CONTINUED EFFORTS IN ADAPTING THE GEOS-5 AGCM TO ACCELERATORS: SUCCESSES AND CHALLENGES
Accelerator Conversion Aims

- Code rewrites will probably be necessary, preserve bit-identical results on CPU whenever possible
- Minimize disruption to end-users
  - Checkout, build, &c., should look the same
  - Accelerated code should be a compile-time decision
- Single Code Base
  - No: irrad_cpu, irrad_gpu, irrad_mic...
  - Yes: irrad
- Make faster, not slower
GEOS-5 AGCM

Continued efforts in adapting the GEOS-5 AGCM to accelerators: successes and challenges
CONTINUED EFFORTS IN ADAPTING THE GEOS-5 AGCM TO ACCELERATORS: SUCCESSES AND CHALLENGES
GEOS-5 AGCM Converted To GPU

Continued efforts in adapting the GEOS-5 AGCM to accelerators: successes and challenges.
GPU Results – Physics Kernels

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Speedup (v. Socket)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWD</td>
<td>5.1x</td>
</tr>
<tr>
<td>TURBULENCE</td>
<td>3.0x</td>
</tr>
<tr>
<td>CLOUD</td>
<td>2.3x</td>
</tr>
<tr>
<td>IRRAD</td>
<td>3.5x / 4.4x</td>
</tr>
<tr>
<td>SORAD</td>
<td>4.6x / 6.4x</td>
</tr>
</tbody>
</table>

Includes allocation, deallocation, and data transfer times.
All speedups versus 1 socket (8 cores)

Using only 1 core to 1 GPU
System: 1 Node, 1 CPU (8-core E5-2670), 1 GPU (K20x)
Model Run: 4 Hours, ½-Degree
## GPU Results – Full Gridded Components

<table>
<thead>
<tr>
<th>Gridded Component</th>
<th>Speedup (v. Socket)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWD</td>
<td>5.1x</td>
</tr>
<tr>
<td>TURBULENCE</td>
<td>1.2x</td>
</tr>
<tr>
<td>MOIST</td>
<td>0.3x</td>
</tr>
<tr>
<td>RADIATION</td>
<td>1.0x / 1.0x</td>
</tr>
<tr>
<td>PHYSICS</td>
<td>0.5x</td>
</tr>
<tr>
<td>GCM</td>
<td>0.3x</td>
</tr>
</tbody>
</table>

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

Note: Subtracted highly variable parameterized chemistry disk read time from PHYSICS and GCM

Using only 1 core to 1 GPU

System: 1 Node, 1 CPU (8-core E5-2670), 1 GPU (K20x)
Model Run: 4 Hours, ½-Degree
GPU Challenges – MPI

- GEOS-5 GCM is fully MPI-decomposed, but GPUs (pre-K20) and MPI don’t mix well

- Choices:
  - Run one CPU core per socket for each GPU card
    - Pro: Easy to implement
    - Con: 7/8ths of your cores per socket are wasted! Amdahl would not be proud
  - Multiple MPI ranks per GPU
    - Pro: Gain back some CPU cores
    - Con: GPU pipelines kernel calls...if it can. Usually will crash as you overload the card
GPU Challenges – MPI

☐ Possible Solution: Gather/Scatter
  ☐ Before each GPU call, gather all work on one process on each CPU socket
  ☐ One core calls the GPU
  ☐ Scatter resulting data back to MPI processes

☐ Not impossible, but difficult/ponderous
  ☐ New communicators
  ☐ Extra communication load
GPU Challenges – MPI

- Possible Solution: Hyper-Q
  - Install some K20s
  - Add a few environment variables
    - `setenv CUDA_VISIBLE_DEVICES 0`
    - `setenv CUDA_MPS_CLIENT 1`
    - `setenv CUDA_MPS_PIPE_DIRECTORY /tmp/mps_0`
    - `setenv CUDA_MPS_LOG_DIRECTORY /tmp/mps_log_0`
  - Start up a proxy server
    - `/usr/bin/nvidia-cuda-mps-control -d`

- But is it that easy?

Adapted from: http://cudamusing.blogspot.com/2013/07/enabling-cuda-multi-process-service-mps.html
### GPU Hyper-Q – Physics Kernels

<table>
<thead>
<tr>
<th>Kernel</th>
<th>1 Core to 1 GPU</th>
<th>8 Cores to 1 GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWD</td>
<td>5.1x</td>
<td>6.4x</td>
</tr>
<tr>
<td>TURBULENCE</td>
<td>3.0x</td>
<td>3.9x</td>
</tr>
<tr>
<td>CLOUD</td>
<td>2.3x</td>
<td>3.1x</td>
</tr>
<tr>
<td>IRRAD</td>
<td>3.5x / 4.4x</td>
<td>4.5x / 5.0x</td>
</tr>
<tr>
<td>SORAD</td>
<td>4.6x / 6.4x</td>
<td>4.9x / 7.3x</td>
</tr>
</tbody>
</table>

Includes allocation, deallocation, and data transfer times.

All speedups versus 1 socket (8 cores)

---

System: 1 Node, 1 CPU (8-core E5-2670), 1 GPU (K20x)
Model Run: 4 Hours, ½-Degree
GPU Hyper-Q – Full Gridded Components

<table>
<thead>
<tr>
<th>Grid Comp</th>
<th>1 Core to 1 GPU</th>
<th>8 Cores to 1 GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWD</td>
<td>5.1x</td>
<td>6.4x</td>
</tr>
<tr>
<td>TURBULENCE</td>
<td>1.2x</td>
<td>2.7x</td>
</tr>
<tr>
<td>MOIST</td>
<td>0.3x</td>
<td>1.8x</td>
</tr>
<tr>
<td>RADIATION</td>
<td>1.0x / 1.0x</td>
<td>2.5x / 2.7x</td>
</tr>
<tr>
<td>PHYSICS</td>
<td>0.5x</td>
<td>2.1x</td>
</tr>
<tr>
<td>GCM</td>
<td>0.3x</td>
<td>1.3x</td>
</tr>
</tbody>
</table>

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

Note: Subtracted highly variable parameterized chemistry disk read time from PHYSICS and GCM

CONTINUED EFFORTS IN ADAPTING THE GEOS-5 AGCM TO ACCELERATORS: SuCCESSES AND CHALLENGES

System: 1 Node, 1 CPU (8-core E5-2670), 1 GPU (K20x)
Model Run: 4 Hours, ½-Degree
GPU Hyper-Q Results

- **Physics Kernels**
  - Kernel run timings same 1 core vs 8 cores
  - Data transfer times less with 8 cores, amount of data is the same -> Hyper-Q overlaps communication?

- **Dynamics**
  - Cubed-sphere dynamics not included here due to PGI limitations (compiler errors)
  - Expect similar results as with physics

- **Full Gridded Components**
  - With Hyper-Q, GPU acceleration of a highly MPI decomposed GEOS-5 becomes viable for first time
GPU Challenges – Code Readability

- CUDA Fortran code can be ugly and intrusive
  - ...especially how GEOS-5 (aka how Matt) implements it
- Valid complaints from other developers
  - `#ifdef` extravaganza
  - Add to CPU code -> Add to GPU code
    - Usually means “Call Matt”
    - Slows down work
- Possible cleanup schemes can make code more unreadable!
GPU Future Directions – OpenACC

- Try conversion of working CUDA Fortran kernels and code to OpenACC: Turbulence
  - Turbulence is a mix of one long kernel and many smaller inlined segments that became kernel subroutines
  - We can use ASSOCIATE blocks to simplify CUDA Fortran section and OpenACC elsewhere, returning to inline as before
  - Eventually use OpenACC on main kernel...which has many subroutine calls!

- We know the data movement, so data pragmas should be easy to write
GPU Future Directions – OpenACC – Pros

- OpenACC is a standard
  - Should look the same for any accelerator supported

- It looks (sort of) like OpenMP
  - Most scientific programmers have seen OpenMP
  - Practice/Learning for Xeon Phi

- Pragmas are pretty readable by others
  - copy: variables copied in and out
  - copyin: variables just copied in
GPU Future Directions – OpenACC – Cons

- OpenACC 1.0 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!

- Pro: OpenACC 2.0 is almost here: !$acc routine
Xeon Phi: Shortwave Radiation Kernel

- First step: Construction of offline tester of shortwave radiation kernel (sorad)
  - Actual CPU/GPU code in our model
  - Uses actual data inputs from our model
- Start by using offload model with Intel directives
  - Any modifications must preserve CPU and GPU capabilities
  - Strive for bit-identical results if possible
- Transition offload knowledge to full model
Xeon Phi: Shortwave Radiation Kernel

- In main sorad.F90:
  - !DIR$ ATTRIBUTES OFFLOAD : mic :: sorad in both caller and sorad kernel, similar for deledd in sorad
  - !DIR$ OMP OFFLOAD TARGET(mic) for main subroutine
  - The usual !$OMP directives to enable OpenMP

- In all, about 5 changes
Xeon Phi: Shortwave Radiation Kernel

- Constant Data Modules
  - Add !DIR$ ATTRIBUTES OFFLOAD : mic :: <constant>

- Make system changes for MIC
  - Add new MICOPT = -openmp to be added to FOPT only in MIC enabled code
    - Some parts of GEOS-5 has OpenMP we don’t want to enable
  - Use xiar -qoffload-build when linking (only in full model)

- If not on MIC, must send -no-offload
Comparison of SW radiation kernel on various platforms

CONTINUED EFFORTS IN ADAPTING THE GEOS-5 AGCM TO ACCELERATORS: SUCCESSES AND CHALLENGES
Xeon Phi: Shortwave Radiation In Full Model

CPU-MIC Difference of dT/dt from Shortwave Radiation in K/day after 6 years

Contours 0-17 K/day (top)
Xeon Phi: Future Directions

- We know the data movement from CUDA, makes the OpenMP easier to implement
- OpenMP 4.0 available soon/now
  - All Intel Directives will be moved to OpenMP 4.0
- More benchmarking with full model
  - Test scatter v balanced v compact, other options
- With tester: Use VTune to look at the code in detail to examine alignment, etc.
Xeon Phi: Challenges

- Currently using one core per MIC like the GPU days, how do we duplicate Hyper-Q success?
  - Could use 8-16 cores to one MIC, but that means running 15-30 threads per CPU core
  - Srinath showed this might be useful to explore as well in native

- Enabling MIC Offload in model requires turning off solar load-balancer
  - “offload error: offload data transfer supports only a single contiguous memory range per variable”; yet only one MPI rank is talking to one MIC…right?

- Intel rightly prefers Native not Offload for MIC but GEOS-5 is, at its heart, an I/O heavy model
  - How do we handle outputting GBs of data per step? Full Native seems impossible: Symmetric will be needed
Xeon Phi: Challenges - Libraries

- GEOS-5 AGCM just to compile requires:
  - JPEG, ZLIB, szlib, cURL, HDF4, HDF5, NetCDF, NetCDF-Fortran, UDUNITS2, ESMF
- Not impossible, mainly issues with autotools and libraries (cross-compiling)
  - And what about “make test”? 
- GEOS-5 is also a DAS:
  - SciPy, CMOR, HDFEOS, HDFEOS5, SDPToolkit...
Sandy Bridge: Shortwave Radiation Kernel

- GPUs and Xeon Phis are not the only multi-core architecture out there that can be accelerated
- What about good ol’ vectorizing and AVX? Can the compiler do something for us with a just a bit of work?

- Working with Dan Kokron of NAS, shortwave radiation kernel modified to better expose vectorization
Sandy Bridge: Shortwave Radiation Kernel

- **Step 1: Compiler options**
  - Before: `-O3`
  - After: `-O3 -axAVX -xSSE4.1 -align array32byte`
    - `-ip` seems to help in a stand-alone tester, but in full model seems to not do much
  - Also: `-fpe0` can limit compiler, lowering to `-fpe1` or `-fpe3` can help...
  - ...but beware you might depend on that behavior and expose issues!

- **Step 2: Use `-vec-reportn` to examine the what the compiler is doing**
Step 3: Code Changes

- !DIR$ ATTRIBUTES FORCEINLINE expensive, scalarized subroutine called multiple times
- For flux integrations, create per-band accumulators that are local to each column, then at end, sum into final fluxes in a SIMD fashion
- Remove loop-bound dependencies preventing vectorization (see vec-report)
- Use !DIR$ SIMD directives to help compiler identify true SIMD loops...
  - ...but be careful. You are telling the compiler you’re smarter than it is!
- Use !DIR$ NOVECTOR and !DIR$ NOFUSION directives to make sure the compiler isn’t too aggressive
In all, about 40 lines additional code in a 1944-line code (excluding code cleanup)

Results (c360, 1-day, 384 cores, Xeon E5-2670)
- Original Code: 282.101 s
- Vector Code: 144.150 s
  - ~2x speedup!
  - Not quite zero-diff (investigating)

Haven’t looked at alignment in depth yet

Vectorizing success on Sandy Bridge doesn’t seem to directly translate on Xeon Phi, needs more investigation
Sandy Bridge: Shortwave Radiation Kernel

- Is sorad a unique success?
  - Code was already nearly fully inlined due to heritage of PGI Accelerator pragma work
  - Only one, very expensive, all scalar internal subroutine to inline
  - Other GPU kernels have been highly scalarized and localized (like sorad, main loop over columns)
- Preliminary work by Dan Kokron on longwave code shows benefits to a careful analysis
  - But is more complex (hoisting calculations, etc.), not just add a few lines/directives
Thanks

- Max Suarez & Bill Putman of GMAO
- Dan Kokron of NAS
- Carl Ponder of NVIDIA
- Mike Greenfield & Indraneil Gokhale of Intel
- GMAO, NCCS, and NAS Computing Support
- NVIDIA/PGI Support
Questions?