A ‘hybrid’ GPU implementation of the cubed-sphere finite-volume dynamics in GEOS-5

William Putman
Global Modeling and Assimilation Office
NASA GSFC

Programming weather, climate, and earth-system models on heterogeneous multi-core platforms- Boulder, CO
Outline

- Where we were last year
- Current status
- Directives/Direct CUDA – Examples
- Test Case
- Results/Scaling
- Future

Development Platform

NASA Center for Climate Simulation
GPU Cluster

32 Compute Nodes
- 2 Hex-core 2.8 GHz Intel Xeon Westmere Processors
- 48 GB of memory per node
- 2 NVIDIA M2070 GPUs
- dedicated x16 PCIe Gen2 connection
- Infiniband QDR Interconnect

64 Graphical Processing Units
- 1 GPU (M2070)
- 448 CUDA cores
- ECC Memory
- 6 GB of GDDR5 memory
- 515 Gflop/s of double precision floating point performance (peak)
- 1.03 Tflop/s of single precision floating point performance (peak)
- 148 GB/sec memory bandwidth
- 1 PCIe x16 Gen2 system interface

http://www.nccs.nasa.gov/gpu_front.html
Motivation

Global Cloud Resolving GEOS-6

- Pushing the resolution of global models into the 10- to 1-km range
- GEOS-5 can fit a 5-day forecast at 10-km within the 3-hour window required for operations using 12,000 Intel Westmere cores
- At current cloud-permitting resolutions (10- to 3-km) required scaling of 300,000 cores is reasonable (though not readily available)
- To get to global cloud resolving (1-km or finer) requires order 10-million cores
- Weak scaling of cloud-permitting GEOS-5 model indicates need for accelerators
- ~90% of those computations are in the dynamics

3.5-km GEOS-5 Simulated Clouds

Weak Scaling of a Cloud-Resolving GEOS-5 Model

10-km Global Mesoscale Simulation w/ Interactive Aerosols
Where we were Last Year

Idealized FV advection kernel

- An offline Cuda C demonstration kernel was developed for the 2-D advection scheme
- For a 512x512 domain, up to 80x speedup
- **Caveats:** Written entirely on the GPU (no data transfers)
  - Single CPU to Single GPU speedup
  - compares Cuda C to C code
Where we were Last Year

Idealized FV advection kernel

• An offline Cuda C demonstration kernel was developed for the 2-D advection scheme
• For a 512x512 domain, up to 80x speedup
• Caveats: Written entirely on the GPU (no data transfers)
  Single CPU to Single GPU speedup compares Cuda C to C code

Direct CUDA Fortran Shallow Water

• FV dynamics converted to single precision
• The 2D shallow water dynamics were being implemented with direct CUDA fortran
• Using asynchronous data transfers and multi-streaming data copies host-device/device-host
• ¼-degree shallow water benchmark was complete

CPU Time
36 cores 21.5692
6 cores 75.5365

GPU Time
36 GPUs 2.1141
6 GPUs 4.6509

Speedup
36 GPUs : 36 cores 10.2x
6 GPUs : 36 cores 4.6x
6 GPUs : 6 cores 16.2x

Times for a 1-day 28-km Shallow Water Test Case

GPU Code Examples

call getCourantNumbersY(…stream(2))
call getCourantNumbersX(…stream(1))

call get_CourantNumbersY(…stream(2))
call get_CourantNumbersX(…stream(1))

call divergence_damping(…stream)
call compute_vorticity(…stream)

call fv_tp_2d(vort,...,stream)
call update_u(v, u_dev, fx, fy,...,stream)

istat = cudaMemcpy(delp, delp_dev, NX*NY)
istat = cudaMemcpy(u, u_dev, NX*(NY+1))
Latest Implementation

A “hybrid” approach using directives with direct CUDA Fortran

• Using PGI CUDA Fortran and Directives
• Maintain the CUDA Fortran implementation for the 2D advection kernel (tp_core)
• Use directives throughout the rest of FV dynamics
  • Beneficial for the many small kernels throughout FV
  • Simplify implementation and maintenance of a single codebase for CPU & GPU
Latest Implementation
A “hybrid” approach using directives with direct CUDA Fortran

- Using PGI CUDA Fortran and Directives
- Maintain the CUDA Fortran implementation for the 2D advection kernel (tp_core)
- Use directives throughout the rest of FV dynamics
  - Beneficial for the many small kernels throughout FV
  - Simplify implementation and maintenance of a single codebase for CPU & GPU
- Memory allocation and data transfers based on directives
  - Seamless CPU/GPU allocations
  - Simple
  - Declarations and Allocations can remain in existing modules
  - Can be used within !$acc regions or in direct CUDA Fortran kernels

**GPU Directives Code Example**

```fortran
module fv_grid_utils

real, allocatable, dimension(:,::) :: dx, dy
real, allocatable, dimension(:,::) :: sina_u
real, allocatable, dimension(:,::) :: sina_v

!$acc mirror (dx, dy, sina_u, sina_v)

dx(:,::) = ...
dy(:,::) = ...
sina_u(:,::) = ...
sina_v(:,::) = ...

!$acc update device(dx, dy, sina_u, sina_v)

use fv_grid_utils, only: dx, dy, sina_u, sina_v

real, allocatable, dimension(:,::) :: ut, vt

!$acc mirror (ut, vt)

!$acc region
do j=js-1,jep1+1
do i=is-1,iep1+1
  ut(i,j) = dt2*ut(i,j)*dy(i,j)*sina_u(i,j)
endo
endo
do j=js-1,jep1+1
  do i=is-1,iep1
    vt(i,j) = dt2*vt(i,j)*dx(i,j)*sina_v(i,j)
  enddo
endo
!$acc end region
```
A “hybrid” approach using directives with direct CUDA Fortran

• A simple fv_cudafor.h include file handles all argument declarations
  • device arrays
  • subroutine attributes
  • grid/block information
  • asynchronous streams

fv_cudafor.h

#include <fv_cudafor.h>

subroutine fv_tp_2d(......)
  integer, intent(IN) :: ord, isd, ied, jsd, jed, is, ie, js, je, npx, npy, ng, stream(2)
  _REAL4_ , dimension(isd:ied,jsd:jed), intent(INOUT) :: q
  _REAL4_ , dimension(isd:ied,jsd:jed), intent(INOUT) :: fx, fy
  _REAL4_ , dimension(isd:ied,jsd:jed), intent(INOUT) :: crx, cry
  _REAL4_ , dimension(isd:ied,jsd:jed), intent(INOUT) :: xfx, yfx
  _REAL8_ , dimension(isd:ied,jsd:jed), intent(INOUT) :: mfx, mfy
  _REAL8_ , dimension(isd:ied,jsd:jed), intent(INOUT) :: dxa, dya
  _REAL4_ , dimension(isd:ied,jsd:jed), intent(INOUT) :: area
  _REAL4_ , dimension(isd:ied,jsd:jed), intent(INOUT) :: q2, qf, fx2, fy2 ! Temporary Arrays

call copy_corners(q, isd, ied, jsd, jed, npx, npy, ng, YDir)
call ytp _STREAM1_ (fy2, q, cry, dya, ord...)
call intermediateQi _STREAM1_ (qi, q, area, fy2, yfx...)
call xtp _STREAM1_ (fx, qi, crx, dxa, ord...)

call copy_corners(q, isd, ied, jsd, jed, npx, npy, ng, XDir)
call xtp _STREAM2_ (fx2, q, crx, dxa, ord...)
call intermediateQj _STREAM2_ (qi, q, area, fx2, xfx...)
call ytp _STREAM2_ (fy, qj, cry, dya, ord...)

!$acc region
do j=js,je
  do i=is,ie+1
    fx(i,j) = 0.5*(fx(i,j) + fx2(i,j)) * mfx(i,j)
  enddo
endo
do j=js,je+1
  do i=is,ie
    fy(i,j) = 0.5*(fy(i,j) + fy2(i,j)) * mfy(i,j)
  enddo
endo
!$acc end region
Latest Implementation

Handling the cubed-sphere corners proved to be tricky for directives

Special cases for Cubed-Sphere corners

```fortran
subroutine fill_corners_dgrid(x, y, npx, npy, ng, VECTOR)
   real , DIMENSION(isd:ied+1, jsd:jed+1), intent(INOUT):: x
   real , DIMENSION(isd:ied+1, jsd:jed), intent(INOUT):: y
   integer, intent(IN):: npx, npy, ng
   logical, intent(IN) :: VECTOR
   ! Local
   integer :: i,j
   real :: mySign
   !$acc reflected(x,y)
   mySign = 1.0
   if (VECTOR) mySign = -1.0
   !$acc region
   !$acc do independent
   do j=1,ng
   !$acc do independent
   do i=1,ng
     if ((is  ==  1) .and. (js  ==  1)) x(1-i    ,1-j  ) = mySign*y(1-j  ,i    )  !SW
     if ((is  ==  1) .and. (je+1==npy)) x(1-i    ,npy+j) = y(1-j  ,npy-i)  !NW
     if ((ie+1==npx) .and. (js  ==  1)) x(npx-1+i,1-j  ) = y(npx+j,i    )  !SE
     if ((ie+1==npx) .and. (je+1==npy)) x(npx-1+i,npy+j) = mySign*y(npx+j,npy-i)  !NE
   enddo
   enddo
   !$acc do independent
   do j=1,ng
   !$acc do independent
   do i=1,ng
     if ((is  ==  1) .and. (js  ==  1)) y(1-i    ,1-j    ) = mySign*x(j      ,1-i  )  !SW
     if ((is  ==  1) .and. (je+1==npy)) y(1-i    ,npy-1+j) = x(j      ,npy+i)  !NW
     if ((ie+1==npx) .and. (js  ==  1)) y(npx-1+i,1-j    ) = x(npx-j  ,1-i  )  !SE
     if ((ie+1==npx) .and. (je+1==npy)) y(npx-1+i,npy-1+j) = mySign*x(npx-j  ,npy+i)  !NE
   enddo
   enddo
   !$acc end region
end subroutine fill_corners_dgrid
```

Scalar operations did not work inside directives

```fortran
! Remove the extra term at the corners:
!$acc region
if ( sw_corner ) vort(1,1) = vort(1,1) + fy(0,1)
if ( se_corner ) vort(npx,1) = vort(npx,1) - fy(npx,1)
if ( ne_corner ) vort(npx,npy) = vort(npx,npy) - fy(npx,npy)
if ( nw_corner ) vort(1,npy) = vort(1,npy) + fy(0,npy)
!$acc end region
```

Had to be rewritten with loops

```fortran
! Remove the extra term at the corners:
!$acc region
if ( sw_corner ) then
   do i=1,1 ; do j=1,1
      vort(i,j) = vort(i,j) + fy(i-1,j)
   enddo ; enddo
endif
if ( se_corner ) then
   do i=npx,npx ; do j=1,1
      vort(i,j) = vort(i,j) - fy(i,j)
   enddo ; enddo
endif
if ( ne_corner ) then
   do i=npx,npx ; do j=npy,npy
      vort(i,j) = vort(i,j) - fy(i,j)
   enddo ; enddo
endif
if ( nw_corner ) then
   do i=1,1 ; do j=npy,npy
      vort(i,j) = vort(i,j) + fy(i-1,j)
   enddo ; enddo
endif
!$acc end region
```
Performance
3D Baroclinic Wave Test Case

- Idealized dry dynamical core test case
- 26-vertical levels
- 30-day test simulation
- Clear validation test (l2-norm PS)
## Performance

### CPU/GPU Speedup

Executing on 54-sockets of our test cluster

324 Westmere Cores  -vs-  54 GPUs

<table>
<thead>
<tr>
<th></th>
<th>50km</th>
<th>28km</th>
<th>14km</th>
<th>7km</th>
<th>3.5km</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_SW</td>
<td>0.3</td>
<td>0.4</td>
<td>0.9</td>
<td>1.8</td>
<td>2.6</td>
</tr>
<tr>
<td>C_GRID_GEOP</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>C_GRID_UPDATE_UV</td>
<td>0.7</td>
<td>0.4</td>
<td>0.3</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td>D_SW</td>
<td>0.3</td>
<td>0.5</td>
<td>1.1</td>
<td>2.1</td>
<td>3.2</td>
</tr>
<tr>
<td>FV_TP_2D</td>
<td>1.1</td>
<td>1.8</td>
<td>3.4</td>
<td>5.4</td>
<td>7.2</td>
</tr>
<tr>
<td>KE_TP</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.8</td>
</tr>
<tr>
<td>D_GRID_GEOP</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>D_GRID_UPDATE_UV</td>
<td>0.6</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>TRACER_2D</td>
<td>0.7</td>
<td>0.8</td>
<td>1.4</td>
<td>2.5</td>
<td>3.4</td>
</tr>
<tr>
<td>TRACERS_TP_2D</td>
<td>0.9</td>
<td>1.3</td>
<td>2.4</td>
<td>4.3</td>
<td>5.7</td>
</tr>
<tr>
<td>REMAPPING</td>
<td>0.9</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>TOTAL</td>
<td>0.3</td>
<td>0.5</td>
<td>1.1</td>
<td>2.0</td>
<td>3.1</td>
</tr>
<tr>
<td>Ratio Compute/Data</td>
<td>1.8</td>
<td>3.7</td>
<td>3.4</td>
<td>4.0</td>
<td>3.3</td>
</tr>
</tbody>
</table>

***Segments in RED have NOT been implemented on the GPU yet***
Status - Summary

• Cubed-Sphere FV Hydrostatic Dynamics mostly implemented on GPUs
• “hybrid” use of !$acc directives and direct CUDA Fortran with PGI
• Modest performance results thus far…
• Code is maintained within a development branch of GEOS-5
  • Runs on both the CPU/GPU
  • The bulk of the GEOS-5 model is now running on GPUs
• Still need to complete the 3-routines in the hydrostatic dynamics
• A test-kernel of the Non-hydrostatic core has been implemented on the GPU
  • Results still to come…
• Will GPUs be our path or MICs?
  • Test cluster of MICs coming soon the GSFC