Porting CAM-SE To Use Titan’s GPUs

2014 Multicore Heterogeneous Workshop

Matthew Norman  ORNL
Valentine Anantharaj  ORNL
Richard Archibald  ORNL
Ilene Carpenter  NREL
Katherine Evans  ORNL
Jeffrey Larkin  Nvidia
Paulius Micikevicius  Nvidia
Aaron Vose  Cray
What is CAM-SE

• Climate-scale atmospheric simulation for capability computing
• Comprised of (1) a dynamical core and (2) physics packages
What is CAM-SE

- Climate-scale atmospheric simulation for capability computing
- Comprised of (1) a dynamical core and (2) physics packages

Dynamical Core

1. “Dynamics”: wind, energy, & mass
2. “Tracer” Transport: (H\text{2}O, CO\text{2}, O\text{3}, …)
   Transport quantities not advanced by the dynamics
What is CAM-SE

- Climate-scale atmospheric simulation for capability computing
- Comprised of (1) a dynamical core and (2) physics packages

Dynamical Core

1. “Dynamics”: wind, energy, & mass
2. “Tracer” Transport: (H$_2$O, CO$_2$, O$_3$, ...)
Transport quantities not advanced by the dynamics

Physics Packages

Resolve anything interesting not included in dynamical core (moist convection, radiation, chemistry, etc)
• Cubed-Sphere + Spectral Element
• Each cube panel divided into elements

http://www-personal.umich.edu/~paulric/A_CubedSphere.png
• Cubed-Sphere + Spectral Element
• Each cube panel divided into elements
• Elements spanned by basis functions
Gridding & Numerics

- Cubed-Sphere + Spectral Element
- Each cube panel divided into elements
- Elements spanned by basis functions
- Basis coefficients describe the fluid

http://www-personal.umich.edu/~paullric/A_CubedSphere.png
Gridding & Numerics

- Cubed-Sphere + Spectral Element
- Each cube panel divided into elements
- Elements spanned by basis functions
- Basis coefficients describe the fluid

Used CUDA FORTRAN from PGI

NEW! → Have run a few OpenACC experiments with Cray

http://www-personal.umich.edu/~paulrnc/A_CubedSphere.png
Life in the Real World: Codes Change

- Total: 2.6
- Tracers: 3.6
- Euler step: 2.9
- Vertical remap: 5.4
- Hyperviscosity: 4.2
Life in the Real World: Codes Change

- Vertical Remap basically removed

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>2.6</td>
</tr>
<tr>
<td>Tracers</td>
<td>3.6</td>
</tr>
<tr>
<td>Euler step</td>
<td>2.9</td>
</tr>
<tr>
<td>Vertical remap</td>
<td>5.4</td>
</tr>
<tr>
<td>Hyperviscosity</td>
<td>4.2</td>
</tr>
</tbody>
</table>
Life in the Real World: Codes Change

- Vertical Remap basically removed
- New backend for CUDA FORTRAN
• Vertical Remap basically removed
• New backend for CUDA FORTRAN
• New sub-cycling methods implemented (More PCI-e traffic)
• Vertical Remap basically removed
• New backend for CUDA FORTRAN
• New sub-cycling methods implemented (More PCI-e traffic)
• New science targets identified
Life in the Real World: Codes Change

- Vertical Remap basically removed
- New backend for CUDA FORTRAN
- New sub-cycling methods implemented (More PCI-e traffic)
- New science targets identified
- Moral of the story: your port must be flexible and maintainable
Transitioning to OpenACC

• Issue # 1: Threading
  – CAM-SE threaded via element chunking (“parallel” only, not “parallel do”)
  – Cray, therefore, doesn’t complain about it
  – BUT, very strange timings occur
  – Cray does complain about OpenACC being inside an “omp master”

• Solution
  – Make your own master
    !$omp barrier
    if (hybrid%ithr == 0) then
      ...
    endif
    !$omp barrier
  – Change loop bounds to cover all elements on the node
Transitioning to OpenACC

• Issue # 2: Internal Compiler (Cray) / “I’m not touching it” (PGI) Errors
  – Perhaps a passive-aggressive way to frustrate you into using tightly nested loops?
  – Original CPU Code below:

```plaintext
do ie = 1, nelemd    ! nelemd = 64
  do q = 1 , qsize    ! qsize = 30
    do k = 1 , nlev    ! nlev = 30 and i,j = 1,np (np = 4)
      gradQ(:,:,1) = vstar(:,:,1,ie) * elem(ie)%Qdp(:,:,k,q,n0_qdp)
      gradQ(:,:,2) = vstar(:,:,2,ie) * elem(ie)%Qdp(:,:,k,q,n0_qdp)
      dp_star(:,:,i,j) = divergence_sphere( gradQ, ... )
      Qtens(:,:,k,q,ie) = elem(ie)%Qdp(:,:,k,q,n0_qdp) - dt * dp_star(:,:,i,j)
    enddo
  enddo
enddo
```

– GPU Problem: Separate trips thorugh “i-j loop”, which only has 16 threads
– Reason is mainly due to “divergence_sphere” works over i & j loops
– Solution: Have divergence_sphere work over one thread
Transitioning to OpenACC

- Version 1: Break computation up over blocks of “np x np x nlev”

```c
!$acc parallel loop gang private(gradQ) collapse(2)
  do ie = 1 , nelemd
    do q = 1 , qsize
      !$acc loop collapse(3) vector
        do k = 1 , nlev
          do j = 1 , np
            do i = 1 , np
              gradQ(i,j,k,1) = vstar(i,j,k,1,ie) * elem(ie)%Qdp(i,j,k,q,n0_qdp)
              gradQ(i,j,k,2) = vstar(i,j,k,2,ie) * elem(ie)%Qdp(i,j,k,q,n0_qdp)
            enddo
          enddo
        enddo
      !$acc loop collapse(3) vector
        do k = 1 , nlev
          do j = 1 , np
            do i = 1 , np
              Qtens(i,j,k,q,ie) = elem(ie)%Qdp(i,j,k,q,n0_qdp) - dt * &
              divergence_sphere( gradQ(:,::,:,:), ... )
            enddo
          enddo
        enddo
    enddo
  enddo
enddo
```
Transitioning to OpenACC

- Version 1: Break computation up over blocks of “np x np x nlev”

```c
!$acc parallel loop gang private(gradQ) collapse(2)
do ie = 1 , nelemd
  do q = 1 , qsize
    !$acc loop collapse(3) vector
    do k = 1 , nlev
      do j = 1 , np
        do i = 1 , np
          gradQ(i,j,k,1) = Vstar(i,j,k,1,ie) * elem(ie)%Qdp(i,j,k,1,ie)
          gradQ(i,j,k,2) = Vstar(i,j,k,2,ie) * elem(ie)%Qdp(i,j,k,2,ie)
        enddo
      enddo
    enddo
  enddo
!$acc loop collapse(3) vector
do k = 1 , nlev
  do j = 1 , np
    do i = 1 , np
      Qtens(i,j,k,q,ie) = elem(ie)%Qdp(i,j,k,q,n0_qdp) - dt * &
      divergence_sphere( gradQ(:,:, :, :) , ... )
    enddo
  enddo
enddo
enddo
enddo
```

Placed in per-SM shared memory on the GPU? Nope!
Transitioning to OpenACC

• Version 1: Break computation up over blocks of “np x np x nlev”

```c
!$acc parallel loop gang private(gradQ) collapse(2)
do ie = 1 , nelemd
do q = 1 , qsize
!$acc loop collapse(3) vector
do k = 1 , nlev
do j = 1 , np
do i = 1 , np
    gradQ(i,j,k,1) = vstar(i,j,k,1,ie) * elem(ie)%Qdp(i,j,k,q,n0_qdp)
    gradQ(i,j,k,2) = vstar(i,j,k,2,ie) * elem(ie)%Qdp(i,j,k,q,n0_qdp)
endo
dendo
dendo

!$acc loop collapse(3) vector
do k = 1 , nlev
do j = 1 , np
do i = 1 , np
    Qtens(i,j,k,q,ie) = elem(ie)%Qdp(i,j,k,q,n0_qdp) - dt * & divergence_sphere( gradQ(:,:,:,:) , ... )
endo
dendo
dendo
dendo
dendo
```

Effectively inserts “__syncthreads()” between blocks
Transitioning to OpenACC

• Version 1: Break computation up over blocks of “np x np x nlev”

```c
$acc parallel loop gang private(gradQ) collapse(2)
do ie = 1, nelemd
  do q = 1, qsize
    !$acc loop collapse(3) vector
do k = 1, nlev
  do j = 1, np
    do i = 1, np
      gradQ(i,j,k,1) = Vstar(i,j,k,1,ie) * elem(ie)%Qdp(i,j,k,q,h0_qdp)
      gradQ(i,j,k,2) = Vstar(i,j,k,2,ie) * elem(ie)%Qdp(i,j,k,q,h0_qdp)
    enddo
  enddo
enddo
```  

• Had to grow local arrays to include “nlev” array index

• Had to duplicate some work in divergence_sphere

• Still performed 2.5x faster than Cray with all optimizations on
Transitioning to OpenACC

- Version 2: One big loop, duplicate even more work

```c
!$acc parallel loop gang vector collapse(5)
do ie = 1, nelemd
  do q = 1, qsize
    do k = 1, nlev
      do j = 1, np
        do i = 1, np
          Qtens(i,j,k,q,ie) = elem(ie)%state%Qdp(i,j,k,q,n0_qdp) - dt &
                          divergence_sphere_2( Vstar(:,:,,:,:,ie), &
                          elem(ie)%Qdp(:,:,,:,:,q,n0_qdp), ... )
        enddo
      enddo
    enddo
  enddo
enddo
```
Transitioning to OpenACC

• Version 2: One big loop, duplicate even more work

```c
!$acc parallel loop gang vector collapse(5)
do ie = 1, nelemd
    do q = 1, qsize
        do k = 1, nlev
            do j = 1, np
                do i = 1, np
                    Qtens(i,j,k,q,ie) = elem(ie)%state%Qdp(i,j,k,q,n0_qdp) + dt * &
                    divergence_sphere_2(Vstar(:,:,:,:,ie), &
                    elem(ie)%dp(:,q,n0_qdp), ...)
                enddo
            enddo
        enddo
    enddo
enddo
```

• Performed **3.2x faster than Cray** with all optimizations on
Transitioning to OpenACC

• Version 3: Improve access to GPU DRAM by moving indices

• Same as Version 2, but divergence_sphere changes:

```fortran
do s=1,np
    dudx00 = dudx00 + dvv(s,i)*metdet(s,j)*qdp(s,j,k)*&
             ( dinv(1,1,s,j)*vstar(s,j,k,1) + dinv(1,2,s,j)*vstar(s,j,k,2) )
    dvdy00 = dvdy00 + dvv(s,j)*metdet(i,s)*qdp(i,s,k)*&
             ( dinv(2,1,i,s)*vstar(i,s,k,1) + dinv(2,2,i,s)*vstar(i,s,k,2) )
enddo
```

```
23  Presentation_name
```
Transitioning to OpenACC

• Version 3: Improve access to GPU DRAM by moving indices

• Same as Version 2, but divergence_sphere changes:

\[
\text{do } s=1, np \\
\text{dudx00 } = \text{dudx00 } + \text{dvv}(s,i) \times \text{metdet}(s,j) \times \text{qdp}(s,j,k) \times \& \\
( \text{dinv}(1,1,s,j) \times \text{vstar}(s,j,k,1) + \text{dinv}(1,2,s,j) \times \text{vstar}(s,j,k,2) ) \\
\text{dvdy00 } = \text{dvdy00 } + \text{dvv}(s,j) \times \text{metdet}(i,s) \times \text{qdp}(i,s,k) \times \& \\
( \text{dinv}(2,1,i,s) \times \text{vstar}(i,s,k,1) + \text{dinv}(2,2,i,s) \times \text{vstar}(i,s,k,2) ) \\
\text{enddo}
\]

• \textbf{Performed 31x slower than Cray with all optimizations on}

• \textbf{Cray “.lst” output looks identical}

• \textbf{Why? No idea as of yet}
How Code Refactoring Affects CPU

• Tabulating average kernel runtimes for CPU / ACC versions

<table>
<thead>
<tr>
<th>Configuration</th>
<th>ACC Time</th>
<th>ACC Speedup</th>
<th>CPU Time</th>
<th>CPU Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>---</td>
<td>---</td>
<td>0.934 ms</td>
<td>---</td>
</tr>
<tr>
<td>Version 1</td>
<td>0.380 ms</td>
<td>2.46x faster</td>
<td>0.966 ms</td>
<td>1.03x slower</td>
</tr>
<tr>
<td>Version 2</td>
<td>0.258 ms</td>
<td>3.62x faster</td>
<td>3.74 ms</td>
<td>4.00x slower</td>
</tr>
<tr>
<td>Version 3</td>
<td>29.1 ms</td>
<td>31.2x slower</td>
<td>3.81 ms</td>
<td>4.08x slower</td>
</tr>
</tbody>
</table>

• Version 1 gave ACC speed-up while barely affecting CPU runtime
  — However, it is likely quite ad hoc for the GPU (e.g., vector over np x np x nlev)

• Version 2 quite negatively affected CPU runtime
  — However, it would likely perform well on other accelerators
### Comparing The Kernel Versions

- Properties of the various kernels

<table>
<thead>
<tr>
<th>Version</th>
<th>Speed-Up</th>
<th>Stack</th>
<th>Spill Stores</th>
<th>Spill Loads</th>
<th>Registers</th>
<th>Occ.</th>
<th>Share Mem</th>
<th>Const Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.46x faster</td>
<td>208</td>
<td>80</td>
<td>80</td>
<td>44</td>
<td>0.25</td>
<td>0</td>
<td>540</td>
</tr>
<tr>
<td>2</td>
<td>3.62x faster</td>
<td>208</td>
<td>80</td>
<td>80</td>
<td>48</td>
<td>0.25</td>
<td>0</td>
<td>532</td>
</tr>
<tr>
<td>3</td>
<td>31.2x slower</td>
<td>208</td>
<td>80</td>
<td>80</td>
<td>44</td>
<td>0.562</td>
<td>2048</td>
<td>612</td>
</tr>
<tr>
<td>3 (vec=64)</td>
<td>14.1x slower</td>
<td>208</td>
<td>80</td>
<td>80</td>
<td>44</td>
<td>0.5</td>
<td>1024</td>
<td>612</td>
</tr>
<tr>
<td>3 (vec=32)</td>
<td>12.5x slower</td>
<td>208</td>
<td>80</td>
<td>80</td>
<td>44</td>
<td>0.25</td>
<td>512</td>
<td>612</td>
</tr>
</tbody>
</table>
Morals Of The Story

• Fully collapsed, tightly nested loops: Good
  – All levels of parallelism trivially exposed: compiler can have its way
  – Often, recomputation can remove dependence
  – However, there are infeasible situations (hydrostasis, limiting, etc.)

• Dependence within the loop: Not as good
  – Parallelism in “chunks” – you must decide where to split loops
  – Loop splitting is likely ad hoc to the architecture (but not entirely sure)
  – Need compilers to more intelligently use shared memory

• Regarding using a single source
  – It appears that strong OpenACC optimizations kill CPU performance
  – Tightly nested loops may equally perform on various accelerators
  – If split nests can be coded flexibly, they may be performance portable