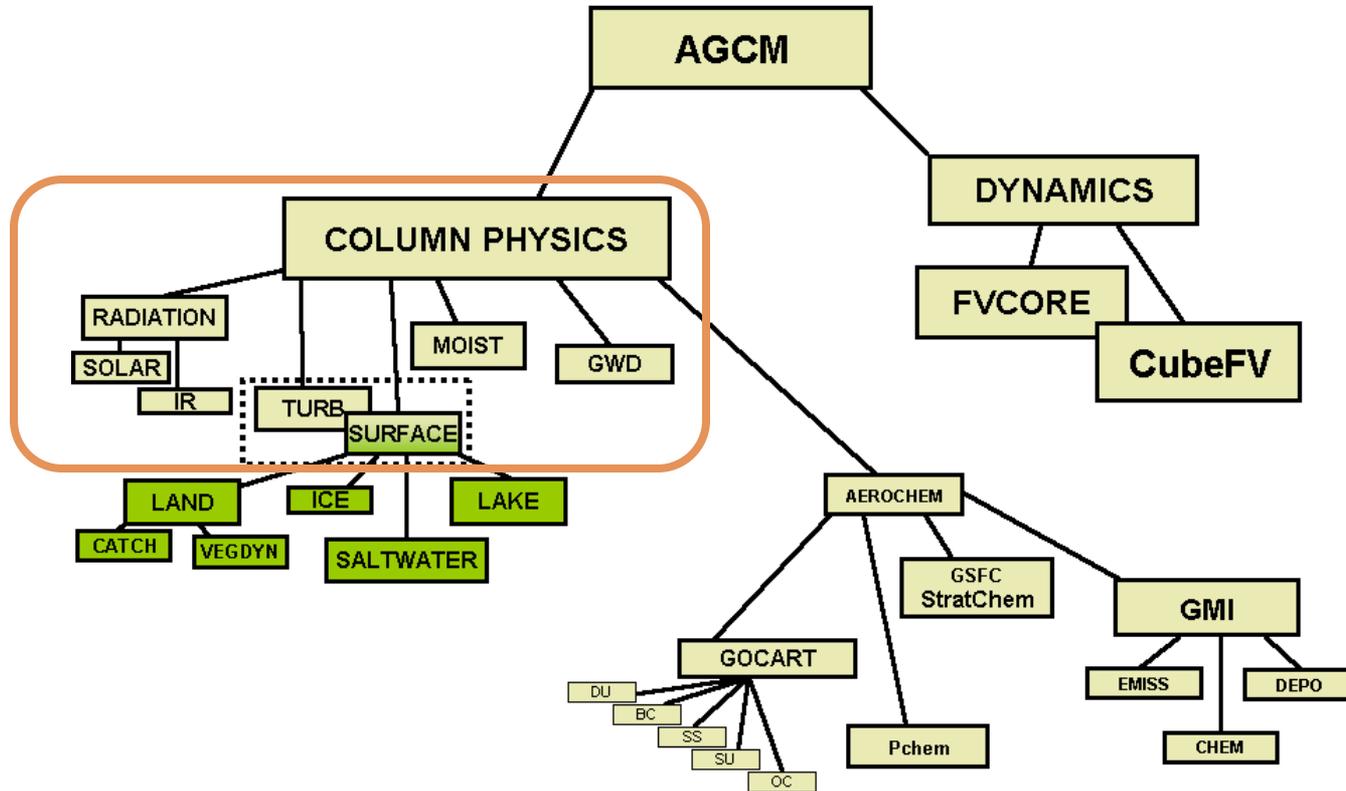


LESSONS LEARNED FROM ADAPTING GEOS-5 GCM PHYSICS TO CUDA FORTRAN

Matt Thompson – GMAO GSFC



GEOS-5 GCM



GPU Conversion Method

- Preserve bit-identical results when run on CPU if possible
 - ▣ Any changes must be approved by scientists
- Minimize disruption to end-users
 - ▣ Checkout, build, etc. should look the same
 - ▣ GPU code a compile-time decision with a flag
 - `make install BOPT=GPU`

GPU Conversion Method

- Host Code (aka “Gridded Component”) Layout
 - ▣ `#ifdef _CUDA`
 - Allocate Device Memory
 - Memory Copies to Device
 - Call GPU Kernels
 - Memory Copies to Host
 - Deallocate Device Memory
 - ▣ `#else`
 - Call CPU Kernel
 - ▣ `#endif`
- Don’t duplicate code!

GPU Conversion Method

- Device (aka "Kernel") Code Layout
 - Declare device & constant arrays (in module, use'd on host)
 - `attributes(global)` main routine
 - `#ifdef _CUDA`
 - `i = (blockidx%x - 1) * blockDim%x + threadidx%x`
 - `RUN_LOOP: if (i <= ncols) then`
 - `#else`
 - `RUN_LOOP: do i = 1, ncols`
 - `#endif`
 - `do k = 1, nlevs`
 - ...
 - Various `attributes(device)` sub-subroutines and functions
 - All levels-loop or lower! Column-loop only in main subroutine!

GPU Conversion Method

- Device Code Layout
 - ▣ Code relooped mainly for memory concerns
 - ▣ Retain current procedure layout if at all possible for less impact to scientists
 - But be cruel to dead code!
 - ▣ Retain all diagnostic capability
 - ▣ Relooped code must maintain bit-identical results on CPU (if possible)

Lesson 1: Reloop Code for Memory Access

- Explicitly Reloop Computational Code for GPUs
 - ▣ **One Thread, One Column**
 - ▣ Stride-1 index (column index)
 - Move from innermost loop (good for vectorizing compiler) to outermost loop (good for CUDA Fortran memory)
 - ▣ Stride-2 index (level index)
 - Now inside column loop

Lesson 1: Reloop Code for Memory Access

- Explicitly Reloop Computational Code for GPUs
 - ▣ Bad: Explicit loops can lose elegant, readable Fortran 90+ array syntax
 - ▣ Good: Relooping and loop fusion often leads to much less temporary space
 - `temp(ncols,nlev) → temp(nlev)`
 - `temp(ncols,nlev) → temp ... Scalarized!`
 - ▣ Good: CPUs like scalars, relooped code usually as fast or better than original

Lesson 2: External Code Calls

- Legacy code often means calls to newer, common procedures have been inserted
- CUDA doesn't like external calls to host code
- Solutions
 - ▣ Move calls that use only inputs to host and pass in results to GPU
 - ...but watch the memory costs!
 - Done with aerosol codes in Radiation
 - ▣ Internalize the calls that use intermediate results
 - Done with saturation specific humidity calls

Lesson 3: Reduce Memory Traffic

- Usual CUDA Mantra: **Memory, Memory, Memory**
- Move as much code, even trivial code, to GPU kernels if it reduces memory traffic
- Move GPU kernel code to host code if it reduces memory traffic

Lesson 4: Local Memory & Registers

- Large kernels mean large local memory needs and lots of registers
 - ▣ Good: Local memory and register spilling doesn't kill you
 - ▣ Bad: Limited amounts of local memory requires clever thinking and rewriting to achieve
 - Less important with Fermi
 - ▣ Good: Clever thinking and rewriting can lead to scalarization
 - Some temp arrays can become 2 or 3 scalars (level above, level below...)

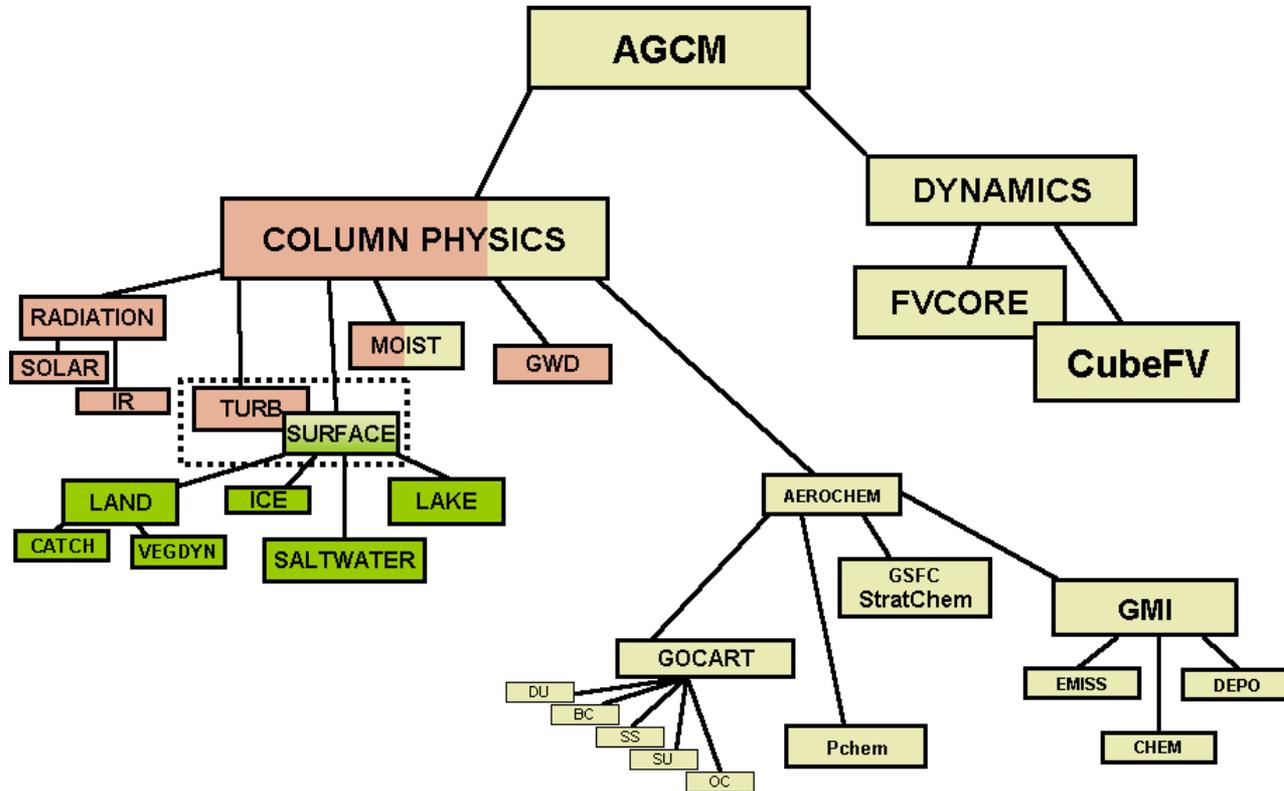
Lesson 5: Shared Memory Isn't Required

- None of the GEOS-5 Column Physics GPU kernels use Shared Memory
 - ▣ Column Physics code has no horizontal dependencies
 - ▣ One column doesn't see the one next to it
- We can use cache reconfiguring to favor L1
 - ▣ `cudaDeviceSetCacheConfig(cudaFuncCachePreferL1)`

Lesson 6: Occupancy Isn't Everything

- When converting large (1000+ line) kernels, don't obsess over the Occupancy Calculator
 - ▣ You won't like what it tells you with kB of local memory and 64 registers
 - 25%, 33%, ...
 - ▣ ...but lots of parallelism hides this!

GEOS-5 GCM Physics Converted



Results – Kernels

Kernel	Speedup
GWD	17.6x
TURBULENCE	11.9x
CLOUD	23.7x
IRRAD	26.7x / 36.4x
SORAD	45.6x / 62.5x

Includes allocation, deallocation, and data transfer times.

System: 16 Nodes, 1 CPU core (X5670), 1 GPU (M2090)
Model Run: 2 Days, 1/2-Degree

Results – Full Gridded Components

Gridded Component	Speedup
GWD	17.5X
TURBULENCE	5.5X
MOIST	2.1X
RADIATION	19.1X / 21.5X
PHYSICS	5.0X
GCM	1.8X

Includes cost of all host code pre- and post-GPU

System: 16 Nodes, 1 CPU core (X5670), 1 GPU (M2090)
Model Run: 2 Days, 1/2-Degree

Results – Full Gridded Components

Gridded Component	Speedup
GWD	17.5x
TURBULENCE	5.5x
MOIST (RAS Prediction)	7.5x
RADIATION	19.1x / 21.5x
PHYSICS	5.8x
GCM	2.0x

Includes predicted effect of RAS acceleration.

System: 16 Nodes, 1 CPU core (X5670), 1 GPU (M2090)
Model Run: 2 Days, 1/2-Degree

Current Issues – Host Code

- Some GEOS-5 code (ESMF/MAPL calls) are inherently non-GPUizable
 - ▣ Make as contiguous as possible
- Current layout/structure can be...*is* confusing to end-users
 - ▣ The curse of a single GPU developer!
 - ▣ Input from scientists will shape future layout/structure

Current Issues – GPU Memory

- GPU allocates/deallocates currently done every timestep
 - ▣ We can use Initialize/Finalize methods of GEOS-5 to do once
 - ▣ ...but doing so breaks the object-oriented nature of GEOS-5
- Constants moved to constant memory every timestep
 - ▣ Most probably can be copied once with no impact on OO

Current Issues – Host Memory

- Host arrays not using pinned memory
 - ▣ GEOS-5 arrays “transparently” allocated by first routine that needs them, pointers then passed around
 - ▣ Allocating/Deallocating pinned memory is **very** expensive, more than moving data sometimes!
- No pinning means no asynchronous movement or multi-buffering can be done
 - ▣ Fortran memory layout might make multi-buffering difficult but it must be explored

Current Issues – Diagnostics

- GEOS-5 Physics codes often have many, optional diagnostics
- Usually handled with `if(associated(diag_export_pointer))` tests to run computation in the kernel
 - ▣ CUDA doesn't know about host pointer association
 - ...yet?

Current Issues – Diagnostics

- How to do this?
 - ▣ Pass in DDT of logicals
 - Messy, means more to edit for scientists when adding new diagnostic (add entry to DDT in host, in GPU declaration...)
 - ▣ Do **all** diagnostic calculations, only copy from GPU to CPU if associated on CPU
 - Calculations “free” on GPU, cost is in memory copy time
 - CPU cost is in host memory space for temporary holding and, in CPU mode, extra calculations that were avoided

Current Issues – Science

- Current runs use CUDA fastmath and single-precision
- Need long climate runs to find if double-precision and/or no fastmath are required for good science

Thanks

- Max Suarez, Larry Takacs, Bill Putman, and the GMAO Scientists
- GMAO, NCCS, and NAS Computing Support
- PGI Support

Questions?
