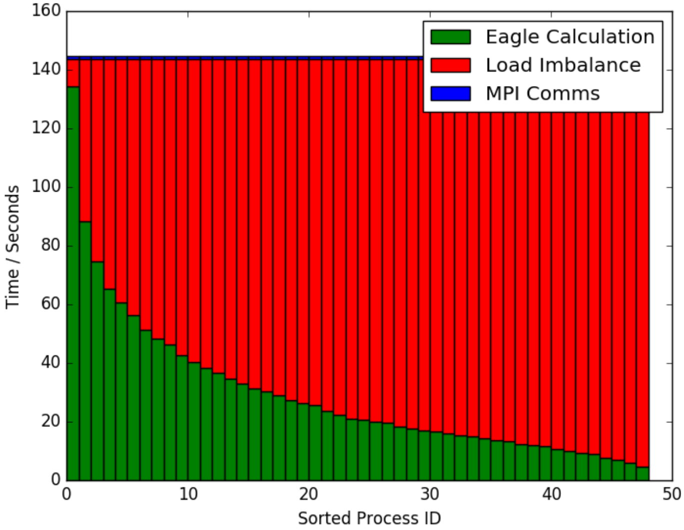
- HDF5, NetCDF, MPI–IO, Posix–IO
- Single shared file vs Multiple files
- File size, transfer size, striping ...
- Tested on GPFS and Lustre file systems on DiRAC and Archer



# Gadget: Astrophysical SPH using Octrees

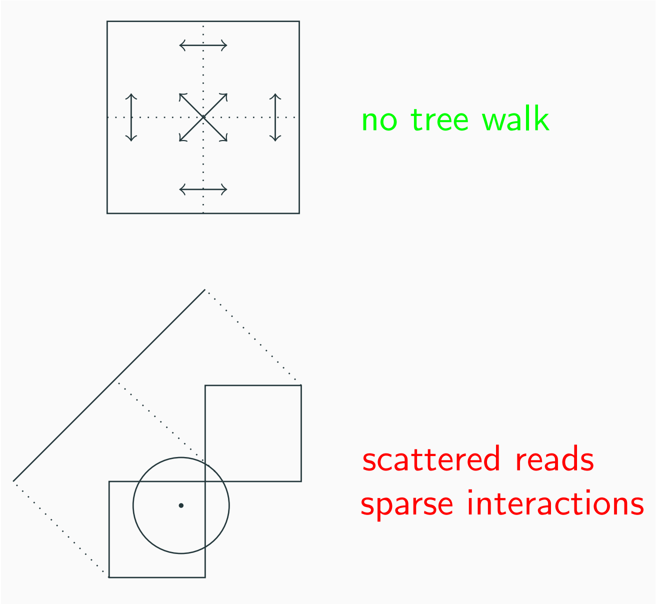
- Most time spent calculating $|r_1 - r_2| < h$ for tree walk
- Single list of all particles – hydrodynamic routines slowed processing dark matter particles
- Domain decomposition leads to severe load imbalance





# SWIFT: Vectorization

Most time spent calculating $|r_1 - r_2| < h$ for nearest neighbours




- Compared auto–vectorization and vector intrinsics
- Conversion from Array–of–Structures to Structure–of–Arrays
- "Cache" particle positions in packed arrays for $3.5\times$ speedup at the cost of a $2\times$ memory footprint overhead or $2\times$ speedup with no memory overhead.

See James Willis' Poster "Swift – Vectorisation"

# Conclusions

- DiRAC–3 hardware to be selected based on the well understood performance of real research software
- Modest efforts optimising and re–engineering existing code for specific hardware can yield fantastic performance gains

DiRAC-benchmarks/DiRAC3-testsuite