OpenMP

SC'20 Booth Talk Series

# Portable extreme scale turbulence simulations using OpenMP

P. K. Yeung and Kiran Ravikumar
Georgia Institute of Technology
with partial support from Oak Ridge National Laboratory

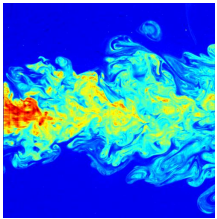
# Outline of the talk

1. Domain Science: Turbulence, and why do we need “extreme scale”

2. Computing: How we can use GPUs and OpenMP 5+ to advance our goals

# Turbulence: grand challenge in fluid dynamics

Great physical complexity: disorderly fluctuations, stochastic, despite being governed by deterministic physical laws. Said to be described by Nobel Laureate Richard Feynman as "the most important unsolved problem of classical physics".



Key to advances in engineering devices, prediction of extreme weather, airborne transmission of disease agents, etc. Fundamental understanding crucial for success in modeling. In short: very complex, very common, and very important.

# Turbulence: grand challenge in computing

Conservation of mass and momentum. With constant density ($\rho$) and viscosity ($\nu$), velocity and pressure fluctuations governed by (in the simplest case, with $\nabla \cdot \mathbf{u} = 0$)

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

The need for leadership-class computational resources:

- Unsteady, three-dimensional, stochastic, nonlinear, *wide range of scales*
- Ratio ($N$) of domain size to grid spacing to be large (for high Reynolds number)
- Resolution of small scales often critical (make $\Delta x$ smaller)
- Number of time steps required scales with $N$
- Overall operation count increases *as least as* fast as $N^4$ (16X per doubling of $N$)
- Large node count, long run time, huge datasets

# Psuedo-spectral methods and the 3D Fourier Transform

Spectral expansions: high accuracy, best in simplified geometries

- Write $\mathbf{u}(\mathbf{x}) = \sum_{\mathbf{k}} \hat{\mathbf{u}}(\mathbf{k}) \exp(i\mathbf{k}\cdot\mathbf{x})$. Then $\hat{\mathbf{u}} \perp \mathbf{k}$ in wavenumber space.

$$\partial\hat{\mathbf{u}}/\partial t + \nu k^2 \hat{\mathbf{u}} = -\widehat{[\mathbf{u}\cdot\nabla\mathbf{u}]}_{\perp\mathbf{k}} + \widehat{.....}$$

- Nonlinear terms: take F.T. of product, at cost $\sim N \ln_2 N$ for each line of data

Turbulence code structure designed with 3D FFTs in mind:

- Take transforms 1D at a time; transpose(s) via MPI_ALLTOALL required
- Domain decomposition: 2D (pencils) favored for massive CPU parallelism ($8192^3$ on Blue Waters); but 1D (slabs) better on fat CPU nodes with GPU accelerators
- Communication-intensive: how do we make best use of GPUs, which compute fast?

# Science questions at $>$ 6 trillion resolution on OLCF Summit

## Intermittency of flow variables prone to extreme fluctuations

- Dissipation rate: $\epsilon = 2\nu s_{ij} s_{ij}$ [intensity of local strain rate on fluid element] has samples $10^3$ times of mean value, or even higher
- Moments of $\epsilon$ averaged over 3D sub-domains of *intermediate* scale sizes needed for "intermittency corrections". Want to clarify scaling and evaluate the exponents:

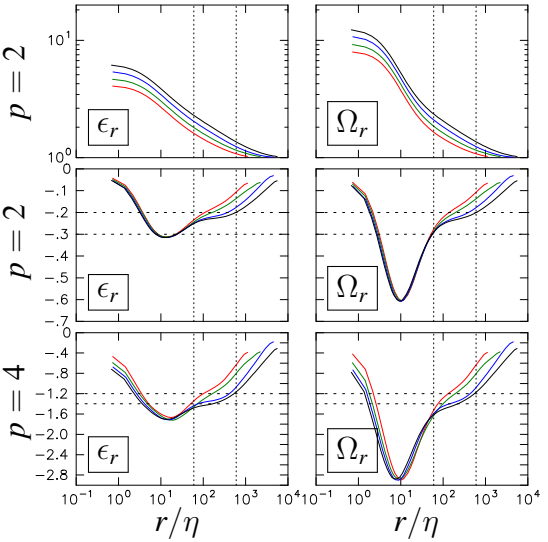
$$\langle \epsilon_r^p \rangle / \langle \epsilon \rangle^p \propto (r/\eta)^{-\zeta_p} \quad (p = 2, 4, 6, ..)$$

## Expensive: both Reynolds number and grid resolution must be high

- Record resolution using CUDA Fortran on Summit: $18{,}432^3$ using 3072 nodes (Ravikumar, Appelhans & Yeung: paper at SC'19)
- Science results: Yeung & Ravikumar: Phys. Rev. Fluids 2020 (invited)

# Science results at leadership resolution on OLCF Summit

- Forced isotropic turbulence at Taylor-scale Reynolds numbers ($R_\lambda$) 390, 650, 1000, 1300 on $3072^3$, $6144^3$, $12288^3$, $18432^3$ grids
- Including data on enstrophy (which is more intermittent)
- Trend towards a scaling range (plateau) as $R_\lambda$ increases — but less good at 4th order
- Conditional statistics (in paper) show that $\epsilon_r$ and $\Omega_r$ scale similarly and *together* — in the "inertial range"



# Future goals, needs, and cross-platform portability

Continuing to push the envelope, such as:

- $32{,}768^3$ for yet higher Reynolds number or small-scale resolution
- Lagrangian particle tracking for study of turbulent dispersion

What do we need to get there?

- Exascale computer with fast communication performance, at scale
- Portable accelerator programming: such as OpenMP 5+ vs. CUDA Fortran

# Outline of the talk

1. Domain Science: Turbulence, and why do we need “extreme scale”

2. Computing: How we can use GPUs and OpenMP 5+ to advance our goals

# 3D FFTs on $N^3$ grid using GPUs and batched approach

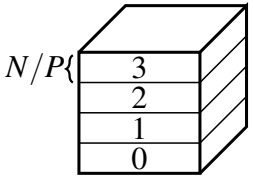
## Synchronous algorithm using $P$ processes

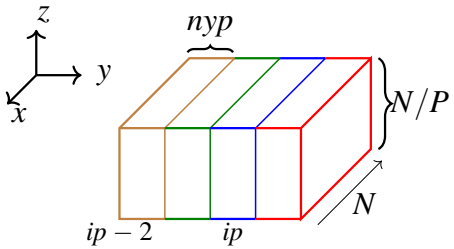
- Copy entire slab from CPU to GPU and back
- 1D FFTs using cufft or rocfft library
- MPI Alltoall among all processes to transpose $x-y$ to $x-z$ slabs

## Batched asynchronous algorithm (Ravikumar *et al.* SC'19)

- Divide slab into $np$ pencils and process each pencil separately ($nyp = nxp = N/np$)
- Overlap: Compute on $ip$, HtoD on $ip+1$, DtoH on $ip-1$ and all-to-all on $ip-2$
- Non-blocking all-to-all allows overlap

1D domain decomposition (Slabs)



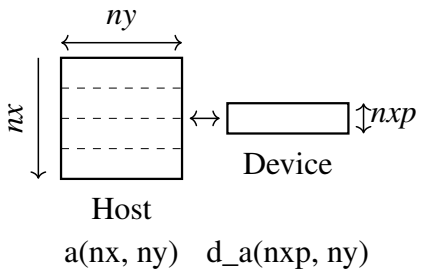



# Non-contiguous maps and strided copies

- FFTs in *y*: need only *a(1:nxp, 1:ny)* on device
- In CUDA Fortran use:

```
1  cudaMemCpy2DAsync (dst, dpitch, src, spitch, &
2      width, height, kind, stream)
```

– unique destination (*dst*) & source (*src*) buffers
– *width* (*nxp*) elements out of *spitch* (*nx*) to copy from first dimension, *height* (*ny*) such copies




## How to do it in OpenMP?

- `MAP(to:a(1:nxp, 1:ny))`: not 5.0 compliant
  — copy *a* to smaller *abuf* on host then MAP
  — but this adds extra work on the host

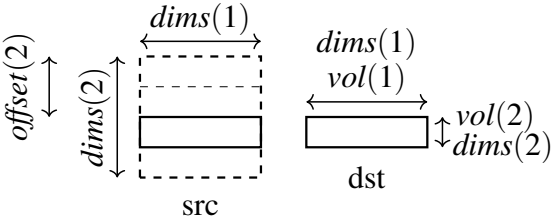
```
1  allocate (a(nx, ny), abuf(nxp, ny))
2  ! copy on host : abuf(:, :) = a(1:nxp, 1:ny)
3  !$OMP TARGET DATA MAP(tofrom: abuf)
4  ! do some work
5  !$OMP END TARGET DATA MAP
6  ! copy on host : a(1:nxp, 1:ny) = abuf(:, :)
```

Performance penalty due to additional operations on host

# OpenMP 4.5+: `omp_target_memcpy_rect`?

Copy rectangular subvolume from a multi-dimensional array

- Callable from C/C++, use C-Fortran interface
- Need to account for C vs. Fortran ordering – first dimension along row even though in Fortran it is along column




```
1  ! src on host of shape (nx, ny)
2  ! dst on device of shape (nxp, ny)
3
4  ! src(2nxp:3nxp, 1:ny) to dst(1:nxp, 1:ny)
5
6  num_dims = 2
7  vol(1) = ny ; vol(2) = nxp
8   dst_offset (1) = 0 ;  dst_offset (2) = 0
9   src_offset (1) = 0 ;  src_offset (2) = 2nxp
10 dst_dims(1) = ny ; dst_dims(2) = nxp
11 src_dims(1) = ny ; src_dims(2) = nx
12
13 omp_target_memcpy_rect (dst, src, elem_size,
    ndims, vol, dst_offset , src_offset ,
    dst_dims, src_dims, dst_dev, src_dev)
```

- OpenMP 5.1: asynchronous copies, `omp_target_memcpy_rect_async` — Use *dependency objects* to synchronize tasks

# Interoperability between OpenMP and non-blocking libraries

```
1  TARGET DATA MAP(tofrom:a)
2
3  TASK DEPEND(OUT:var)                          Ⓐ
4  TARGET DATA USE_DEVICE_PTR(a)
5  FFTExecute (a, forward, stream)
6  FFTExecute (a, inverse , stream)
7  END TARGET DATA
8  END TASK
9
10 TARGET TEAMS DISTRIBUTE DEPEND(IN:var) NOWAIT
11 a(:, :, :) = a(:, :, :)/nx                    Ⓑ
12 END TARGET TEAMS DISTRIBUTE
13
14 TASKWAIT
15 END TARGET DATA
```

- Task A: Non-blocking FFT call
- Task B: OpenMP kernel with dependency on previous task
- Expect kernel runs after FFT completes
- But task A is considered complete after call to library, freeing dependency
- B executes before FFT completes
- Returns incorrect results

Task dependency not sufficient to enforce required synchronization

# DETACH to enforce synchronization

```
1  TARGET DATA MAP(tofrom: a)
2
3  TASK DEPEND(out:var) DETACH(event)
4
5  TARGET DATA USE_DEVICE_PTR(a)   Ⓐ
6  FFTExecute (a, forward, stream)
7  FFTExecute (a, inverse , stream)
8  END TARGET DATA
9
10 hipStreamAddCallback (stream, ptr_callback, C_LOC(event), 0)
11 END TASK
12
13 ! Copy or compute on other data   Ⓒ
14
15 TARGET TEAMS DISTRIBUTE DEPEND(IN:var) NOWAIT
16 a(:, :, :) = a(:, :, :)/nx                    Ⓑ
17 END TARGET TEAMS DISTRIBUTE
18
19 END TARGET DATA
```

```
1 subroutine callback (stream, status, event)
2   type(c_ptr) :: event
3   integer(kind=omp_event_handle_kind) :: f_event
4   call C_F_POINTER (event, f_event)
5   call omp_fulfill_event(f_event)
6 end subroutine callback
```

1. A: launch FFT, add call to *callback* in stream where FFT is running
2. B waits as dependent on A, C executes asynchronously
3. After FFT, function *callback* is called and event fulfilled
4. A completes allowing B to run

# Porting asynchronous CUDA Fortran to OpenMP

```
1  do ip=1,np
2      NEXT = mod(ip+1,3); CURR = mod(ip,3);
3      PREV = mod(ip−1,3); COMM = mod(ip−2,3);
4      cudaStreamWaitEvent (trans_stream, DtoH(NEXT), 0)
5      cudaMemCpy2DAsync (abuf(NEXT),a(ip+1),trans_stream)
6      cudaEventRecord (HtoD(NEXT),trans_stream)
7      cudaStreamWaitEvent (comp_stream, HtoD(CURR), 0)
8      FFTExecute  (abuf(CURR), comp_stream)
9      cudaEventRecord (comp(CURR), comp_stream)
10     cudaStreamWaitEvent (trans_stream, comp(PREV), 0)
11     cudaMemCpy2DAsync (snd(ip-1), abuf(PREV), &
12         trans_stream)
13     cudaEventRecord (DtoH(PREV), trans_stream)
14     cudaEventSynchronize (DtoH(COMM))
15     MPI_IALLTOALL (snd(ip-2))
16 end do
```

```
1  do ip=1,np
2      NEXT = mod(ip+1,3); CURR = mod(ip,3);
3      PREV = mod(ip−1,3); COMM = mod(ip−2,3);
4      TASK DEPEND (IN:DtoH(NEXT), OUT:HtoD(NEXT))
5      omp_target_memcpy_rect (abuf(NEXT), a(ip+1))
6
7      TASK DEPEND (IN:HtoD(CURR), OUT:comp(CURR))
8          DETACH(event)
9      FFTExecute  (abuf(CURR), comp_stream)
10     TASK DEPEND (IN:comp(PREV), OUT:DtoH(PREV))
11     omp_target_memcpy_rect (snd(ip−1),  abuf(PREV))
12
13
14     TASK DEPEND(IN:DtoH(COMM))
15     MPI_IALLTOALL (snd(ip-2))
16 end do
```

- `DEPEND` clause replaces `cudaEventRecord` & `cudaStreamWaitEvent`
- `omp_target_memcpy_rect` replaces `cudaMemCpy2DAsync`

# Performance of 3D FFT kernel on Summit

Summit (XL compiler) up to 1024 nodes ($\sim 22\%$ of full machine) using 1 task/GPU
Timings for 3 pairs of forward & inverse transforms using non-batched synchronous code

| # Nodes | Prob. Size | Time (s) | | |
|---|---|---|---|---|
| | | CPU | CUDA | OMP |
| 2 | $1536^3$ | 5.21 | 2.39 | 2.41 |
| 16 | $3072^3$ | 6.79 | 3.30 | 3.16 |
| 128 | $6144^3$ | 9.10 | 5.26 | 5.01 |
| 1024 | $12288^3$ | 10.59 | 4.30 | 4.12 |

- OpenMP & CUDA show similar performance ($\sim$ 2.6X speedup for $12k^3$)
- Turbulence simulation using CUDA: $18k^3$ on $3k$ nodes, $\sim$ 3X speedup

## Batched asynchronous code using OpenMP

- `omp_target_memcpy_rect` slow compared to `cudaMemCpy2D`
- Asynchronous 3D FFT kernel using DETACH under development

Portability: OpenMP code compiles and runs on Nvidia and AMD GPUs

# Summary and Future Work

- Direct numerical simulation at record-setting resolution
  - High-fidelity results on intermittency and extreme events at high Reynolds number
  - Computationally-demanding studies of turbulent mixing and particle dispersion
- 3D FFT kernel using OpenMP 5.0+ to target GPUs
  - At present: batched, synchronous, which enable large problem sizes
  - Porting successful batched asynchronous CUDA Fortran code (SC'19): in progress
- Future work towards 3D FFTs at massive scale, at resolution beyond $18{,}432^3$
  - Batched asynchronism algorithm (using `DETACH`) needed for optimal performance
  - Fast strided copy b/w small device & large host array: `omp_target_memcpy_rect`


**openmp.org** OpenMP API specs, forum, reference guides and more

**link.openmp.org/sc20** Videos and PDFs of OpenMP SC'20 presentations