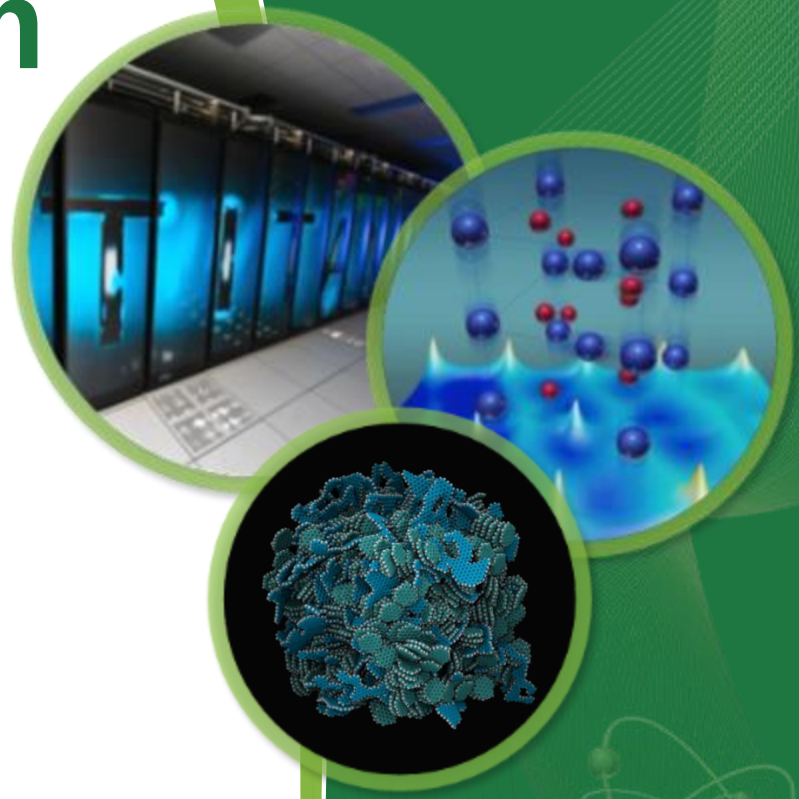
Refactoring the Energy Exascale Earth System Model (E3SM) Super Parameterization for GPUs



Matt Norman	Oak Ridge National Laboratory
Sarat Sreepathi	Oak Ridge National Laboratory
Anikesh Pal	Oak Ridge National Laboratory
Jeff Larkin	Nvidia
Lixiang (Eric) Luo	IBM
Mark Taylor	Sandia National Laboratory

ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Energy Exascale Earth System Model (E3SM) – Multiscale Modeling Framework

- E3SM
  - Branched as “ACME” from CESM a few years ago
  - Changed names from ACME to E3SM recently
  - Coupling of five main models: Atm, Ocn, Land, Land Ice, Sea Ice
  - Targeted to hi-res DOE climate Grand Challenge questions
- E3SM Multi-Scale Modeling Framework (MMF)
  - Cloud resolving scales (dx=1km) will require about 22,000x more computation
  - MMF is a compromise: each global model column has its own hi-res Cloud Resolving Model (CRM) on a reduced domain
  - Global model nudges dynamical and moist variables in the CRM
  - CRM provided a single column of forcing to the global model
  - CRM retains a persistent internal state from one time step to the next

MMF vs Taditional Simulation

- Advantages
  - Traditional E3SM spends roughly 50% runtime in MPI in production runs
  - Traditional E3SM dynamics spends 90% runtime in latency-dominated MPI
  - CRMs do not communicate directly with one another
  - Thus, >90% of runtime has no MPI
  - Code is only about 30k lines
- Challenges
  - Original code was not well structured for correctness or threading
  - Most subroutines floated around outside modules (no parameter checking)
  - Most data was “use”d, not passed (hard to thread, dependencies unclear)
  - The loop over multiple CRM instances was outside the CRM code
  - SYPD throughput still enforces very little work per node (kernels are very small)

First Order of Business: Clean Up the Code

- Put all subroutines in modules, switch some “use” to parameters to aovid circular module dependency
  - Found several bugs with wrong number / type of parameters passed
- Allocate / deallocate module-level data to get off the stack
  - Some ghost in the machine bugs with PGI were resolved by this
  - Valgrind complained slightly less about the stacksize
- Replace “equivalence” statements with pointers
- F90-ize the borrowed ECMWF FFT routines
- Pass the entire E3SM-MMF code through valgrind
  - It now runs clean

Next: Create Fast, High-Coverage Test Suite

- Finding a bug is much faster if only 10 LOCs change could've caused it
- Created two low-res tests to cover all the code we care about
  - Both tests run at 80km grid spacing, 30 vertical levels for one model day
  - Test 1: 1-mom micro, 3-D CRM
  - Test 2: 2-mom micro, active aerosols, 2-D CRM
  - Compiling with GNU, total test time of 10 minutes on my desktop
- Create two baseline files per test
  - O0 and O3 to get the feel for how bit-level changes propagate over a model day
  - Then, compare refactored file against the diff between O0 and O3

Next: Push Loop Across CRMs Down the Callstack

- Requires redimensioning all module and high-level subroutine data to include another dimension (“ncrms”)
- Chose to make ncrms the slowest-varying dimension
  - Makes performance impact on CPU minima
  - Makes certain sub-cycling easier to handle
- Changed 20K LOC in a single GitHub Pull Request
  - Passed bit-for-bit checks in the E3SM test system
  - We managed to stay off of the “gitlost” Twitter feed

Next: Find a Way to Unify OpenACC and OpenMP

- We cannot continue to be exposed to a single compiler's bugs
- An OpenACC / OpenMP solution enables PGI, XL, Cray, and GNU*
- I'm not good with parsing or writing pseudo-compilers
  - So, we opted for a relatively simple solution using the CPP
- OpenMP 4.5 and OpenACC's "parallel loop" share much in common
  - Ad hoc unified directive from the intersection of OpenMP 4.5 and OpenACC
- Requires variadic macro functions in CPP but nothing else
  - PGI anc XL support full CPP in .F* files; GNU doesn't yet; Intel says "never!"
- Requires you to explicitly mention data flow in each kernel
  - This is tedious but useful for robustness
  - You don't always know your routine will be called with data already on the GPU

Mapping between OpenACC and OpenMP 4.5

OpenACC	OpenMP 4.5
!$acc parallel loop	!$omp target teams
gang (Loop across GPU's SMs)	distribute
worker (Outer loop within SMs)	parallel do
vector (Inner loop within SMs)	simd
copyin(), copyout(), copy()	map(to:), map(from:), map(tofrom:)
!$acc data, !$acc end data	!$omp target data, !$omp end target data
!$acc enter data, !$acc exit data	!$omp target enter data, !$omp target exit data
create(), delete()	map(alloc:), map(release:)
!$acc update host(), !$acc update device()	!$omp target update to() !$omp target update from()
async(id), wait(id) *	nowait, taskwait *

*Asynchronous behavior is very different

Mapping between OpenACC and OpenMP 4.5

OpenACC	OpenMP 4.5
!$acc parallel loop	!$omp target teams
gang (Loop across GPU's SMs)	distribute
~~worker (Outer loop within SMs)~~	parallel do
vector (Inner loop within SMs)	~~simd~~
copyin(), copyout(), copy()	map(to:), map(from:), map(tofrom:)
!$acc data, !$acc end data	!$omp target data, !$omp end target data
!$acc enter data, !$acc exit data	!$omp target enter data, !$omp target exit data
create(), delete()	map(alloc:), map(release:)
!$acc update host(), !$acc update device()	!$omp target update to() !$omp target update from()
async(id), wait(id) *	nowait, taskwait *

*Asynchronous behavior is very different

Differences b/t OpenMP 4.5 & OpenACC async Engines

- Likely the biggest difference between OpenMP 4.5 and OpenACC
- OpenACC mirrors the simplistic CUDA "stream" ideology
  - Everything is synchronized within streams, independent between streams
- OpenMP is more cumbersome, yet more capable
  - All asynchronous data and kernel clauses coordinate with "depend()" clauses
  - An op wih depend(in:var) must wait for the previous op's depend(out:var)
  - Theoretically, OpenMP compilers create GPU streams under the hood based on your explicitly given dependencies
- Since we decided to mention data per-kernel, our CPP-generated directives can use this info to create appropriate dependencies
- However, we must have separate data clauses between kernels and data statements (they have different dependencies)

Kernel Approaches for GPU porting

- Parallel dimensions
  - crm_nx, crm_ny, crm_nz (x,y,z dimensions of CRM)
  - ncrms (Number of CRM instances per compute node)
- In general, tightly nest and collapse all data-parallel loops
  - We don't know a priori what nx, ny, nz, and ncrms will be
  - Collapsing gives the most flexible performance across configurations
- Use "atomic" for race conditions (mostly in the vertical dimension)
  - Atomic used to perform awfully on K20x, no hardware double precision atomics
  - But Volta does very well with them
  - Prefix / cumulative sum must still be isolated and extracted into its own loop
- Push intermittent if-statements down the loop stack to allow collapsing
  - This typically ruins vectorization on the CPU

Data and Asynchronous Approaches for GPU porting

- Allocate and deallocate data on the GPU in each routine
  - You can't guarantee each CRM call will have the same "ncrms"
- Use "nested" data statements with the "present or" logic in Open*
  - All data statements have an implied "do this unless it's present"
  - If the data is allocated, nothing happens, so you don't have performance penalty
- Use asynchronous execution liberally
  - NOTE: This is not to overlap CPU and GPU computation
  - This is only to hide two things: (1) Kernel launch latencies; (2) cudaMalloc[Host]
- We have very small workloads in realistic simulations; high latencies
  - By launching asycnhronously, we don't see most of this latency
- Asynchronous launching makes correctness more difficult to maintain

Miscillaneous Stuff

- Managed memory performs poorly for small kernels with PGI
  - Allocations are excessively expensive, pool allocator is not performant
  - Gaps between kernels increase by 5-10x with Managed memory turned on
- XL currently cannot pin memory by default (poor CPU-GPU bandwidth)
  - You currently have to use the CUDA FORTRAN “pinned” attribute
  - They’re working on a compiler flag to fix this
- With all-asynchronous execution, cudaMalloc[Host] costs are hidden
- Currently redoing HOMME tracers/dynamics port with unified directives
- Going to complete RRTMGP radiation port in unified directives
  - RRTMGP is already ported to the E3SM-MMF code under ECP

Obligatory Performance Slide

- Previous OpenACC port gave expected GPU bandwidth improvement
  - IF we have enough work per GPU
  - We've since cleaned the code and are currently diversifying compiler support
- However, we do not have enough work per node in hi-res climate
  - In realistic simulations, we only got about 4x improvement on Summit
  - Since MPI isn't involved, this is solely due to small workload issues

Current and Future Challenges for Climate Modeling

- Our 2,000x throughput requirement is starving hardware for work
  - Each 2x horizontal refinement needs 8x more work (time step reduction)
  - But we only have 4x more data to work on
  - Thus, workload per node cuts in half for each 2x refinement
  - Eventually MPI message time dominates, and accelerators starve for work
- We really need to consider creative ways to improve realism without reducing per-node workloads
  - MMF is helpful to overcome the MPI problem but doesn't solve workload problem
  - Traditional numerics improvements can give 2-10x help but nothing fundamental
  - Deep Learning surrogate models and physics-based parameterizations may have a positive impact on intelligent dimensionality reduction
  - Simplified physics that directly target regional cloud-resolving grids
  - Direct ensemble-based approaches that relax the throughput requirement