OpenMP offloading on the Exascale Frontier: An example application in pseudo-spectral algorithms

Stephen Nichols (ORNL) and P.K. Yeung (Georgia Tech)

E-mails: nicholss@ornl.gov; pk.yeung@ae.gatech.edu

OpenMP Users Telecon
October 27, 2023
CAAR Program for Frontier, and DOE INCITE Award 2023
Oak Ridge Leadership Computing Facility, USA
NSF Subaward via The Johns Hopkins University

An Outline of This Talk

- Why use OpenMP offload, what are the alternatives
- Pseudo-spectral methods, including domain decomposition
- Code fragments showing how we use OpenMP
  (incl. TARGET DATA MAP, TEAMS, UPDATE, GPU-aware MPI)
- One remaining challenge: array reductions on the GPU
- Performance data (for turbulence code) on Frontier
- Science results (re: turbulence) achieved on Frontier, to date

GPU at the Exascale: Why OpenMP?

Code is written in Fortran

- Underlying code base developed over many years by the PI and collaborators (CPU version performed well on Blue Waters, Frontera, etc)
- Includes features allowing interoperability with library packages in C
- Emphasize in-house development: minimize dependence on external packages
- Like most scientific codes, large nested loops do all the work
- Distributed memory: communication among MPI processes is dominant

OpenMP Offloading

- OpenMP already used in code for thread-level parallelization on CPU
- Uses a familiar pattern of directives and clauses
- OpenMP standards provide a higher degree of portability

Application: Fourier pseudo-spectral methods and 3D FFT

- Fourier representation decomposes a complex signal into contributions from a wide range of scale sizes. Inverse and forward transforms linked by reciprocal relations.

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

- For nonlinear products: exact result is $\widehat{fg}(\mathbf{k}) = \sum_{\mathbf{k}'} \hat{f}(\mathbf{k}')\hat{g}(\mathbf{k}-\mathbf{k}')$, but cost of convolution sum is prohibitive ($\propto N^6$ operations in 3D)
- "Pseudo"-spectral: multiply in physical space, then transform. Cost now scales as $N^3 \log_2 N$. Some numerical (aliasing) errors arise, but those are controllable.
- We use this approach for fundamental studies in fluid turbulence, which is a complex multi-scaled unsteady, 3D phenomenon governed by nonlinear PDEs.
- FFTs have many other uses: imaging, materials science, and others.

Direct Numerical Simulations of Turbulence

DNS — computing all the details of the instantaneous, 3D fluid flow according to exact governing equations (conservation of mass and momentum)

Due to their problem size, these simulations require Leadership Class facilities:

- with memory to hold the problem
- with reliable, high-speed communications
- with technology to accelerate the solution process
- with storage to hold the solution

In 2019: 18432^3 simulation on 200-PF/s "Summit" at OLCF (Ravikumar, Appelhans & Yeung, SC'19), with batched-asynchronism, on 3072 nodes ($\approx 67\%$ of "Summit").

On Frontier, we're aiming higher yet: a 32768^3 simulation, resolving small scales better and higher Reynolds numbers importance for turbulence theory

Pseudo-spectral DNS on a 3-D periodic domain

Take Fourier transform, project onto plane $\perp$ to wavenumber vector **k**

$$\partial\hat{\mathbf{u}}/\partial t = -[\widehat{\nabla\cdot(\mathbf{uu})}]_{\perp \mathrm{k}} - \nu k^2\hat{\mathbf{u}} - \hat{\mathbf{f}}$$

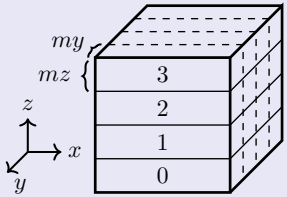
Time advance: 2nd order Runge Kutta for nonlinear terms, integrating factor for viscous terms. Starting with $\hat{\mathbf{u}}(\mathbf{k})$ in wavenumber space, key tasks for each RK substep are:

- **3D FFT transform to physical space**
- form nonlinear terms in physical space
- **3D FFT transform of nonlinear terms back to wavenumber space**
- spatial differentation via Fourier space (e.g. $\widehat{\partial u/\partial x} = ik_x\hat{u}$)
- advance in time ... and then repeat

The 3D FFT transforms include intense computation AND communication

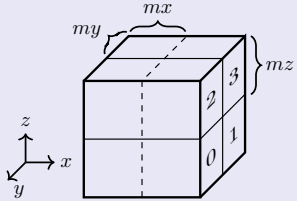
Domain Decomposition on an $\mathbf{N^3}$ grid: Slabs or Pencils?

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



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

Sub-divide in 2D: each parallel process handles 1 pencil (P up to N^2)



- FFT in x; transpose; FFT in z; transpose; FFT in y
- Action within row and column sub-communicators, P_r and P_c

Actual choice deepends on memory per node or core and the problem size

What do the 3D FFT transforms look like?

Inverse transform from wavenumber space to physical space:

- 1D C2C FFT in y
- pack send buffer, **MPI AllToAll**, unpack receive buffer
- 1D C2C FFT in z
- *(for "pencils" only) pack send buffer,* **MPI AllToAll**, *unpack receive buffer*
- 1D C2R FFT in x

Foward transform from physical space to wavenumber space:

- 1D R2C FFT in x
- *(for "pencils" only) pack send buffer,* **MPI AllToAll**, *unpack receive buffer*
- 1D C2C FFT in z
- pack send buffer, **MPI AllToAll**, unpack receive buffer
- 1D C2C FFT in y

We're doing a lot of FFT computations and MPI communications!

General Features of the Turbulence Code

Fortran 95 with iso-c-bindings

Custom 3D FFT packages for slab and pencil decompositions:

- FFTW on the CPUs (for reference)
- cuFFT/rocFFT libraries on Nvidia/AMD GPUs
- CUDA Fortran or HIPFORT provides the fortran interfaces for the libraries

GPU-Direct MPI communications to avoid extra copies between CPU and GPU

OpenMP Offloading for computation and data management/movement:

- TARGET DATA region for the entire time-evolution loop (with *target data map*)
- Working/Computing loops (*target teams distribute parallel do (simd)*)
- Data management/movement (*update to/from*, *use_device_ptr*)

Code Snippet: MPI AllToAll Communications

If all calculations are done on the GPU, MPI via CPU requires extra data copies between the host and device, but the form of the MPI call remains unchanged.

```
if (use_gpu_mpi .eq. 1) then
    !$OMP TARGET DATA USE_DEVICE_PTR(upxz,upxy)
    call MPI_ALLTOALL (upxz, data_size, MPI_COMPLEX, upxy, data_size, MPI_COMPLEX,
                       MPI_COMM_WORLD, mpierr)
    !$OMP END TARGET DATA
else   !! on the CPU
    !$OMP TARGET UPDATE FROM(upxz)
    call MPI_ALLTOALL (upxz, data_size, MPI_COMPLEX, upxy, data_size, MPI_COMPLEX,
                       MPI_COMM_WORLD, mpierr)
    !$OMP TARGET UPDATE TO(upxy)
end if
```

Code Snippet: Typical cuFFT/rocFFT Function Call

```
!$OMP TARGET DATA MAP(tofrom:vxz)
.....
!$OMP TARGET DATA USE_DEVICE_PTR(vxz)
#ifdef USE_NVIDIA
    call cufftExecR2C(cu_plan_r2c,C_LOC(vxz(1,1,1,ivar)),C_LOC(vxz(1,1,1,ivar)))
#elif USE_AMD
    ierr = rocfft_execute(roc_plan_r2c, C_LOC(vxz(1,1,1,ivar)),C_NULL_PTR,
                          C_NULL_PTR)
#endif
!$OMP END TARGET DATA
.....
!$OMP END TARGET DATA
```

Code Snippet: Typical OpenMP Offload Kernel

```
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD COLLAPSE(4) &
!$OMP         PRIVATE (ivar,y,z,x) &
!$OMP         SHARED (nvar,my,nz,nxhp,factor,vxz)

do ivar=1,nvar
   do y=1,my
      do z=1,nz
         do x=1,nxhp
            vxz(x,z,y,ivar)=vxz(x,z,y,ivar)*factor
         end do
      end do
   end do
end do

!$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD
```

Performant Array Reductions on GPUs Are Needed!

Array reductions work correctly on the MI250x GPUs but are not performant
— performance worsens as the problem size increases

OpenMP Atomics perform much better on the GPUs than the array reductions but are still slower than performing array reductions on the CPU
— as with array reductions, performance worsens as the problem size increases

For now, array reductions are the last part of the computations that we perform on the CPUs

Additional OpenMP + FFT Examples

Fortran+OpenMP Offloading:
— https://code.ornl.gov/gests-reproducers/fortran-rocfft-strided-c2c-example.git

C++ for strided transforms:
— separate branches for data management with OpenMP Offloading or HIP
— https://code.ornl.gov/gests-reproducers/rocfft-strided-c2c-example.git

C++ for stride = 1 transforms:
— https://code.ornl.gov/gests-reproducers/rocfft_example_code.git

More demo/example codes are available. Send me an email if you're interested.

Suggestions for using Fortran + OpenMP Offloading

Be flexible in your coding

- Each compiler has its own implementation that will mature at its own rate
- Don't over-load and/or over-optimize offloaded kernels/routines
- Likely need to re-factor your code for best performance

Portability (functionality and correctness) before performance

- A few iterations may be necessary for portability
- "General" optimization for reasonable cross-platform performance

Wall time/step (secs) and weak scaling: Slabs code

Frontier: CCE 15.0.0, ROCM 5.4.0, using $\sim$ 42 GB of 64 GB memory per GPU

N^3	#Nodes	#MPI	FFT	Pack+Unpack	MPI	Other	Total	WS(%)
2048^3	1	8	1.360	0.243	1.846	0.643	4.09	-
4096^3	8	64	1.409	0.246	7.762	0.685	10.11	40.5
8192^3	64	512	1.580	0.255	8.152	0.764	10.75	94.0
16384^3	512	4096	2.196	0.268	8.360	0.700	11.53	93.2
32768^3	4096	32768	3.423	0.276	8.621	0.747	13.07	88.2

- Low scalability from 1 node to 8 nodes due to communication across nodes being slower than within the node. Yet, not much deterioration beyond 8 nodes.
- From 4096^3 onwards, approx 90% weak scaling for every doubling of N
- Slabs faster than pencils (by 1 sec at 32768^3) because of less pack+unpack.
- But pencils code still needed for particle tracking, where we need 8192 nodes to provide memory for 32768^3 grid. Some further improvements still possible.

Strong scaling and message sizes: Pencils Code

- Get results faster by using more nodes than needed for memory requirements. Hopefully, only minor increase in overall cost if strong scaling is high.
- More nodes imply larger column communicator. Inter-mode MPI is slower. Messages smaller than a certain size lead to latency issues and variability.
- Can increase message size by doing alltoalls for several variables at a time. Memory needed by larger communication buffers may still fit (using 2X, 4X, nodes, etc).
- 16384^3, 2048 nodes (vs. 13.0 secs on 512 nodes), over 8 time steps:
  between 3.33 and 8.79 secs when using altoall 1 variable at a time
  between 3.41 and 3.43 secs when using altoall up to 4 variables at a time
- However benefit is not as great for 32768^3 at 8192 vs 4096 nodes.
  — We may have reached the limits of the machine.

From Advances in Computing to Advances in Domain Science

Checkpointing, with flexibility

- In Fourier space, where continuity equation allows some savings
- Allow for change of domain decomposition and/or resolution

How many time steps do we need? By what criteria?

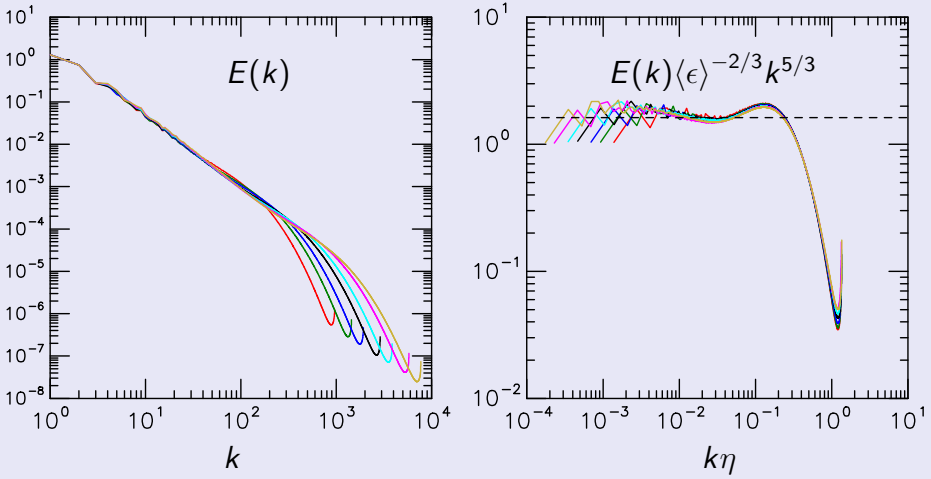
- Studying the small scales: use the Kolmogorov time scale
- Smaller Δt (more time steps) if need to resolve small scales better (smaller Δx), increase Reynolds number, or improve resolution in time.

Extracting statistics from the raw numerical solution

- On-the-fly processing can take significant resources, depending on how frequent
- Post-processing is only option for some quantities, but results not time-resolved

Early Science results: Energy spectrum at R_λ up to 2550

Data from stationary states reached with N from 2048 to 16384, R_λ 650 to 2550 approx:



Already seeing very conclusive results in inertial range (intermediate k) versus classical Kolmogorov theory. Next: to resolve small scales better ... using 32768^3 simulations.

Summary and Next Steps

Turbulence simulations at the Exascale

- Successfully developed a GPU turbulence code using Fourier pseudo-spectral methods up to 32768^3 resolution on the world's first Exascale supercomputer.
- OpenMP offload between coherent CPU and GPU memories has played a major role, with only minimal costs in host-device copies
- Code has been used to obtain statistically isotropic turbulence at Taylor-scale Reynolds number 2550, by reducing the viscosity while refining the grid spacing.

Extensions and future plans include:

- Study of turbulent dispersion: tracking particles in the flow.
- Study of turbulent mixing: a focus on case of very low diffusivity (e.g. salinity in ocean) using a dual-resolution approach, with asynchronism.
- Make selected datasets available via the Johns Hopkins Turbulence DataBase.