Experiences with CUDA & OpenACC from porting ACME to GPUs

Matthew Norman  ORNL
Irina Demeshko  Sandia
Jeffrey Larkin  Nvidia
Aaron Vose  Cray
Mark Taylor  Sandia
Experiences with CUDA & OpenACC from porting ACME to GPUs

Matthew Norman  |  ORNL
Irina Demeshko  |  Sandia
Jeffrey Larkin  |  Nvidia
Aaron Vose      |  Cray
Mark Taylor     |  Sandia
What are ACME & CAM-SE

- **ACME**: Accelerated Climate Model for Energy
  - Began as fork from CESM; geared toward DOE science targets
  - Global climate simulation for leadership computing

- **CAM-SE**: atmospheric component of ACME; most costly
  - (1) a dynamical core and (2) physics packages

- At hi-res, dynamics costs dominate, mostly tracer transport

- Cubed-Sphere + Spectral Element
- 4x4 bases per element
- Typically 120x120 elems per panel
- Roughly ¼ degree grid spacing

http://www-personal.umich.edu/~paullric/A_CubedSphere.png
OLCF’s Center for Accelerated Application Readiness (CAAR)

• PI: Dave Bader;  Point of Contact: Mark Taylor
• Preparing ACME to run efficiently on OLCF’s Summit
• Targeting OpenACC porting of:
  – CAM-SE tracers and dynamics
  – MPAS-O tracers and dynamics
  – Super-parametrization: 2-D vertical slice models for moist physics
• CAM-SE tracers now completed; dynamics soon to follow
• Eventually transition to OpenMP as it becomes available
• Manage data placement and movement on unified memory
• Identify performance maintainable best practices
Why OpenACC?

• OpenMP4 doesn’t work for FORTRAN yet

• More portable than CUDA
  – Two compilers available: PGI & Cray
  – PGI developing CPU and MIC targets for OpenACC
  – Similar code would likely perform well in OpenMP for MICs & GPUs

• More Maintainable than CUDA
  – No incomprehensible kernel code w/ shared memory & syncthreads()
  – Exact same code runs correctly on the CPU (helpful for debugging)
  – Easier to perform more incremental changes

• Data management considerably easier

• Easier to port large sections of code quickly
  – Optimizations done mainly via loop restructuring
  – In 8 workdays, all tracer routines ported & optimized from scratch
Software Engineering Concerns

• Performance on multiple architectures and maintainable code

• Performance considerations affecting code structure:
  – CAM-SE has inner loops with only 16 vectorizable iterations
    
    ```
    do j = 1 , np
        do i = 1 , np
            ...
    ```

    Work moves in & out many of these loops, surrounded by outer loops
  – Vectorization: Need enough data-independent work to hide memory latency (GPUs: at least 256-512 threads per SM)
  – Solution: Push all or part of vertical level loop down callstack
    • Sometimes, this overflows cache space on CPUs
Software Engineering Concerns

• Performance on multiple architectures and maintainable code

• Performance considerations affecting code structure:
  – Outer loop(s) span many routines: “one giant GPU kernel”
  – Registers: Large kernels suffer increased register spillage
  – L1 Cache: K20x GPUs require “shared memory” to use L1 cache
    • Shared Memory makes kernels ad hoc (no reusable code)
      – Must explicitly create temp variable, make it gang private only, and then use !$acc cache()
    • K40 and newer do not require shared memory for L1 cache use
      – Giant kernels require !$acc routine(), which is a black hole of bugs
      – Solution: Push all loops down the callstack
        • Enables reusable routines that can be easily optimized separately
        • However, destroys cache locality on CPUs
Software Engineering Concerns

• Performance on multiple architectures and maintainable code

• Performance considerations affecting code structure:
  – Original code uses OpenMP parallel regions
  – Reduces threading overheads compared to loop-level OpenMP
  – OpenACC doesn’t play nicely with OpenMP parallel regions
  – Also, dynamics has so little work, cannot afford to split kernels further
  – Solution: Aggregate work to master thread
Performance Portability

- **Exact** same code performing on all architectures is a pipe dream
- Is it enough to just avoid being really slow on a given architecture?
  - Many domains would not accept this as OK
  - When burning $10^9$ core hours, optimization matters
- How about “performance maintainable” (credit: Markus Eisenbach)
- Code can be sensibly maintained while allowing room for optimizing for different architectures
Performance Portability

• **Exact** same code performing on all architectures is a pipe dream

• Is it enough to just avoid being really slow on a given architecture?
  – Many domains would not accept this as OK
  – When burning $10^9$ core hours, optimization matters

• How about “performance maintainable” (credit: Markus Eisenbach)

• Code can be sensibly maintained while allowing room for optimizing for different architectures

• Each architecture should have:
  – Sensible looking code w/ optimizations (no CUDA, !DIR$, etc)
  – Must use tools that are on the hook with OLCF contracts
  – Identical interfaces to duplicated routines
  – Clearly identifiable operations (divergence, laplace, pack, etc)
  – Branch at the lowest level possible
ACME Performance Maintainability

- Default code is for CPU
- OpenACC code goes into src/share/openacc directory
- OpenACC modules suffixed with "_openacc"
  - But routine names remain identical
- Use a "switch_mod" to choose correct module based on preprocessor macros
  - Hides distracting #ifdef’s from the rest of the code
- Functions transformed to subroutines to allow array reshaping
- Each call specifies a length (nlev or nlev*qsize), a range of elements, and time level, even if all are simple "1"
- Pushing all loops down the callstack now
ACME Performance Maintainability

• Pushing all loops down the callstack means routines have to know the structure of “element_t”
  e.g., elem(ie)%state%qdp(i,j,k,q,nt)

• If a routine has to name the variable, it is not reusable

• Solution: Make variables flat arrays, and point into flat array from element_t data structure:
  allocate(qdp_flat(np,np,nlev,qsize,2,nelemd))
  do ie = 1 , nelemd
      elem(ie)%state%qdp => qdp_flat(:,:,,:,:,:,ie)
  enddo

• This will likely affect caching efficiency on CPUs, so it’s only done if USE_OPENACC macro is set to 1

• Only variables passed to a reusable routine are flattened
Current OpenACC performance

• Before optimization, MPI exchange took 85% of total tracer time
  – For 32 elements per node (2 per CPU core on Titan)
  – PCI-e copies of MPI buffers were not pinned & not overlapped w/ MPI
• First, pinned everything with “-ta=tesla,pin”
• We overlapped PCI-e and MPI exchanges via polling loop
• MPI exchanges consume 40% for OpenACC and 22% for CPU
• Interlagos vs. K20x (Titan); Flags: -fastsse -Mvect

<table>
<thead>
<tr>
<th>Speed-Ups</th>
<th>Elems / Node</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>tracers</td>
<td>1.94x</td>
<td>2.28x</td>
<td>2.37x</td>
<td></td>
</tr>
<tr>
<td>dycore</td>
<td>1.62x</td>
<td>1.84x</td>
<td>1.87x</td>
<td></td>
</tr>
</tbody>
</table>
Numerics for Large Scale Computing

- We need to invest in higher-order methods (very high-order)
- Low-order methods only benefit from DRAM bandwidth increase
- Accelerators were made for compute bound computations
- Higher-order methods re-use data & increase compute intensity
- There are ways to improve resolution of discontinuities as order increases
Topics

• Latency sensitivity of dynamics

• Need for higher-order accuracy to make climate & weather compute-bound