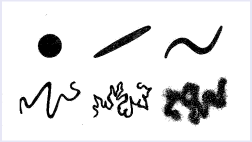
Turbulence, Turbulent Mixing and GPU-Accelerated Computing on TITAN

M. P. Clay[1], D. Buaria[2], P. K. Yeung[1]
E-mail: mclay6@gatech.edu

[1]Georgia Institute of Technology
[2]Max Planck Institute for Dynamics and Self-Organization



U.S. DEPARTMENT OF ENERGY INCITE LEADERSHIP COMPUTING

OLCF User Meeting, Oak Ridge, May 15, 2018

Outline

1. Computational Algorithm for Mixing at High Schmidt Number
2. Brief Review of OpenMP 4.X Target Offloading
3. OpenMP 4.5 GPU Algorithm on Titan
4. Conclusions and References

Outline

1 Computational Algorithm for Mixing at High Schmidt Number

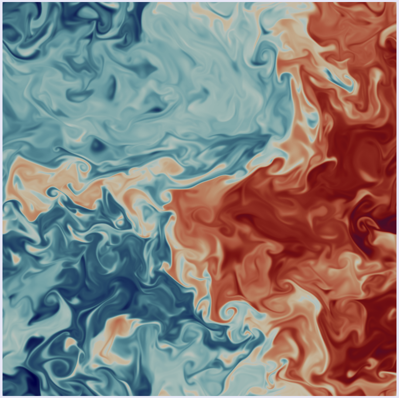
2 Brief Review of OpenMP 4.X Target Offloading

3 OpenMP 4.5 GPU Algorithm on Titan

4 Conclusions and References

Challenges Facing Simulations of Turbulent Mixing

When the scalar is weakly-diffusive (e.g., salinity in the ocean), resolution requirements for scalar are stricter than the velocity field.



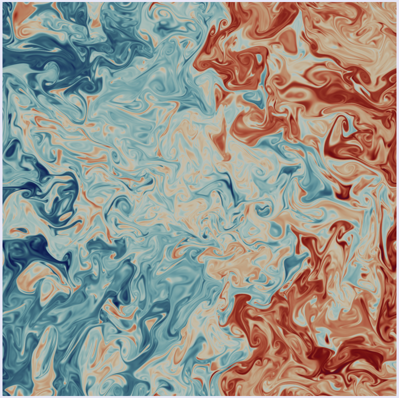


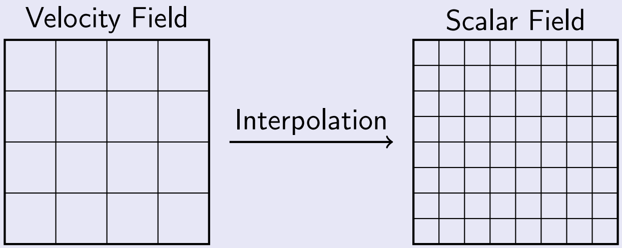
Figure: Scalar fluctuations for (left) a low-diffusivity scalar and (right) a scalar with even lower diffusivity in same (statistically) turbulence.

A Dual-Grid Dual-Scheme Approach

- Velocity field: coarse grid, N-S equations, Fourier pseudo-spectral scheme.
- Scalar fluctuations (main interest) on finer grid (Gotoh *et al.* 2012)

$$\partial\theta/\partial t + \boldsymbol{u}\cdot\nabla\theta = D\nabla^2\theta - \boldsymbol{u}\cdot\nabla\langle\Theta\rangle$$

  - Derivatives via eighth-order combined compact finite differences (CCD).
  - Interpolate velocity field from coarse grid to fine grid for advection terms.

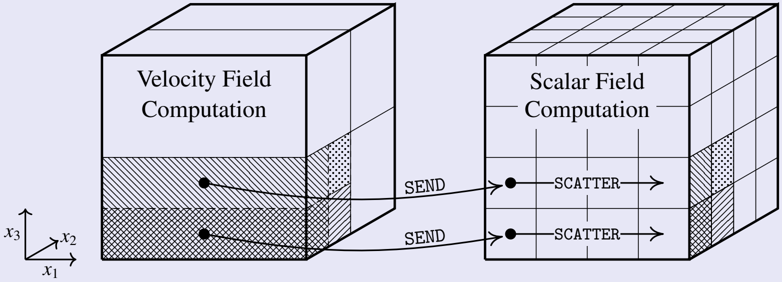


- For our simulations, scalar grid is finer than the velocity grid by a factor of 8 in each direction $\Longrightarrow$ computational cost dominated by scalar.

Parallel Implementation for Weakly-Diffusive Scalars

Disjoint groups of processors for the two fields (Clay *et al.* 2017)

- To form advective terms, send well-resolved velocity field to scalar communicator, and perform tricubic interpolation.
- Overlap inter-communicator transfer with computations for scalar.



Our focus here is on how OpenMP is used for the scalar field computations appearing on the right (the larger computation).

CCD Scheme and Opportunities to Improve Scalability

Application of the CCD scheme is the most expensive part of the code. Scheme is implicit: all points along a grid line are coupled.

- Parallel algorithm (Nihei *et al.* 2003) to solve system w/o transposes.
- Basic steps required for distributed memory CPU implementation:

Op.	Operation Summary
A	Fill ghost layers for scalar field with `SEND` and `RECV` operations
B	Form right-hand-side of linear system and obtain solution
C	Pack and distribute data for reduced system with `MPI_ALLTOALL`
D	Unpack data and solve reduced linear system
E	Pack and distribute data for final solution with `MPI_ALLTOALL`
F	Unpack data and finalize solution of CCD linear system

- Operations for three coordinate directions are independent.
  - Try to overlap communication with computation.

Outline

1 Computational Algorithm for Mixing at High Schmidt Number

2 Brief Review of OpenMP 4.X Target Offloading

3 OpenMP 4.5 GPU Algorithm on Titan

4 Conclusions and References

Basics of GPU Acceleration with OpenMP 4.X

Like with OpenACC, when using OpenMP for GPU acceleration users rely on the compiler to do most of the "heavy lifting". Users will need:

1. Constructs to control data movement between the CPU and GPU(s).
2. Constructs to execute (hopefully accelerate) kernels on the GPU(s).

To interface with the GPU (device), use `TARGET` constructs:

- `TARGET DATA MAP(...)`: map data to the device data environment.
- `TARGET UPDATE TO/FROM(...)`: push/pull data to/from the device.
- `TARGET TEAMS`, `DISTRIBUTE`, and `PARALLEL DO`: split up work over GPU threads, with a team corresponding to a CUDA thread block.

Will see some examples in the coming slides. Also see "OpenMP Application Programming Interface Examples" for many examples.

Asynchronous Execution with OpenMP 4.5

Can often improve performance if host/device operate asynchronously.

- For example, both the host and device can perform computations, or the host can perform communication while the device computes.
- Try to keep all resources active as much as possible!

OpenMP has supported async. target execution since version 4.5.

- Cray supports the necessary clauses, beginning with CCE/8.5.

To use this capability, code must indicate which kernels can run asynchronously, and must express necessary synchronization explicitly. Relevant OpenMP clauses to append to `TARGET` constructs include:

- `NOWAIT`: a kernel may be run asynchronously.
- `DEPEND`: used to enforce an ordering of operations involving the device.

Outline

1. Computational Algorithm for Mixing at High Schmidt Number
2. Brief Review of OpenMP 4.X Target Offloading
3. OpenMP 4.5 GPU Algorithm on Titan
4. Conclusions and References

Accelerating a Production DNS Code

At the right place at the right time:

- Started porting code to run on GPUs with OpenACC in summer 2016 to apply for 2017 INCITE allocation.
- Transitioned to Cray's OpenMP 4.X implementation (Clay *et al.* 2018) in early 2017, which was mature enough for production-level work.

Original plans for the acceleration effort:

- Overall cost dominated by scalar field computation: accelerate this portion, leave small velocity computation untouched.
- Minimize data movement: put entire scalar computation on the GPU.
  - Challenge: on XK7 nodes, 32 GB on the host, 6 GB on the device.
- Can scalability be improved by overlapping communication and computation as much as possible?

Summary of Acceleration of DNS Code Using CCE/8.6

Algorithmic changes required to run on Titan:

- Drastic reduction in memory to reduce minimum required node count.
  - Now require 8192 nodes instead of 16834 nodes for 8192^3 problem.
- CCD linear system in x_1 requires different use of available memory.
  - Perhaps the most performance-sensitive GPU kernel in the code.
- For accelerated code, cannot calculate all derivatives simultaneously.
  - Memory restrictions: calculate x_2 and x_3 together, calculate x_1 separately.
- Manually packing/unpacking buffers for host/device data transfers.

Challenges to achieve good scalability with the new algorithm:

- Computations accelerated, but communication remains the same.
- Use OpenMP 4.5's `NOWAIT` and `DEPEND` to overlap communication and computation, wherever possible.

Time Stepping Algorithm on Titan

Algorithmic changes to RK4 required to run on Titan.

- For best performance, do not calc. all derivatives together (memory).

Step	Device	Operation Summary
1	CPU	Receive velocity field and fill ghost layers
2	PCI	Transfer u_1 velocity to GPU
3	ALL	Calculate scalar derivatives in x_1; interpolate u_1
4	PCI	Begin transfer of u_3 velocity to GPU
5	GPU	Increment RK4 with x_1 diffusion and partial advection
6	ALL	Calculate advection derivative in x_1
7	GPU	Increment RK4 with x_1 advection term
8	ALL	Calculate scalar derivatives in x_2 and x_3; interpolate u_3
9	PCI	Begin transfer of u_2 velocity to GPU
10	GPU	Increment RK4 with x_2 and x_3 diffusion and x_3 advection
11	ALL	Begin calculation of x_3 advection derivative; interpolate u_2
12	GPU	Increment RK4 with x_2 partial advection
13	ALL	Finalize advection derivatives in x_2 and x_3
14	GPU	Perform RK4 sub-stage update

A Conflict: Memory Layout vs Computational Performance

Code uses 3D arrays which are all allocated in the same way.

- For example: `ALLOCATE(df1(nc1,nc2,nc3))`.
- For most loops, we get great (coalesced) access along the inner index.

A problematic kernel: solving a linear system in the x_1 direction.

```
DO k=1,nc3; DO j=1,nc2; DO i=2,nc1
df1(i,j,k)=F[df1(i,j,k),df1(i-1,j,k)]
END DO; END DO; END DO
```

- Cannot vectorize `i` loop, but need memory access along inner index.

Swap memory layout to improve this kernel:

- Make `j` loop the inner index: `ALLOCATE(buf(nc2,nc1,nc3))`

```
DO k=1,nc3; DO i=2,nc1; DO j=1,nc2
buf(j,i,k)=F[buf(j,i,k),buf(j,i-1,k)]
END DO; END DO; END DO
```

- Not free: loops elsewhere in code need original memory layout.

Performance of Routine Applying CCD in the x_1 Direction

Use kernel to measure performance for CPU and GPU execution.

- Focusing on computations in the x_1 direction.
- GPU performance metrics with nvprof: `dram_util.`, `alu_fu_util`.
- Test problem: 512^3 with 2x2x2 process layout and 4 OpenMP threads.

Computations with original memory layout

Loop	CPU (s)	GPU (s)	Speedup	`dram`	`alu`
RHS	0.0895	0.0125	7.13	7	9
Lin. Sys.	0.5576	0.2161	2.58	2	1
Final Sol.	0.2354	0.0211	11.2	7	8
Total	0.8824	0.2497	3.53	—	—

Computations with swapped memory layout and loop blocking

Loop	CPU (s)	GPU (s)	Speedup	`dram`	`alu`
RHS	0.1265	0.0185	6.84	5	9
Lin. Sys.	0.1407	0.0336	4.19	7	2
Final Sol.	0.1515	0.0153	9.88	8	9
Total	0.4187	0.0674	6.21	—	—

OpenMP 4.5 Usage with CCE/8.6 in DNS Code

Use tasking clauses on TARGET constructs to overlap comm./comput.

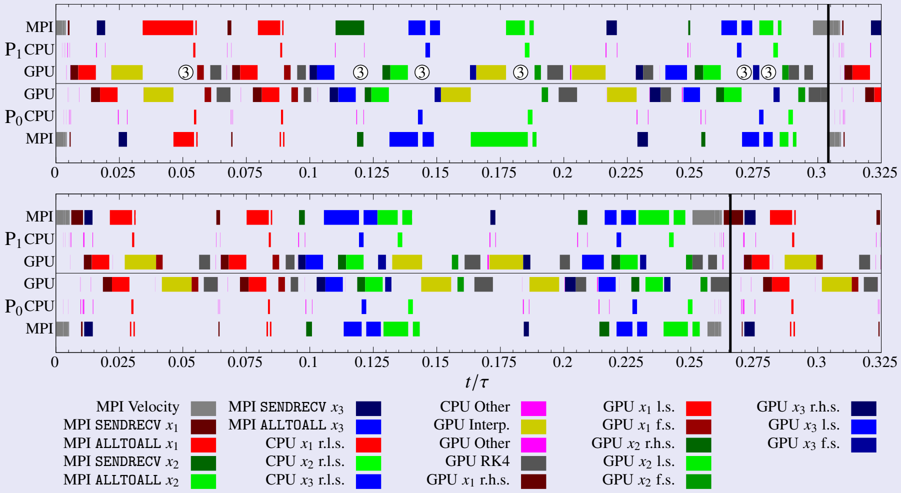
- Ensure correct ordering of kernels with DEPEND and a directionally-dependent dummy variable, e.g., SYNCX3 for the x_3 direction.
- Before performing communication in, say, x_3, launch all available x_2 kernels asynchronously with NOWAIT.

```
! From previous data movement, make sure data is on host.
!$OMP TARGET DEPEND(IN:SYNCX3)
!$OMP END TARGET
!
! Launch all kernels in the X2 direction (showing just one).
!$OMP TARGET TEAMS DISTRIBUTE DEPEND(INOUT:SYNCX2) NOWAIT
<Computational task on the GPU for the X2 direction>
!$OMP END TARGET TEAMS DISTRIBUTE
!
! Proceed with communication call in the X3 direction.
CALL MPI_ALLTOALL(...)
```

Timelines with and without `NOWAIT`

Examine how node utilization changes with asynchronous execution

- 4096^3 grid using 1024 nodes with 2 MPI processes per node



Performance and Scalability of Accelerated DNS Code

Appx. 5X speedup, with improvement from 80% (non-async.) to 90% (async. with `NOWAIT`) weak-scaling for 8192^3 on 8192 nodes.

CPU-only:	14.8 for 512^3 and 16.13 for 8192^3
OpenMP 4.5 GPU:	2.93 for 512^3 and 3.26 for 8192^3

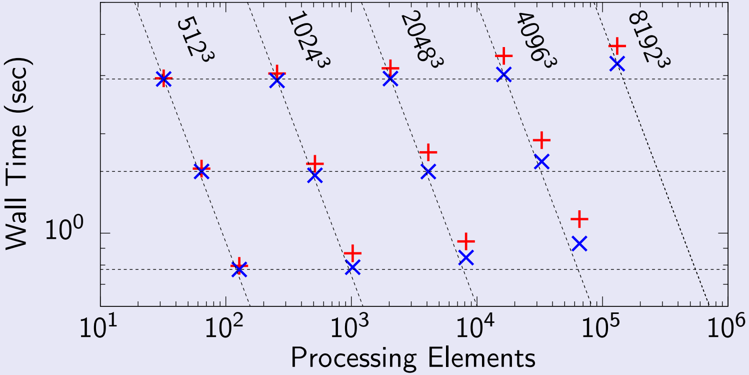


Figure: GPU code timings for non-async. (+) and async. using `NOWAIT` (X).

Impact of Host Configuration on Overall Performance

Question: what host (CPU) MPI/OpenMP configuration to use?

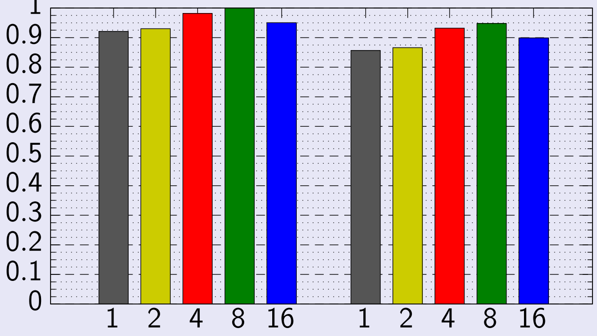


Figure: Normalized time per step for (left) 1024^3 and (right) 4096^3 varying num. of OpenMP threads on CPU (1 to 16). Normalized by 8-thread 1024^3 timing.

Neither extreme (pure MPI or OpenMP), but larger sub-domains give better kernel performance, and two MPI processes keep GPU busy.

Outline

1 Computational Algorithm for Mixing at High Schmidt Number

2 Brief Review of OpenMP 4.X Target Offloading

3 OpenMP 4.5 GPU Algorithm on Titan

4 Conclusions and References

Conclusions and Outlook

Using OpenMP 4.5 (via CCE/8.6) to accelerate a turbulence code:

- Strategy was to place entire scalar field computation on the GPUs.
- Following algorithmic changes for a key kernel (i.e., swapping memory layout for x_1 derivatives) we achieve 5X speedup.
- Use of OpenMP 4.5 tasking clauses on `TARGET` constructs (i.e., `DEPEND` and `NOWAIT`) to make code asynchronous improves scalability.

Future work and extensions

- For Summit, kernels modified for IBM XLF (2017 OLCF hackathon). Velocity field must be accelerated (see poster by K. Ravikumar).
- For differential diffusion of two scalars, include a moderate Schmidt number scalar in the pseudo-spectral computation.
- For active scalars, communicators become strongly coupled. Must assess performance to determine final INCITE 2018 configurations.

References

1. T. Nihei & K. Ishii (2003) Parallelization of a highly accurate finite difference scheme for fluid flow calculations, *Theor. Appl. Mech. Japan*, **52**, 71–81.
2. T. Gotoh, S. Hatanaka & H. Miura (2012) Spectral compact difference hybrid computation of passive scalar in isotropic turbulence, *J. Comput. Phys.*, **231**, 7298–7414.
3. OpenMP Architecture Review Board (2015) OpenMP Application Programming Interface Examples, `www.openmp.org`.
4. M. P. Clay, D. Buaria, T. Gotoh & P. K. Yeung (2017) A dual communicator and dual grid-resolution algorithm for petascale simulations of turbulent mixing at high Schmidt number, *Comput. Phys. Commun.*, **219**, 313–328.
5. M. P. Clay, D. Buaria, P. K. Yeung & T. Gotoh, GPU acceleration of a petascale application for turbulent mixing at high Schmidt number using OpenMP 4.5, *Comput. Phys. Commun.*, **228**, 100–114.