An asynchronous algorithm for massive pseudo-spectral simulations of turbulence on Summit

K. Ravikumar[1], D. Appelhans[2], P.K. Yeung[1], M.P. Clay[1]

[1]Georgia Institute of Technology, [2]IBM Research

Fluid Turbulence: Overview and Challenges

- Disorderly fluctuations in time and 3D space, over a wide range of scales (which increases with the Reynolds no.)
- A search for scale similarity, with energy cascade from large scales to small scales (via intermediate scales)
- Some fluctuations can be extreme (intermittent), requiring better resolution than commonly thought/practiced [1]
- Agent of efficient mixing and dispersion, may be coupled to other phenomena (buoyancy, chemical reaction, etc)
- Our emphasis: fundamental understanding in simplified geometries, yet of general relevance to applications

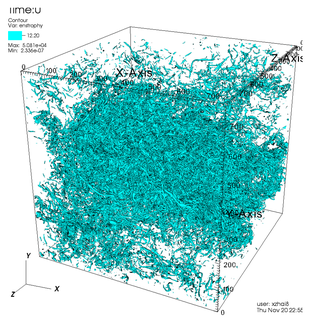


Figure: Vortex filaments in 8192^3 direct numerical simulation

Governing equations and numerical methods

- Navier Stokes, for conservation of mass and momentum:

$$\nabla \cdot \mathbf{u} = 0 \quad (1)$$

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

- Fourier pseudo-spectral methods using FFT for 3D domain with periodic boundary conditions
- Grid spacing should be comparable to, preferably smaller than smallest scales (Kolmogorov)
- 2nd or 4th order Runge-Kutta in time (wavenumber space)
- Time step based on Courant number for numerical stability

Major Algorithmic Elements

- Domain decomposition: can be 1D (slabs) or 2D (pencils)
- One FFT per direction with data local to each MPI process
- Transpose using alltoall communication (or variants)
- Pack and unpack in local memory before/after alltoall

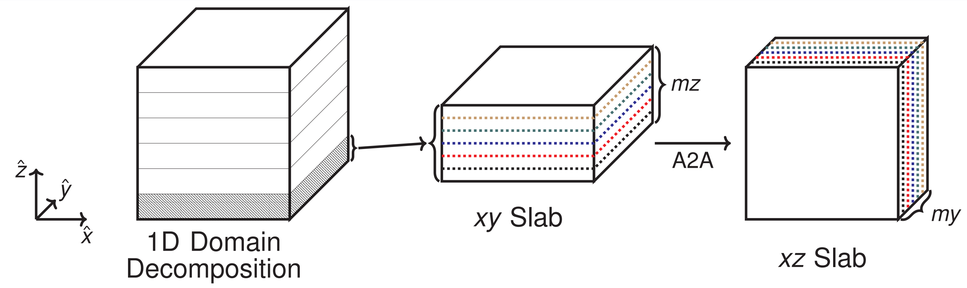
Target on Summit: 16384^3 or higher

- Fat nodes offer large memory: 1D decomposition
- Fast CPU-GPU data transfer through NVLINK
- Spectrum-MPI for communication: 1-sided is best
- Fine-grained overlapping among CPU/GPU computations, NVLINK transfers and non-blocking alltoalls

IBM XL compiler, CUDA Fortran[2] for host-device data copies, CUDAFFT on GPUs

- Detailed profiling using NVPROF
- Measurements of network bandwidth
- Small reproducers helped diagnose bugs in system

Domain Decomposition & Algorithm



Synchronous Algorithm

H2D : ($nv = 3$)

↓

FFT : $\kappa \rightarrow x$ in $\hat{y}$ ($nv = 3$)

↓

MPI : Pack, A2A, Unpack ($nv = 3$)

↓

FFT : $\kappa \rightarrow x$ in $\hat{x}, \hat{z}$ ($nv = 3$)

↓

Non-linear products ($nv = 3 \rightarrow 5$)

↓

FFT : $x \rightarrow \kappa$ in $\hat{x}, \hat{z}$ ($nv = 5$)

↓

Derivatives wrt $\hat{x}, \hat{z}$ ($nv = 5 \rightarrow 4$)

↓

MPI : Pack, A2A, Unpack ($nv = 4$)

↓

FFT : $x \rightarrow \kappa$ in $\hat{y}$ ($nv = 4$)

↓

Convective terms ($nv = 4 \rightarrow 3$)

↓

Update velocity field ($nv = 3$)

↓

D2H : ($nv = 3$)

Asynchronous Algorithm

nplanes : $1 \rightarrow mz$

H2D | FFT : $\kappa \rightarrow x$ in $\hat{y}$ | D2H | Pack, A2A, Unpack

nplanes : $1 \rightarrow my$

H2D | FFT : $\kappa \rightarrow x$ in $\hat{x}, \hat{z}$ Non-linear products FFT : $x \rightarrow \kappa$ in $\hat{x}, \hat{z}$ Derivatives wrt $\hat{x}, \hat{z}$ | D2H | Pack, A2A, Unpack

nplanes : $1 \rightarrow mz$

H2D | FFT : $x \rightarrow \kappa$ in $\hat{y}$ Convective terms Update velocity field | D2H

Limitations of synchronous algorithm

- Problem size restricted by available GPU memory
- Computations and NVLINK cost add to runtime

Advantages of asynchronous algorithm

- Run larger problem using CPU memory
- Overlapping Compute and NVLINK under MPI lowers cost
- Network, being the bottleneck, is continuously used

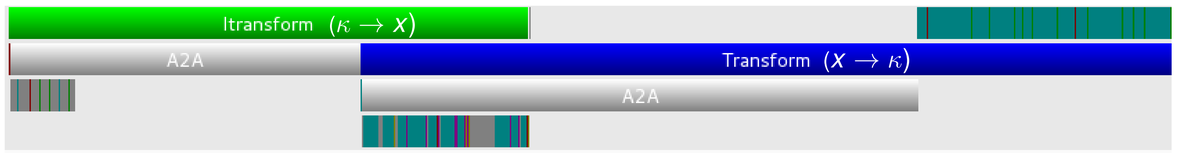


Figure: Nvprof timeline of asynchronous code for one Runge-Kutta substep. Alltoall (silver) overlapped with GPU computations and NVLINK (shown in other colors). Further operations have to wait on previous alltoall to complete making it the bottleneck.

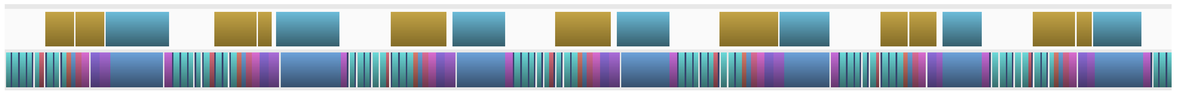


Figure: Overlapping GPU compute and NVLINK data transfers in asynchronous code. First row: Transfer stream (NVLINK), Second row: Compute stream. GPU is continuously used for computations. Data transfers are performed as soon as computations on previous plane are completed

Implementation using CUDA Fortran

- Compute and Transfer streams created
- Non-blocking copy between host and device using *cudaMemCpyAsync* in Transfer stream
- cudaFFT and other computes queued in Compute stream
- Synchronization between streams enforced using *cudaEventRecord* and *cudaStreamWaitEvent*

Proposed use of OpenMP 4.5

- *use_device_ptr* clause to call cudaFFT functions
- *DEPEND* and *NOWAIT* clauses to mimic CUDA *streams*, *events* and *asynchronous* execution

Scaling and Runtime Performance

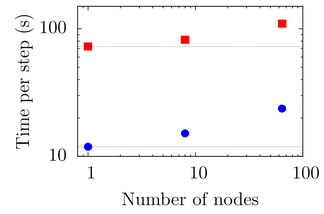


Figure: Scaling of GPU (blue circle) and CPU (red square) code on Summit. Weak scaling of CPU and GPU code for the 64-node problem is 66.35% and 50.23% respectively.

# Nodes	Problem Size	A2A (% total time)	Speedup
1	1536^3	9.31 (78.16%)	6.11
8	3072^3	12.40 (81.87%)	5.42
64	6144^3	20.71 (87.40%)	4.63

Table: Percentage of total time spent on alltoall by the GPU code and speedup of the GPU code compared to the CPU code. Recent modifications to GPU code give speedup $> 5X$ for 1 node problem

Network performance on Summit

- Max achieved BW (R+W), $20GB/s$ for larger problems (Theoretical max 46GB/s)
- A2A using one-sided MPI tested (gives higher BW)
- In conversation with IBM spectrumMPI team
- Plan to implement hierarchical A2A using non-blocking GATHER and SCATTER

Conclusions & Future Work

- Process parts of plane instead of full plane at a time
- Port cudaFortran code to OpenMP 4.5
- One-sided MPI for better network performance

Acknowledgments & References

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. We are grateful for the dedicated assistance from O. Hernandez, R. Budiardja and other staff members from OLCF and IBM.

[1] P. K. Yeung, X. M. Zhai, and K. R. Sreenivasan *PNAS*, vol. 112, pp. 12633–12638, 2015.

[2] G. Ruetsch and M. Fatica, *CUDA Fortran for Scientists and Engineers Elsevier, 2013.*