Challenges in particle tracking in turbulence on a massive scale

Dhawal Buaria [1] P. K. Yeung [1]

[1] Georgia Institute of Technology

Supported by NSF (CBET-1235906)
Supercomputing resources: NCSA (PRAC), TACC (XSEDE)

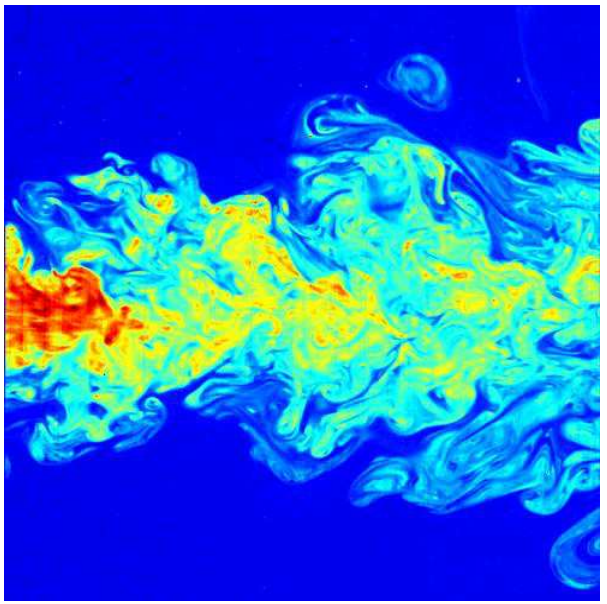
XSEDE 2014
Atlanta, GA, July 13-18, 2014

Outline

- Introduction
- Turbulence and Direct Numerical Simulations (DNS)
- Particle tracking and 3-D interpolation
- Parallel implementation
- Hybrid MPI/OpenMP implementation
- Using PGAS programming model (CoArray Fortran)
- Conclusions and Future Work

Introduction: What is Turbulence?

- Most common state of fluid motion in nature and engineering
- Characterized by disorderly fluctuations in time and 3-D space
- Wide range of non-linearly interacting scales, usually at high Reynolds number ($Re = UL/\nu$)



Turbulence and DNS

- Wide range of scales depending strongly on Reynolds number
  - domain size (L_0) larger than largest scale L
  - grid spacing (Δx) comparable to smallest scale (η)
  - classical scaling $L/\eta \sim {R_L}^{3/4}$, where $R_L = uL/\nu$
- Range of time scales: $(L/u)/\tau_\eta \sim {R_L}^{1/2}$
- Direct Numerical Simulations (DNS):
  - resolve all spatial and temporal scales
  - many grid points $(N^3 \sim {R_L}^{9/4})$ and many time steps
  - tremendous details (beyond experiments)
  - computationally very expensive (total cost $\sim {R_L}^3$)
- Need efficient numerical methods and parallel algorithms
- Data storage and postprocessing also a major challenge

Eulerian vs. Lagrangian frame of reference

Eulerian:

- Fixed observer in lab frame
- Treat fluid motion as whole and depict flow quantities as a function of position $\mathbf{x}$ and time t
- Solve the governing equations for a fixed set of grid points (N^3) and large number of time steps

Lagrangian:

- Observer follows an individual fluid 'particle' as it moves through space and time
- Fluid 'particle' has zero size and moves with local flow velocity
- Important in studying applications like pollutant dispersion, cloud physics, mixing, etc.
- Similar approach to study molecular motion, interial particles, etc.

Governing equations and Numerical approach

- DNS of isotropic turbulence in a periodic cubic domain
- The Eulerian flow velocity $(\mathbf{u}(\mathbf{x},t))$ is obtained by solving the Navier-Stokes equations with constant density $(\nabla.\mathbf{u}=0)$

$$\partial\mathbf{u}/\partial t + \mathbf{u}.\nabla\mathbf{u} = -\nabla(p/\rho) + \nu\nabla^2\mathbf{u} + \mathbf{f}$$

- Equations are solved in Fourier space using a pseudo-spectral scheme (3D-FFT as code kernel)
- Advance in time by 2nd or 4th order explicit Runge-Kutta; Δt based on CFL number (for numerical stability)
- 2-D domain decomposition, 3 X 1D-FFT, 2 X ALLTOALL : (Donzis *et al.* , Turbulence simulations on $O(10^4)$ processors, Proc. TeraGrid 2008 Conference)

Particle tracking and interpolation

- Lagrangian approach to study turbulent dispersion
- Large number of fluid particles introduced at $t = 0$ in the domain
- Integrate $d\mathbf{x}^+/dt = \mathbf{u}^+$, where $+$ denotes Lagrangian quantity
- $\mathbf{u}^+$ is the velocity at particle position $\mathbf{x}^+$: $\mathbf{u}^+ = \mathbf{u}(\mathbf{x}^+, t)$
- Use cubic-spline interpolation: 4th order accurate, twice differentiable; useful for computing velocity gradients and Laplacian also

$$\mathbf{u}^+ = \sum_k \sum_j \sum_i b_i(x^+) c_j(y^+) d_k(z^+) e_{ijk}(\mathbf{x})$$

where (b_i, c_j, d_k) are basis functions at 4 adjacent grid intervals, and (e_{ijk}) are $(N+3)^3$ Eulerian spline coefficients
(Yeung and Pope JCP 1988, Yeung and Pope JFM 1989)

Parallel Implementation for Particle Tracking

- Large number of fluid particles (up to tens of millions)
- Divided equally among all MPI tasks
- Particles wander randomly
- Neighboring grid points for interpolation typically on different task

Steps for Interpolation

- Recall: $\mathbf{u}^+ = \sum_k \sum_j \sum_i b_i(x^+) c_j(y^+) d_k(z^+) e_{ijk}(\mathbf{x})$
- Interpolation done in three steps
  1. Compute the 3-D spline coefficients (e_{ijk}) from the Eulerian velocity; involves 3 computation cyles and 2 transposes (ALLTOALLVs); independent of number of particles, can be reused
  2. Compute the basis functions (b_i, c_j, d_k) locally on each task; use MPI_ALLGATHER to collect them on all tasks (requires lot of memory, so process in smaller batches)
  3. Scan through all particles and collect partial sums on each task; MPI_REDUCE (using MPI_SUM) to collect all sums and then MPI_SCATTER to send them to respective tasks.

All steps communication dominant

Hybrid MPI/OpenMP approach

- Due to random nature of particle movement, neighboring interpolation stencil almost always on different MPI tasks
- More than 90% time in particle tracking spent on MPI communication at 16k cores (even worse for more cores)
- Hybrid MPI/OpenMP paradigms reduce latency effects
- Number of MPI tasks reduced by number of OpenMP threads; also provides more memory per MPI task
- Use MPI_THREAD_FUNNELED; master thread communicates
- Usually benefits the computational part also

Performance results on BlueWaters, NCSA

Timings on BlueWaters (BW), NCSA using 512 nodes (16384 cores)
4096^3 grid points, 16 million particles

Proc. grid	32x512	16x512	8x512
No. of threads	1	2	4
MPI_Allgather	0.847	0.704	0.428
Computations	0.236	0.117	0.071
Reduce+Scatter	1.588	1.067	0.907
Total	2.671	1.888	1.406

- Better performance with more threads (can't go above 4 threads ?)
- Performance also depends on network topology and contention; best performance when running in isolated cabinet(s) with no interference from other jobs (reservation on request)

Performance results on BW (increasing particles)

- Recall processor grid ($P = P_1$ x P_2) set by Eulerian part
- Best performance for: P_1 x no. of threads = cores on single node (32 for BlueWaters); also required that $P_1 \geq$ no. of threads
- How does the code scale with more particles ?

Timings on BW, 4096^3 grid points,
512 nodes (16384 cores), 8x512, 4 threads

No. of particles	16M	32M	64M
MPI_Allgather	0.505	0.755	1.476
Computation	0.068	0.137	0.272
Reduce+Scatter	1.012	1.759	3.478
Total	1.585	2.651	5.226

- Similar improvement on other machines.

Performance results on BW (increasing particles)

- Recall processor grid ($P = P_1$ x P_2) set by Eulerian part
- Best performance for: P_1 x no. of threads = cores on single node (32 for BlueWaters); also required that $P_1 \geq$ no. of threads
- How does the code scale with more particles ?

Timings on BW, 4096^3 grid points,
512 nodes (16384 cores), 8x512, 4 threads

No. of particles	16M	32M	64M
MPI_Allgather	0.505	0.755	1.476
Computation	0.068	0.137	0.272
Reduce+Scatter	1.012	1.759	3.478
Total	1.585	2.651	5.226

- Similar improvement on other machines.

Performance results on Stampede (TACC)

2048^3 grid points, 16M particles, 128 nodes (2048 cores)

Proc. Grid	16x128	4x128
No. of threads	1	4
MPI_Allgather	0.458	0.373
Computation	0.282	0.182
Reduce+Scatter	1.249	0.944
Total	1.989	1.499

- Going to larger problem size and more particles

  4096^3 grid points, using 1024 nodes (16384 cores) , 8x1024, 2 threads

No. of particles	16M	64M
MPI_Allgather	0.616	2.262
Computations	0.162	0.502
Reduce+Scatter	1.005	4.662
Total	1.783	7.426

Performance results on Stampede (TACC)

2048^3 grid points, 16M particles, 128 nodes (2048 cores)

Proc. Grid	16x128	4x128
No. of threads	1	4
MPI_Allgather	0.458	0.373
Computation	0.282	0.182
Reduce+Scatter	1.249	0.944
Total	1.989	1.499

- Going to larger problem size and more particles

  4096^3 grid points, using 1024 nodes (16384 cores) , 8x1024, 2 threads

No. of particles	16M	64M
MPI_Allgather	0.616	2.262
Computations	0.162	0.502
Reduce+Scatter	1.005	4.662
Total	1.783	7.426

PGAS model, Co-Array Fortran (CAF) on Cray/Gemini

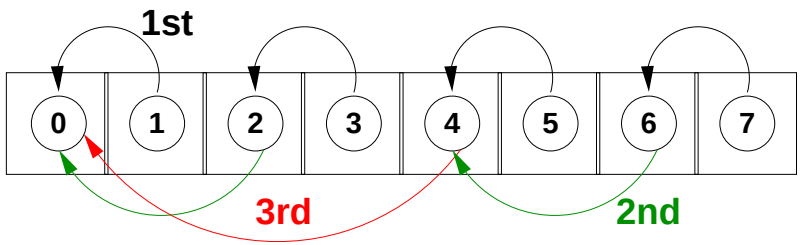
PGAS = Partitioned Global Address Space

- Instead of MPI, can use PGAS programming model to do communication
- FORTRAN + CoArrays: allocated in 'global memory address space', can be accessed by all 'images' (MPI tasks)
- Currently fully supported only on Cray compiler and Gemini network
- CAF has one-sided communication, lower latency, smaller headers
- Replaced MPI_ALLTOALL(V) calls by library routines in CAF; generated with help from consultants at NCSA/CRAY
- Library routine copies messages to/from statically allocated co-array 'buffer' on each image; breaks messages into small chunks
- Pulls chunks from other images in random order (reduces network congestion)
- Reduces overall time by $\sim 33\%$ on 4096 nodes (BW, NCSA)

Improving the Performance of the PSDNS Pseudo-Spectral Turbulence Application on BW using CAF and Task Placement, R. Fiedler *et al.*, CUG '13

CAF for REDUCE + SCATTER

- Consider the REDUCE + SCATTER model in last step of interpolation
  - too much traffic to and from one particular MPI task
  - not the most efficient pathing for data movement
- Instead can use a binary tree type algorithm, best implemented by using Co-Array Fortran (CAF)
- Images (MPI tasks) form mutually exclusive pairs to exchange information both relevant to themselves and images they subsequently communicate with

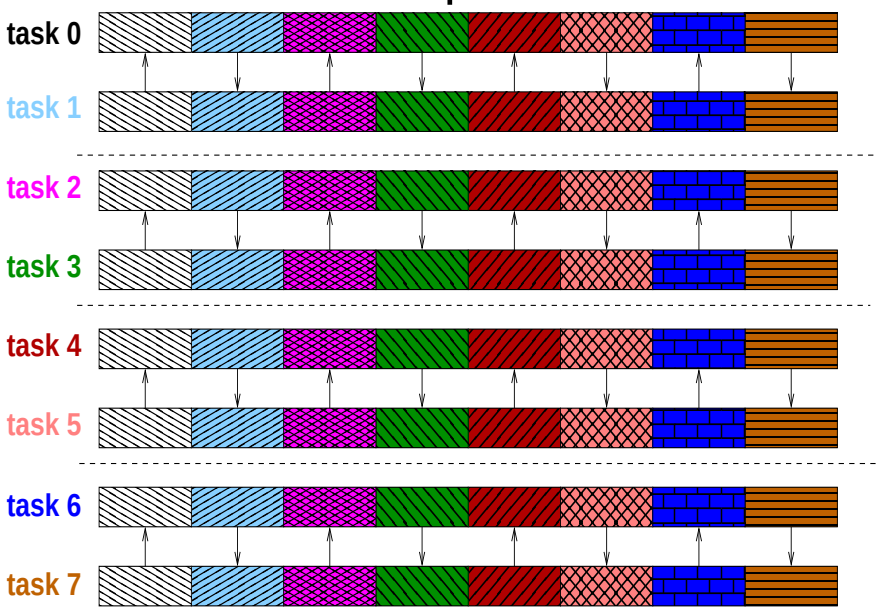


- $log_2 P$ steps required, where $P =$ total number of images; diminishing message size for each step

Schematic of the algorithm

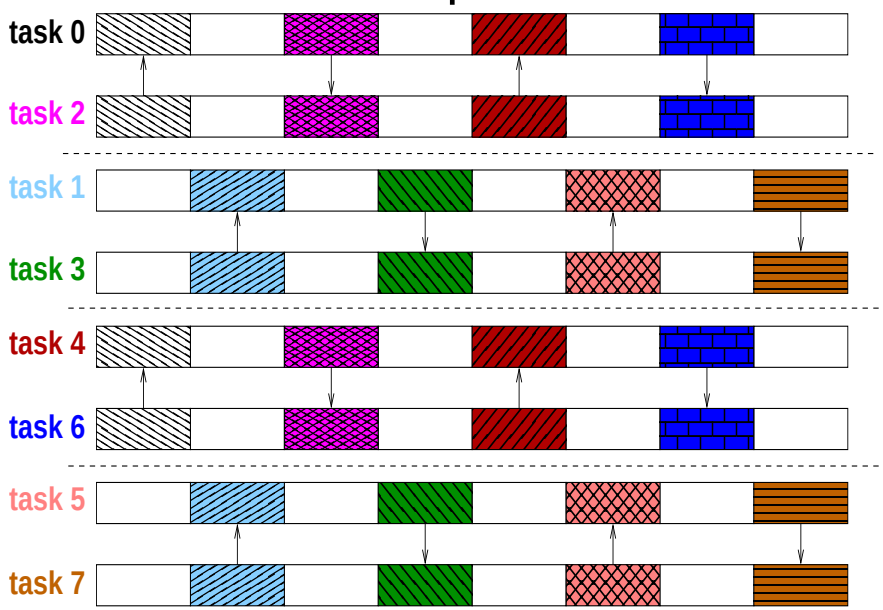
- Consider a case with $P = 8$; $log_2 P = 3$ required steps
- Arrow indicates diretcion of movement followed by summation

Step 1



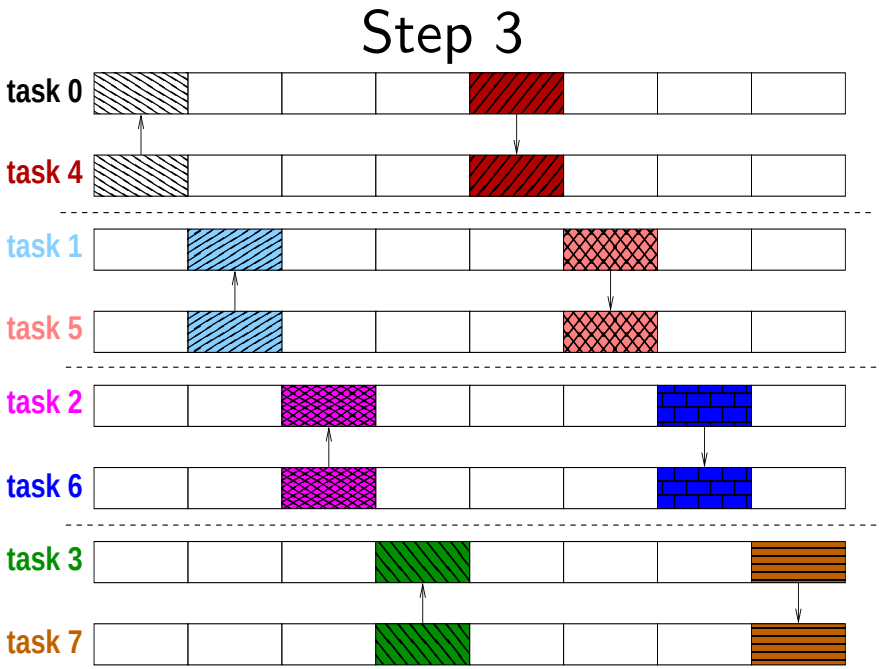
Task pairs: (0,1) (2,3) (4,5) (6,7)

Step 2

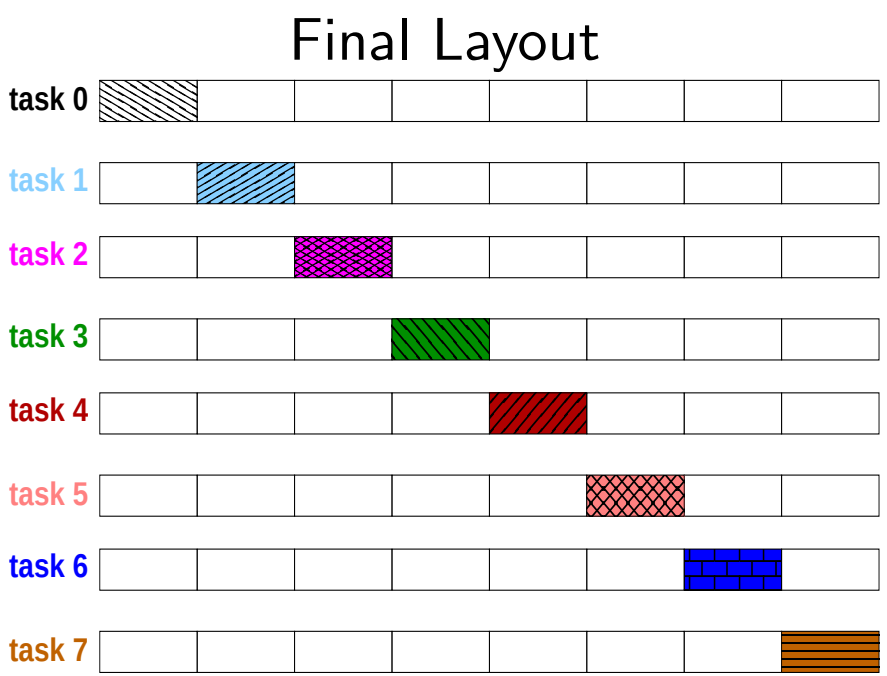


Task pairs: (0,2) (1,3) (4,6) (5,7)

Schematic of the algorithm (contd.)



Pairs: (0,4) (1,5) (2,6) (3,7)



- Communication pattern always evenly distributed; no congestion to any particular image

Performance on BW, 4096^3, 16M particles, 16k cores

	MPI	MPI	CAF	CAF
Proc. Grid	32x512	8x512	32x512	8x512
No. of threads	1	4	1	4
MPI_Allgather	1.810	0.861	1.789	0.833
Computation	0.498	0.144	0.488	0.144
Reduce+Scatter	3.648	1.956	2.252	0.905
Total	5.956	2.961	4.529	1.882

- 4X speedup for REDUCE+SCATTER using OpenMP and CAF

Conclusions and Future Work

- Hybrid MPI/OpenMP implementation for parallel interpolation helps to improve both communication and computation
  (though code is communication dominant)
- Overall time reduced by almost 50% on BW when using 4 threads
- Scales well with increased number of particles
- Further improvement in communication timings by using CAF

Future Work:

- Extend scaling to even larger problems. Working on $8192^3/16$M particles using 8192 nodes (262144 cores) on BW
- Working on alternate algorithm to do the interpolation, where each set of particles go through each task one by one in a pipelined fashion
- Extend to utilize MIC architecture on Stampede
  (recently applied for a ECSS project)