Fourier pseudo-spectral algorithms for turbulence on leadership-class GPU platforms

P.K. Yeung (Georgia Tech), pk.yeung@ae.gatech.edu
Kiran Ravikumar (Sci. & Tech. Corp.), kiran.ravikumar@nasa.gov

w/ contributions from S. Nichols (ORNL), D. Appelhans (NVIDIA),
R. Uma-Vaideswaran, D. Dotson, B. Santiago (GT)

Advanced Modeling and Simulation (AMS) Seminar Series
NASA Ames Research Center
June 27, 2024 (Virtual)
CAAR Program for Frontier at Oak Ridge Leadership Computing Facility
INCITE 2023 and 2024 Awards at OLCF: total 2.4 M node hours on Frontier
NSF, subaward via The Johns Hopkins Univ. (Grant 2103874, C. Meneveau (PI))

The Challenges of Fluid Turbulence

Disorderly fluctuations: unsteady, 3D, nonlinear, wide range of scales

— great complexity, many applications in nature and engineering
— extreme weather, wing-tip vortices, pollution, disease transmission, etc.



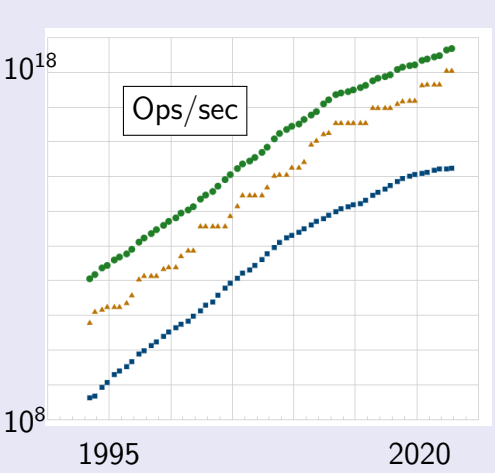



A quest for fundamental understanding, driven by curiosity and need

The growth of computational power: top500.org

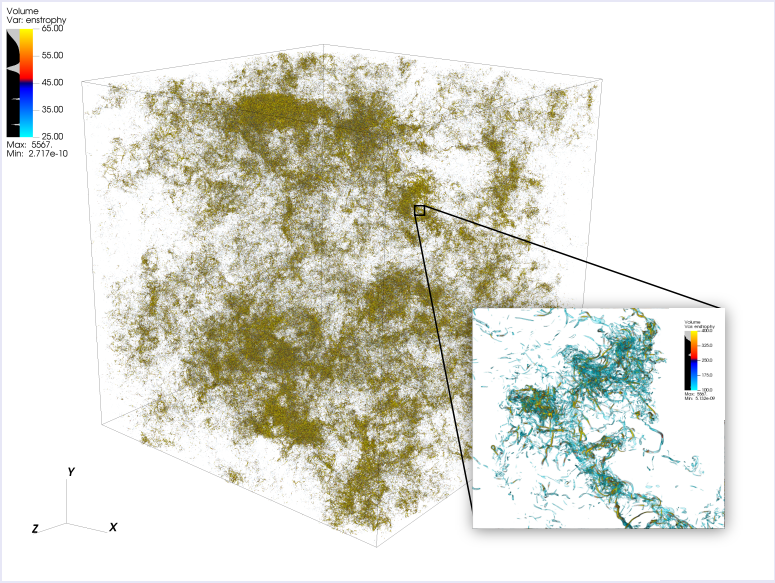
Nearly exponential increase for some 30 years, factor of 10^6 1995 to 2022.

- Massive distributed parallelism is vital, yet brings challenges in scalability
- Heterogeneous architectures, such as GPUs, are now pre-dominant.
- Exascale (10^{18} Ops/sec in theoretical peak) first reached in Nov 2022
- Adaptability to new programming models is crucial for best outcomes
- Data movement — not covered in numerical analysis textbooks!



Current #1: "Frontier" at US DOE ORNL

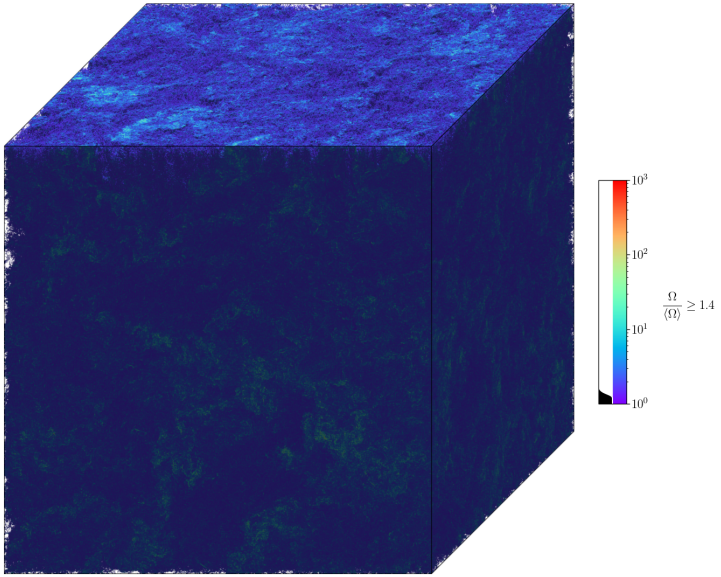
Volume rendering of a 3D $12{,}288^3$ enstrophy field (Summit)



Zoom-in shows high intensity region, with $100 \leq \Omega/\langle\Omega\rangle \leq 400$

Courtesy of David Pugmire, Michael Matheson (Oak Ridge Natl. Lab.)

Shaded volume rendering of enstrophy, 4096^3, with zoom-in



An Outline of This Talk

- Pseudo-spectral methods for the Navier Stokes equations
- Batched Asynchronism algorithm on Summit, up to 18432^3 resolution
- GPU algorithm and performance on Frontier, up to 32768^3 resolution
- Science results: energy spectrum and "intermittency corrections"
- Particle tracking (for Lagrangian or point-particle studies)
- Some concluding remarks

Direct Numerical Simulation (DNS) on a 3D Periodic Domain

- Incompressible N-S eqs. for velocity fluctuations, with no mean velocity

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

- Fourier pseudo-spectral (traceable back to Rogallo 1981 at NASA Ames)
  – Start with $\hat{\mathbf{u}}(\mathbf{k})$ in wavenumber space, transform to physical space, form nonlinear terms, transform back, differentiate, advance in time (explicit Runge-Kutta).
- Need to capture both largest (ℓ) and smallest (η) scales in the flow
  - Linear domain size (L) should be several times larger than ℓ
  - Grid spacing smaller than or comparable to Kolmogorov scale $\eta = (\nu^3/\langle\epsilon\rangle)^{1/4}$
  - In each direction: $N = L/\Delta x \propto \ell/\eta$, increasing strongly with Reynolds number
- Resolution requirements increase further if we are to study extreme events
- Prior to Frontier: 18432^3 on IBM/NVIDIA *Summit*, using CUDA Fortran and "batched asynchronism" (K. Ravikumar *et al.*, Supercomputing Conference 2019)

3D Discrete Fourier Transform

- With periodic B.C., any field variable f can be decomposed into contributions from different scale sizes. Forward: $\{f(\mathbf{x})\} \rightarrow \{\hat{f}(\mathbf{k})\}$, Inverse: $\{\hat{f}(\mathbf{k})\} \rightarrow \{f(\mathbf{x})\}$

$$\hat{f}(\mathbf{k}) = \sum_{\mathbf{x}} f(\mathbf{x}) \exp(-i\mathbf{k}\cdot\mathbf{x})\ d\mathbf{x} \ \longleftrightarrow\ f(\mathbf{x}) = \sum_{\mathbf{k}} \hat{f}(\mathbf{k}) \exp(i\mathbf{k}\cdot\mathbf{x})\ d\mathbf{k},$$

  $\mathbf{k} = (k_x, k_y, k_z)$ is the wavenumber vector, $\hat{f}$ is complex-valued Fourier coefficient.

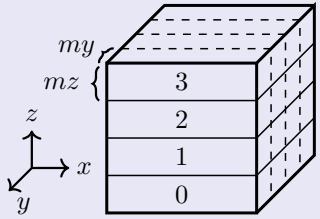
- One direction at a time, using highly optimized FFT libraries, within factor of $(\Delta x)^3$

$$\hat{f}(k_x, k_y, k_z) = \sum_{y} \left\{ \sum_{z} \left\{ \sum_{x} f(x, y, z) \exp(-ik_x x) \right\} \exp(-ik_z z) \right\} \exp(-ik_y y)$$

  provided complete lines of data are in the local memory (which is the hard part!)

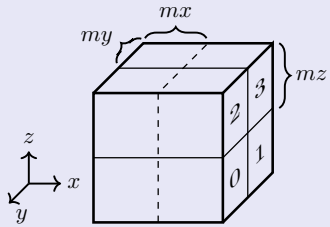
Domain Decomposition: 1D or 2D?

Sub-divide in 1D: each parallel process handles 1 slab (P up to N)



- FFT in x and y; transpose into $x-z$ slabs; FFT in z
- "All-to-all" communication among all P processes

Sub-divide in 2D: each parallel process handles 1 pencil ($P_r \times P_c$, up to N^2)



- FFT in x; transpose; FFT in z; transpose; FFT in y
- Communication within row and column sub-communicators

Slabs: if memory per node allows and largest problem size satisfies $P \leq N$

Pencils: if largest problem needs $P > N$, care needed in choosing $P_r \times P_c$

Using GPUs (dominant form of heterogeneous computing)

GPUs have 100s or 1000s of threads, and can compute very fast

- Usually, several GPUs are attached to each single CPU node
- Used as accelerators: the threads need to divide up the work
- Data copied from host to device for compute, back when done
- Communication still needs to be performed, on CPU or GPU

Size of GPU memory has a major effect

- Small GPU memory: data needs to be moved from CPU to GPU in batches that fit, and back to CPU when done. Overhead from host $\longleftrightarrow$ device copying. Can different operations on different batches of data operate simultaneously?
- Large GPU memory: send data to GPU, let GPU do everything, Goal is to minimize host $\longleftrightarrow$ device transfers. Can we do communication effectively on GPUs too?

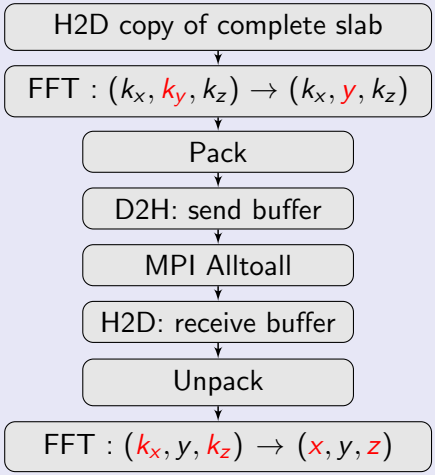
Using Summit at Oak Ridge National Lab

- 4600 nodes each with 2 IBM POWER9 CPUs, 44 cores, and 6 NVIDIA Volta GPUs
- 512 GB host memory per node
  16 GB per GPU: large problem size
  - 1D (slab) decomposition preferred
- NVLINK and Fat-tree Infiniband interconnect
- Software:
  - IBM XL compiler, CUDA Fortran
  - NVIDIA's cufft library: fast 1D FFTs on GPUs



Objective: run the largest problem size possible

Synchronous GPU algorithm using 1D decomposition



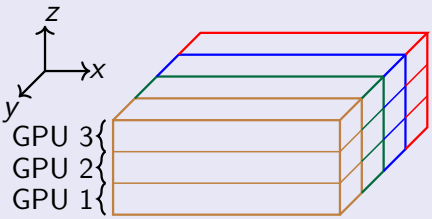
- Copy entire slab from CPU (host (H)) to GPU (device (D)) and back to CPU at end
- 1D FFTs in y, z, x directions using cufft library
- Pack and unpack data on GPU: faster than CPU
- MPI Alltoall among all tasks to transpose $x-y$ to $x-z$ slabs
- D2H and H2D copies of send and receive buffers before and after Alltoall
- Similar operations to transform back to wavenumber space from physical space

Large problem that may not fit on GPU? Can asynchronism help?

Batched asynchronous algorithm

Run large problem sizes efficiently without being limited by GPU memory

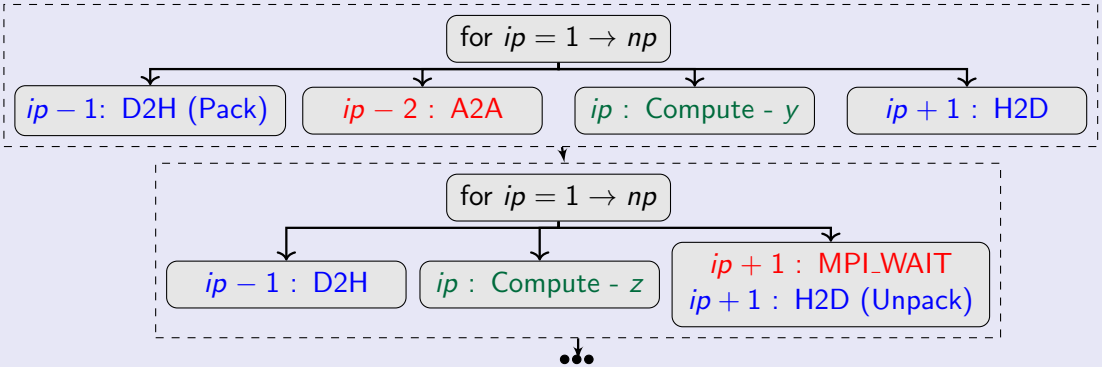
- Divide slab into np pencils ($N \times nyp \times mz$) and process each pencil separately ($nyp = N/np$)
- Overlap operations on different pencils to hide some data transfer and compute costs
- Multiple GPUs: work on sub-division of pencils



Asynchronous execution using CUDA streams and events

- One compute stream and one data transfer stream
  — compute and data transfers can occur simultaneously
- CUDA events are used to enforce synchronization between streams

Batched asynchronism: Illustrated via operations in y and z



- Operations on same row executed asynchronously but launched from left to right
- Pack and unpack: strided data copy to avoid reordering data before transpose
- Non-blocking all-to-all allows overlap. Call MPI_WAIT before compute
- Optimize strided copies and larger message sizes in all-to-all using fewer MPI tasks

Batched asynchronous code performance

- Performance data collected on Summit
  — 2nd order Runge Kutta, 3 inverse and 5 forward transforms, 2 substages per time step

Nodes	Problem Size	Time(s)			
		Sync CPU (Pencils)	Async GPU		
			6 tasks/node	2 tasks/node	
				1 pencil/A2A	1 slab/A2A
16	3072^3	34.38	8.09	6.70	7.50
128	6144^3	40.18	12.17	8.66	8.07
1024	12288^3	47.57	13.63	12.62	10.14
3072	18432^3	41.96	25.44	22.30	14.24

- 2 tasks/node performs better than 6 task/node for all problem sizes tested
- 128 nodes and above: 1 slab/A2A better than 1 pencil/A2A
  — suggests better overall performance without MPI overlapping GPU operations
- $18{,}432^3$: $\sim$ 3X speedup to pencils CPU version; communication bound code
- More details: K. Ravikumar *et al.*, Supercomputing Conference 2019

"Frontier" at OLCF



The first true Exascale machine, by HPE-AMD. No. 1 in the world, since June 2022

Frontier: machine characteristics and our core strategies

The most important features to us

- Over 9400 nodes, 512 GB memory per node, same for both CPU and GPU
- Each node has 4 AMD MI250X GPUs, but can function as 8 GPUs per node
- HPE Slingshot for communication, network cards built into the GPUs
- Orion Lustre filesystem, max aggregate data rate around 10 TB/sec

To solve biggest problem and get best performance on this machine

- Reduce memory footprint: alltoall 1 variable at a time will allow smaller buffers
- Let GPU do (almost) all the work: minimize host-device data transfers
- OpenMP offloading for compute loops, including AMD FFT library calls
- 8 MPI tasks per node; 1 GPU per MPI simplifies programming
- MPI calls made directly on GPUs, avoiding data copying before and after

Using GPUs: The OpenMP Programming Model

- Fortran 90 base code, with Fortran-C interoperability support
- OpenMP is directive-based, active inside "Parallel Regions",

  starting with "OMP TARGET DATA MAP (....)"
  and ending with "OMP END TARGET DATA".
- TARGET refers to the GPU
- DATA MAP (.....) defines mapping between specified arrays on CPU and GPU, and whether data are copied from host to device at start, or device to host at end
- OMP TARGET UPDATE FROM (array) can cause data to be copied explicitly from device to host, e.g for output or checks on CPU
- Other tools (e.g OpenACC) available, but OMP is well supported, with a standard

Code fragment for OpenMP offload on GPU

Map data from CPU to GPU, then divide work among many (thousands) GPU threads, getting results back on CPU at the end of a large "parallel region".

```
1   !$OMP TARGET DATA MAP (TOFROM: ....)
2         ......................
3   !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO COLLAPSE(4) &
4   !$OMP       DEFAULT(NONE) PRIVATE(i,xp,yp,zp,y) &
5   !$OMP       SHARED(jproc,yjsz,isz, zjsz , jjst , iv , sndbuf_col , source)
6   do i=1,jproc
7       do zp=1,zjsz
8           do yp=1,yjsz
9               do xp=1,xisz
10                  y=jjst(i-1)+yp-1
11                  sndbuf_col(xp,yp,zp,iv,i) = source(xp,y,zp,iv)
12              end do
13          end do
14      end do
15  end do
16  !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO
17        ......................
18  !$OMP END TARGET DATA MAP
```

Code fragments for MPI on CPU vs GPU

Since 'all' calculations are done on GPU, MPI via CPU will involve extra copying from device to host before MPI and from host to device after (the OMP TARGET UPDATEs)

```
1  if (gpumpi.eq.0) then
2      !$OMP TARGET UPDATE FROM(sndbuf_row)
3      call MPI_ALLTOALL (sndbuf_row, xisz*zisz*yjsz, mpicomplex, &
4         rcvbuf_row, xisz*zisz*yjsz, mpicomplex, mpi_comm_row, ierr)
5      !$OMP TARGET UPDATE TO(rcvbuf_row)
6   else
7      call c_f_pointer (sndbuf_row_buffer,sndbuf_row,[xisz,zisz,yjsz,nv,iproc])
8      !$OMP TARGET DATA USE_DEVICE_PTR(sndbuf_row, rcvbuf_row)
9         call MPI_ALLTOALL (sndbuf_row, xisz*zisz*yjsb, mpicomplex, &
10           rcvbuf_row, xisz*zisz*yjsb, mpicomplex, mpi_comm_row, ierr)
11     !$OMP END TARGET DATA
12  end if
```

Frontier node design: GPU-aware MPI is fast, uses exactly the MPI alltoall arguments. Buffers need to be declared as device pointers.

Analyzing code performance

- Elapsed wall time per non-output time step, split into (i) 1-D FFTs, (ii) packing/unpacking for alltoall; (iii) alltoall, and (iv) other compute operations
- Test, say M steps, N^3 problem, P parallel processes. Take average or minimum?
  - time per step is like a random variable with lower bound
  - the minimum (or "best time step") converges faster with increasing M
  - variability due to network contention possible at large N and large P
- Strong Scaling: ideally, time $\propto 1/P$ at fixed N
  - important for faster turnaround, or to accommodate memory for other needs
- Weak Scaling: ideally, time unchanged if $P \propto N^3$
  - important for enabling higher resolution ("more science")

Weak Scaling of Pencils code (journal paper in revision)

2048^3 is special (largest problem size that fits into a single node, fastest MPI)

Case	N^3	P_r	P_c	FFT	P+U	R-MPI	C-MPI	Other	Total	WS1(%)
1	2048^3	2	4	1.81	1.08	0.74	1.27	0.58	5.48	
2	2048^3	1	8	1.65	0.66	0.00	1.83	0.42	4.55	
3	4096^3	8	8	2.03	1.09	2.08	5.97	0.82	11.98	
4	4096^3	4	16	1.77	0.89	1.52	5.95	0.52	10.64	
5	4096^3	2	32	1.89	0.74	0.74	6.86	0.45	10.68	
6	4096^3	1	64	1.78	0.43	0.00	7.00	0.43	9.62	
9	8192^3	2	256	1.46	0.60	0.74	8.15	0.43	11.39	101.6 [5]
10	8192^3	1	512	1.56	0.40	0.00	8.14	0.43	10.52	99.1 [6]
13	16384^3	2	2048	2.34	0.59	0.74	8.77	0.48	12.92	96.5 [5]
14	16384^3	1	4096	2.57	0.40	0.00	8.32	0.43	11.71	95.8 [6]
17	32768^3	2	16384	3.41	0.60	0.74	9.72	0.49	14.95	88.9 [5]
18	32768^3	1	32768	3.76	0.40	0.00	8.50	0.46	13.10	91.7 [6]

(Rough) assessment of GPU vs CPU speedup

Test on the same hardware, and isolate benefit of GPU by not using it
(e.g. work of GPU threads now borne by OpenMP threads on the CPU)

Case	N^3	#Nodes	P_r	P_c	FFT	P+U	MPI	Other	Total
1	2048^3	1	2	32	23.1	37.5	16.3	27.4	104.3
2	2048^3	1	8	8	19.5	33.5	14.8	28.1	95.9
3	2048^3	1	64	1	19.5	36.1	18.0	29.5	103.1
4	4096^3	8	64	8	19.3	35.0	21.8	30.9	107.0
5	4096^3	8	8	64	22.1	36.9	21.0	30.9	109.7
6	8192^3	8	64	64	20.9	33.3	22.6	28.1	104.9

GPU code is roughly 10X faster: 13X for FFT, 33X for P+U, 3X for MPI, 65X for other compute. Current implementation of OpenMP offloading is highly efficient.

Further analysis of MPI timings

Small MPI kernel with same communication pattern as DNS code:

- Timings for one variable shown here. Close match with DNS code when multiplied by number of variables being passed to alltoall per time step
- Column-communicator point-to-point (P2P) message size $\propto N^3/P/P_c$

N^3	Nodes	P_c	P2P (MB)	H2D/D2H Copy (sec)	CPU MPI (sec)	CPU Tot (sec)	GPU MPI (sec)
2048^3	1	4	1024	1.241	0.444	1.685	0.092
4096^3	8	32	128	1.185	0.660	1.845	0.539
8192^3	64	256	16	1.224	0.689	1.913	0.580
16384^3	512	2048	2	1.223	0.826	2.049	0.626
32768^3	4096	16384	0.25	1.231	0.848	2.079	0.705

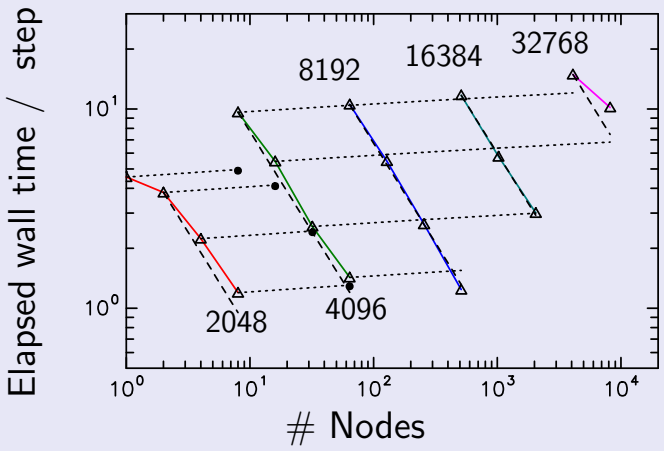
Significant overhead due device-to-host and host-to-device transfers if performing MPI on CPU (longer than actual transfer) is avoided for direct MPI on GPU.

Strong Scaling at 16384^3

Case	Q	P_r	P_c	FFT	P+U	R-MPI	C-MPI	Other	Total	SS1(%)
1	1	8	512	2.09	0.66	2.08	9.26	0.53	14.61	–
2	1	8	1024	1.05	0.33	1.05	4.36	0.26	7.05	103.6
3	4	8	2048	0.53	0.17	0.53	2.27	0.13	3.62	100.9
4	1	4	1024	2.19	0.61	1.51	9.04	0.49	13.85	–
5	1	4	2048	1.19	0.30	0.76	4.16	0.24	6.57	105.4
6	4	4	4096	0.55	0.15	0.38	2.21	0.12	3.42	101.3
7	1	2	2048	2.34	0.59	0.74	8.77	0.48	12.92	–
8	1	2	4096	1.18	0.29	0.37	4.27	0.24	6.36	101.6
9	4	2	8192	0.59	0.15	0.19	2.26	0.14	3.32	97.3
10	1	1	4096	2.57	0.40	0.00	8.32	0.43	11.71	–
11	1	1	8192	1.28	0.22	0.00	4.06	0.20	5.76	101.6
12	4	1	16384	0.65	0.10	0.00	2.02	0.12	2.90	100.9

Q is number of variables per column-communicator alltoall. Usually $Q = 1$.
But $Q = 4$ (if memory still fits) helps with "message-too-small" scenarios.

Overall scalability: 2048^3 to 32768^3



Only 2048^3 *(single node) and* 32768^3 *(max problem size) appear to deviate.*

Comparing performance: Summit and Frontier

- Figure of Merit: Number of grid points updated per second (N^3/t)

	Summit	Frontier
Problem Size	18432^3	32768^3
Number of nodes	3072	4096
time/step (s)	14.24	13.10
Theoretical Flops (PFLOPS)	134	871
FOM (N^3/t in 10^9)	440	2686

- Theoretical flops on Frontier on 4096 nodes is 6.5x that on Summit on 3072 nodes
- Achieved FOM on Frontier is 6.1x that on Summit

Statistical Stationarity and Reynolds Number

Forced isotropic turbulence, hoping reaching stationarity ASAP:

- ASAP here means reducing cost in supercomputer resources
- If reliable data on $E(k)$ at low wavenumbers are available: maintain energy in large scales while turbulence still evolves (Donzis & Yeung 2010).

Obtaining progressively higher Reynolds numbers

- Start with some older datasets, freeze spectrum at lowest few wavenumber shells.
- Go to a finer grid. Reduce the viscosity to target a new value of the Kolmogorov scale $\eta = (\nu^3/\langle\epsilon\rangle)^{1/4}$. The flow will adjust, mainly at the small scales.
- Repeat the process. Goal is to minimize number of time steps that must be run at the highest resolution. The Reynolds number increases in stages.
- If desired, hold the viscosity but improve resolution in both time and space for better accuracy, especially in studies of intermittency (Y, Sreenivasan & Pope 2018)

Intermittency and intermittency corrections

Extreme events of intense activity, localized in time and space

- What is the probability of fluctuations exceeding a certain level?
- Need to resolve small scales well, in both time and space
- Observed for energy dissipation, enstrophy, scalar dissipation.

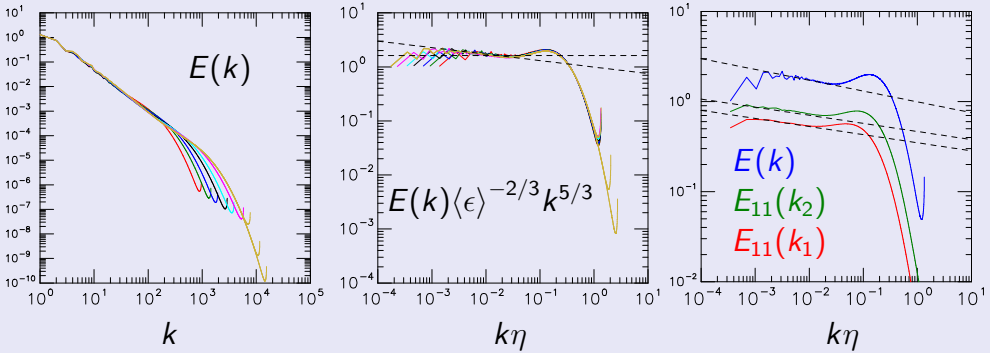
Kolmogorov similarity theory and corrections for intermittency

- 1941: at sufficiently high Re, small scales are isotropic and universal
- 1941: intermediate scales only depend on scale size and mean dissipation rate
- 1962: account for fluctuations of energy dissipation: e.g, in the relation $E(k) \propto \langle\epsilon\rangle^{2/3} k^{-5/3}$ the factor $\langle\epsilon\rangle^{2/3}$ needs to be replaced by $\langle\epsilon_r^{2/3}\rangle$, where ϵ_r is a local average. $r \sim 1/k$, which leads to change in the -5/3 exponent.

DNS: R_λ now comparable to most laboratory experiments

Early Science Results: Energy Spectrum at R_λ up to 2550

$N = 2048$ to 16384, R_λ from 650 to 2550 approx. Then refine grid to 32768^3



Favorable results for study of inertial range, especially R_λ 2550 at 32768^3 resolution (last line in gold), with intermittency correction and bottleneck readily identifiable.

MRIS: Multiple Resolution Independent Simulations

Simulations considered leading at any one time are usually short

- Numerical stability requires $\Delta t \propto \Delta x$. Overall Cost $\propto N^4$, at least.
- Simulating for a fixed physical time period with 2X in N would require 16 X increase in computing power, typically more than one machine generation.

But if our focus is on small scales (as in studies of intermittency)

- Start with long simulation at modest size, at the desired Reynolds number
- Take multiple well-separated snapshots, refine the grid, and run for a couple of Kolmogorov time scales: small scales adjust very fast.
- "MRIS" technique (Y & Ravikumar Phys. Rev. Fluids 2020) suffices to give high-accuracy results at extreme resolution, yet manageable cost

Particle tracking in 3D turbulence

Observer following the local fluid flow (vs a fixed observer)

- $d\mathbf{x}^+(t)/dt = \mathbf{u}^+(t)$ given $\mathbf{x}^+(0)$, initially homogeneously distributed
- $\mathbf{u}^+(t) = \mathbf{u}[(\mathbf{x}^+(t)), t]$: interpolate for velocity from neighboring grid points
- Cubic splines: fourth-order accurate, twice differentiable. Important for acceleration $\mathbf{a}^+(t) = d\mathbf{u}^+(t)/dt$ and small-scale quantities sampled along particle trajectories.

Algorithmic requirements and objectives:

- Track as many particles as possible ($\sim$ 1 billion, say)
- Additional memory due to the cubic spline coefficients
  — pencils code is a must for 32768^3, using 8192 nodes (65536 MPI tasks)

Particle-tracking performance on Frontier, with pencils code

- Dynamic mapping between particles and MPI processes (Buaria & Y, 2017)
- Ghost layer approach for access to spline coefficients in neighboring sub-domains
- Costs for spline coefficients and ghost layers are independent of particle count
- Eulerian part hitting MPI bandwidth limitations at 32768^3

N^3	$N_p(M)$	Nodes	P_r	Time(s)	Spl(s)	Ghost(s)	Lag(s)	L/E	WS(%)
2048^3	2	2	2	9.00	4.81	0.05	4.90	1.20	-
4096^3	16	16	2	10.96	5.16	0.10	5.29	0.93	84.2
8192^3	64	128	2	11.87	5.50	0.29	5.81	0.96	90.6
16384^3	128	1024	4	12.43	5.23	0.60	5.84	0.89	95.5
32768^3	256	8192	16	*18.49	4.62	0.86	5.62	0.43	67.2

Cost depends little on particle count. 32768^3 timing will improve w/ use of smaller P_r.

Some Concluding Remarks

The computer science point of view

- GPUs are the dominant technology in current leadership-class computing. (Last 3 world's #1 machines, all at OLCF, are all GPUs)
- Directives-based OpenMP programming have increased ease of use
- Even communication-intensive computations, such as those for turbulence and turbulent mixing based on multi-dimensional PDEs, can benefit.

The computational science point of view

- Direct numerical simulations have reached at 32768^3 on the world's first exascale machine. (10^9 times more grid points than 50 years ago)
- Looking to more collaborations, at scale, with theorists, modelers, experimentalists.