PROGRESS IN ADAPTING THE GEOS-5 GCM TO CUDA FORTRAN: SUCCESSES AND CHALLENGES

Matt Thompson
Goddard Earth Observing System Model, Version 5

- Includes a Data Assimilation System (DAS)
  - Integrates the Global Climate System (GCM) with Gridpoint Statistical Interpolation (GSI)

- Focusing here on the Atmospheric GCM (AGCM)
  - Hierarchy of ESMF Gridded Components connected via MAPL
GEOS-5 GCM

Progress in Adapting the GEOS-5 GCM to CUDA FORTRAN: Successes and Challenges
GPU Conversion Aims

- Preserve bit-identical results on CPU whenever possible
- Minimize disruption to end-users
  - Checkout, build, etc. should look the same
  - GPU code a compile-time decision with a flag
GPU Conversion Method

- Current Host Code 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!
  - There is no irrad_gpu and irrad_cpu, only irrad!
GPU Conversion Method

- Device (aka “Kernel”) Code Layout
  - Declare device & constant arrays (in module, use’d on host)
  - attributes(global) main routine
    #ifdef _CUDA
    RUN_LOOP: do i = (blockidx%x-1)*blockdim%x+threadidx%x, &
    ncols, blockdim%x*griddim%x
    #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!

Progress in Adapting the GEOS-5 GCM to CUDA FORTRAN: Successes and Challenges
GPU Conversion Method

- Device Code Layout
  - Code changes mainly for memory concerns
  - Retain current procedure layout if at all possible for less impact to scientists
    - But be cruel to dead code!
  - Minimize new inputs/outputs
  - Retain all diagnostic capability
Progress in Adapting the GEOS-5 GCM to CUDA FORTRAN: Successes and Challenges
GEOS-5 GCM Targets

Progress in Adapting the GEOS-5 GCM to CUDA FORTRAN: Successes and Challenges
GEOS-5 GCM Converted

Progress in Adapting the GEOS-5 GCM to CUDA FORTRAN: Successes and Challenges
Results – Physics Kernels

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Speedup (v. Socket)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWD</td>
<td>14.7X</td>
</tr>
<tr>
<td>TURBULENCE</td>
<td>16.7X</td>
</tr>
<tr>
<td>RAS</td>
<td>2.3X</td>
</tr>
<tr>
<td>CLOUD</td>
<td>14.7X</td>
</tr>
<tr>
<td>IRRAD</td>
<td>7.0X / 9.5X</td>
</tr>
<tr>
<td>SORAD</td>
<td>9.2X / 14.6X</td>
</tr>
</tbody>
</table>

System: 24 Nodes, 1 CPU (6-core X5670), 1 GPU (M2090)
Model Run: 2 Days, ½-Degree

Only computation (no data transfer)
## Results – Physics Kernels

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Speedup (v. Socket)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWD</td>
<td>2.6x</td>
</tr>
<tr>
<td>TURBULENCE</td>
<td>1.3x</td>
</tr>
<tr>
<td>RAS</td>
<td>1.3x</td>
</tr>
<tr>
<td>CLOUD</td>
<td>2.4x</td>
</tr>
<tr>
<td>IRRAD</td>
<td>6.4x / 8.4x</td>
</tr>
<tr>
<td>SORAD</td>
<td>7.9x / 11.1x</td>
</tr>
</tbody>
</table>

Includes allocation, deallocation, and data transfer times.

**System:** 24 Nodes, 1 CPU (6-core X5670), 1 GPU (M2090)

**Model Run:** 2 Days, ½-Degree
## Results – Full Gridded Components

<table>
<thead>
<tr>
<th>Gridded Component</th>
<th>Speedup (v. Socket)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWD</td>
<td>2.6x</td>
</tr>
<tr>
<td>TURBULENCE</td>
<td>0.8x</td>
</tr>
<tr>
<td>MOIST</td>
<td>1.0x</td>
</tr>
<tr>
<td>RADIATION</td>
<td>1.7x / 1.7x</td>
</tr>
<tr>
<td>PHYSICS</td>
<td>0.9x</td>
</tr>
<tr>
<td>GCM</td>
<td>0.5x</td>
</tr>
</tbody>
</table>

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

- **System:** 24 Nodes, 1 CPU (6-core X5670), 1 GPU (M2090)
- **Model Run:** 2 Days, ½-Degree
Successes – Radiation

- Radiation codes are significantly faster; could allow us to do new science
  - Currently: Calculate fluxes every hour at lower resolutions, every half-hour at higher while dynamics (and all other physics) runs as often as every 3 minutes!
  - Future: Calculate fluxes every time step at all resolutions
Expensive cloud physics code faster as well

Exploiting will require careful thought to reduce data transfer costs

- Investigate moving some calculations back to CPU if it reduces data transfer?
Successes – Climate

Ten-Year Climate Run at 2-degrees
DJF Zonal Mean
Left - Temp
Right - RH
Challenges – Highly Branched Code

- RAS scheme:
  - ...is highly branched
    - Up to 7 levels of nested if’s
  - ...has different amounts of work by design
    - Some clouds are high, some are low; all columns can be different and have different path

- Possible Solution
  - Sort columns based on return codes from previous time steps
  - Hope is that columns with similar physical characteristics have similar code paths, so warps take same path
Challenges – Many Outputs & Diagnostics

- Cloud & Turbulence tens of outputs and diagnostics
  - Allocating 50+ large 3D arrays and associated memcpys are expensive
    - In TURB: Runtime is ~10% of overall time, rest allocs and moves
  - Possible Solution
    - Test if diagnostic is needed before copying
    - Pro: Reduces data transfer time
    - Con: Code gets uglier with extra tests everywhere
    - Con: The “default” run setup exports nearly all diagnostics, and most people run that default setup to get useful data
Challenges – Using Streams

- All code still uses Stream Zero, losing out on advantages of multiple streams
  - E.g., Asynchronous data and kernel overlapping

- Why still Stream Zero?
  - Cost of allocating pinned memory currently obviates any help streams could provide
  - Might have to revisit with Kepler
Challenges – Code Readability

- CUDA Fortran code is ugly and intrusive
  - especially how GEOS-5 implements it

- Valid complaints from other developers
  - #ifdef extravaganza
  - Add to CPU code -> Add to GPU code
    - Usually means “Call Matt”
    - Slows down work
  - Interfaces can be different due to CUDA limitations
    - Might be less important with CUDA 4.0

- Possible cleanup schemes can make code more unreadable!
Challenges – Code Readability

- Managed Heap

```fortran
ALLOCATE(Vars2d(num2dvars))
n_temp = 1
_MEMCPY(Vars2d(n_temp), temp, size(temp))
_CALL(kernel) (n_cols...)
_MEMCPY(temp, Vars2d(n_temp), size(temp))
DEALLOCATE(Vars2d)
```
Challenges – Code Readability

- Managed Heap
  - Pro: One heap allows data to stay resident on GPU longest
  - Pro: Using macros, can construct a code that is nearly CPU/GPU agnostic
    - _MEMCPY maps to either cudaMemcpy or a “cpuMemcpy” call
  - Con: Opaque (like a brick wall) to all but a few
  - Con: Requires a consistent memory placement scheme that must be adhered to rigidly
    - Slot 1 is Temperature, Slot 2 is Pressure, &c.
Challenges – Code Readability

- F2003 ASSOCIATE Block
  
  ```
  #ifdef __CUDA__
  // Allocate Device Memory
  // Memory Copies to Device
  ASSOCIATE(t=>t_dev, u=>u_dev...)
  
  #endif
  _CALL (kernel1) (t,u,...)
  _CALL (kernel2) (t,u,...)
  #ifdef __CUDA__
  END ASSOCIATE
  // Memory Copies to Host
  // Deallocate Device Memory
  #endif
  ```
Challenges – Code Readability

- **F2003 ASSOCIATE Block**
  - **Pro:** With macros, presents a single subroutine interface to multiple kernel calls
    - Not possible before CUDA 4 for us (some interfaces have 50+ members)
  - **Pro:** Some have experience of EQUIVALENCE, much the same style
  - **Con:** Memory movement is back to being an #ifdef controlled block of cudaMemcpys before and after calls
  - **Con:** Might require more abstraction of CUDA variables
Challenges – Code Portability

- Most important: CUDA Fortran is not that portable!
  - If PGI drops support, trouble!
- Possible solution to all our troubles: OpenACC!
  - ...but...
Future Directions – OpenACC – Pros

- OpenACC is a standard
  - Should look the same for any accelerator supported
- It’s like OpenMP
  - Most scientific programmers have seen OpenMP
  - Practice/Learning for Xeon Phi
- Just pragmas that are pretty readable by others
  - copy: variables copied
  - copyin: variables just copied in
Future Directions – OpenACC – Cons

- OpenACC is not designed for large, multi-nested codes
  - Requires manual inlining...
    - Pretty much a no-go
  - ...or inlining by compiler
    - Every attempt has led to ACON or other compiler errors
    - GEOS-5 might require a dedicated PGI engineer just to solve these!

- Lack of memory control (tables in constant memory) could reduce performance gains
  - But by how much?
Future Directions – OpenACC

- Try conversion of working CUDA Fortran kernel to OpenACC
  - Work with PGI on one, maybe solve issues with others
- We know the data movement, so pragmas should be easy to write
Future Directions – Kepler & CUDA 5

- Our code is highly MPI decomposed so Hyper-Q might be quite helpful
  - At present can only run one core per GPU
- Big kernels mean register spilling galore at present
- Dynamic Parallelism
  - Possible boon for RAS?
- CUDA Libraries
  - Shared code (e.g., saturation specific humidity in both Turb and Cloud) in a library prevents duplication of interfaces
Future Directions – Xeon Phi

- Our code is highly MPI decomposed so native mode could be interesting
- OpenACC attempts will inform the OpenMP for Xeon Phi
- MKL on a Xeon Phi could be worth exploring
- Questions
  - Enough memory on a Xeon Phi for a full native model?
  - Data traffic is still data traffic
  - Are the memory/loop optimizations done for CUDA bad for Xeon Phi (big loops usually aren’t vectorizer friendly...?)
Thanks

- Max Suarez
- Bill Putman
- GMAO, NCCS, and NAS Computing Support
- PGI Support
Questions? Suggestions?