Directive-Based Parallelization of the NIM

Mark Govett
Tom Henderson, Jacques Middlecoff, Jim Rosinski, Paul Madden

NOAA Earth System Research Laboratory
Outline

• Brief overview of NIM model and performance using F2C-ACC
  – Further details in the three talks to follow
    • Jacques Middlecoff
      – describe efforts to improve parallel GPU performance
    • Tom Henderson
      – report on MIC parallelization of WRF physics
    • Jim Rosinski
      – Parallelization and performance of NIM and FIM

• Evaluation of the OpenACC compilers
  – Performance comparison to F2C-ACC
  – Comment on experiences
Non-Hydrostatic Icosahedral (NIM)

• Uniform, global, hexagonal-based icosahedral grid
  – Single horizontal dimension, indexed via lookup table
  – Approach adopted by MPAS model
• Designed in 2008 for GPU (and MIC)
  – Scientists, parallel programmers, computer scientists
• Target 3.5KM or finer resolution
  – Focus on running model operationally

Status

• Dynamics parallelization is complete
  – Parallel runs to 7.5KM resolution on ~10K GPUs of Titan
    • Similar tests have been done on the TACC MIC system
  – Demonstrated good performance and scaling
• Physics parallelization in progress
  – WRF physics is being used
  – Code changes to support MIC and GPU parallelization
    • Goal to get changes into WRF repository

Array Structure

dynamics: a [ k, i ]
physics: a [ i, k ]
Performance Comparisons: CPU, GPU, MIC

• Ideal comparison
  – Identical source code
  – Same CPU chip, different accelerators
  – Multiple nodes linked using same interconnect
  – Same software stack

• This comparison
  – Identical source code
    • Changes that improve performance on one architecture cannot degrade on the others
  – Different CPU chips, same generation
  – Single node performance
    • not reliant on the interconnect
    • CPU, GPU, MIC only
    • Symmetric Mode: CPU + MIC, CPU + GPU
Node Configuration (2014)

- 2 nodes, 2 sockets, 2 accelerators
Symmetric Execution on the GPU & MIC

2012-2013: Dynamics on MIC or GPU + Physics on CPU

2013 / 2014: Symmetric Mode: divide the points between CPU and GPU or MIC

2015: Fully Symmetric: CPU + GPU, CPU + MIC
# Performance: CPU, GPU, MIC

## Parallelization and Performance

- **Single source code (NIM rev 2724)**
- Directive-based parallelization
  - OpenMP: CPU, MIC
  - F2C-ACC: GPU
  - SMS: MPI
  - OpenACC: GPU

## System / Node configurations

- **NVIDIA PSG Cluster**
  - **IB20**: Intel IvyBridge, 20 cores, 3.0 GHz (Intel E5-2690 v2)
  - **GPU**: Kepler K40, 2880 cores, 745 MHz, 12GB memory
- **Intel Endeavor Cluster**
  - **IB24**: Intel IvyBridge, 24 cores, 2.70 GHz (Intel E5-2697v2)
  - **MIC**: KNC 7120, 61 cores, 1.238 GHz, 16 GB memory
NIM Dynamics: Single Node Performance

**System / Node configurations**
- NVIDIA PSG Cluster
  - **IB20**: Intel IvyBridge, 20 cores, 3.0 GHz (Intel E5-2690 v2)
  - **GPU**: Kepler K40, 2880 cores, 745 MHz, 12GB memory
- Intel Endeavor Cluster
  - **IB24**: Intel IvyBridge, 24 cores, 2.70 GHz (Intel E5-2697 v2)
  - **MIC**: KNC 7120, 61 cores, 1.238 GHz, 16 GB memory

**Parallelization and Performance**
- Single source code (NIM rev 2724)
- Directive-based parallelization
  - OpenMP: CPU, MIC
  - F2C-ACC: GPU
  - SMS: MPI
  - OpenACC: GPU

**120 KM Resolution (NIM – G6)**
- 40,968 Columns, 96 Vertical Levels
- 100 time steps

**Numeric values represent node run-times for each configuration**

<table>
<thead>
<tr>
<th>Node Type</th>
<th>CPU runtime</th>
<th>MIC runtime</th>
<th>GPU runtime using F2C-ACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>IB20 only</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IB24 only</td>
<td>74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIC only</td>
<td></td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>GPU only</td>
<td></td>
<td></td>
<td>58</td>
</tr>
<tr>
<td>IB24 + MIC</td>
<td></td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>IB20 + GPU</td>
<td></td>
<td></td>
<td>46</td>
</tr>
<tr>
<td>IB20 + 2 GPU</td>
<td></td>
<td></td>
<td>33</td>
</tr>
</tbody>
</table>

**Run-time (sec)**

- IB20 only
- IB24 only
- MIC only
- GPU only
- IB24 + MIC
- IB20 + GPU
- IB20 + 2 GPU

**Nodes**
- IB20
- IB24
- MIC
- GPU

**Run-time**
- CPU run-time
- MIC run-time
- GPU run-time

** OMAP 1541 CPU: Intel JOHN, 2880 cores, 745 MHz, 16GB memory**

**OpenMP**

**F2C-ACC**

**GPU**

**MPI**

**OpenACC**

**http://www.esrl.noaa.gov/gsd/ab/ac/NIM-Performance.html**
F2C-ACC Compiler

• Developed in 2008 before commercial compilers were available
• Limited Capabilities, Scope, Support
  – Partial support for Fortran 90
  – Mostly line for line conversion, limited analysis
  – Supports global, local & shared GPU memory
• Performance optimizations added (NCAR workshop, 2012)
  • Increase parallelism (block, chunk)
  • Memory management (variable promotion, demotion)
• Support NIM & FIM, portions of WRF Physics
  – New capabilities added as needed
• Development Plans
  – No attempt to conform to openACC standard
  – No new development since 2012 (only bug fixes)
  – Use OpenACC compilers when ready
• Evaluations of CAPS, Cray, PGI in 2011, 2013, & 2014
  – Capability: can they support FIM, NIM, WRF?
  – Performance: are they within ~10-20% of F2C-ACC?
Dynamics Code + F2C-ACC Directives

- Directives appear as Fortran comments
  - ACC$REGION defines an accelerator region
  - ACC$DO identifies parallelism
  - ACC$THREAD restricts parallelism to a single thread

```fortran
!ACC$REGION(<96>,<10242>) BEGIN
!ACC$DO PARALLEL(1)
do ipn=ips,ipe   ! Loop over horizontal
!ACC$DO VECTOR(1)
do k=1,nz-1     ! Loop over vertical levels
    bedgvar(k,ipn,1) = ca4k(k)* u(k,ipn)+ca4p(k)* u(k+1,ipn)
    bedgvar(k,ipn,2) = ca4k(k)* v(k,ipn)+ca4p(k)* v(k+1,ipn)
end do
!ACC$THREAD(nz-1) BEGIN
    bedgvar(nz,ipn,1)= ca4k(nz)* u(nz,ipn)+ca4p(nz)* u(nz,ipn)
    bedgvar(nz,ipn,2)= bedgvar(nz,ipn,2)=ca4k(nz)* v(nz,ipn) &
                     +ca4p(nz)* v(nz,ipn)
!ACC$THREAD END
end do
!ACC$REGION END
```
F2C-ACC Generated Code

- Mostly a direct conversion to C and CUDA
- Readable, debuggable
- Loops directly mapped to threads and blocks

```c
__global__ void vdmintv_Kernel1( float *ca4k,...)

//!ACC$DO PARALLEL(1)
ipn = blockIdx.x+1;

//!ACC$DO VECTOR(1)
k = threadIdx.x+1;

bedgvar[FTNREF3D(k,ipn,1,nz-0+1,ime-ims+1,0,ims,1)] = ca4k[FTNREF1D(k,1)] * u[FTNREF2D(k,ipn,nz,1,ims)] + ca4p[FTNREF1D(k,1)] * u[FTNREF2D(k+1,ipn,nz,1,ims)];

bedgvar[FTNREF3D(k,ipn,2,nz-0+1,ime-ims+1,0,ims,1)] = ca4k[FTNREF1D(k,1)] * v[FTNREF2D(k,ipn,nz,1,ims)] + ca4p[FTNREF1D(k,1)] * v[FTNREF2D(k+1,ipn,nz,1,ims)];

// !ACC$THREAD(nz-1) BEGIN
if (threadIdx.x == nz-1) {
    bedgvar[FTNREF3D(nz,ipn,1,nz-0+1,ime-ims+1,0,ims,1)] = ca4k[FTNREF1D(nz,1)] * u[FTNREF2D(nz,ipn,nz,1,ims)] + ca4p[FTNREF1D(nz,1)] * u[FTNREF2D(nz,ipn,nz,1,ims)];
}

// !ACC$THREAD END
```
Example with F2C-ACC & OpenACC

!ACC$REGION(<96>,<10242>) BEGIN
!$acc parallel num_gangs(10242) vector_length(96)

!ACC$DO PARALLEL(1)
!$acc loop gang
do ipn=ips,ipe
!ACC$DO VECTOR(1)
!$acc loop vector
do k=1,nz-1
   bedgvar(k,ipn,1) = ca4k(k)* u(k,ipn)+ca4p(k)* u(k+1,ipn)
   bedgvar(k,ipn,2) = ca4k(k)* v(k,ipn)+ca4p(k)* v(k+1,ipn)
end do
!ACC$THREAD(nz-1) BEGIN
   bedgvar(nz,ipn,1)= ca4k(nz)* u(nz,ipn)+ca4p(nz)* u(nz,ipn)
   bedgvar(nz,ipn,2)= bedgvar(nz,ipn,2)=ca4k(nz)* v(nz,ipn)
!ACC$THREAD END
enddo
!$acc end parallel
!$acc end data

!ACC$REGION END
OpenACC Compiler Evaluation (2014)

• Results shared with NVIDIA, PGI, Cray
  – Correctness: improving, bugs reported to vendors
  – Performance: Significantly slower than F2C-ACC

**NIM Dynamics**
Runtimes in seconds, 100 time steps, single precision, single GPU

<table>
<thead>
<tr>
<th>Routine</th>
<th>(% of CPU)</th>
<th>F2C-ACC</th>
<th>PGI - OpenACC</th>
<th>Cray - OpenACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vdmints¹</td>
<td>(38%)</td>
<td>7.10</td>
<td>18.40 (2.6)</td>
<td>14.61 (2.1)</td>
</tr>
<tr>
<td>Vdmintv</td>
<td>(15%)</td>
<td>3.59</td>
<td>7.19 (2.0)</td>
<td>5.50 (1.5)</td>
</tr>
<tr>
<td>Flux</td>
<td>(9%)</td>
<td>1.06</td>
<td>1.94 (1.8)</td>
<td>1.52 (1.4)</td>
</tr>
<tr>
<td>Diag</td>
<td>(8%)</td>
<td>0.81</td>
<td>2.00 (2.5)</td>
<td>0.89 (1.1)</td>
</tr>
<tr>
<td>Total</td>
<td>(100%)</td>
<td>16.12</td>
<td>35.53 (2.2)</td>
<td>28.52 (1.8)</td>
</tr>
</tbody>
</table>

¹ represents runtimes for 3 variants of the same routine

**WRF Physics**
Runtimes in micro-seconds, 1 kernel invocation, double precision

<table>
<thead>
<tr>
<th>Routine</th>
<th>(% total)</th>
<th>F2C-ACC</th>
<th>PGI - OpenACC</th>
<th>Cray - OpenACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSM3</td>
<td>(19%)</td>
<td>21.4</td>
<td>599.6 (28.0)</td>
<td></td>
</tr>
<tr>
<td>PBL</td>
<td>(1%)</td>
<td>1.5</td>
<td>3.4 (2.3)</td>
<td></td>
</tr>
</tbody>
</table>
WRF Physics: WSM3

- OpenACC parallelism only worked for 1 dimension

```fortran
subroutine phys_mps_wsm3(its,ite, ...)

integer,intent(IN) :: its,ite,kts,kte
real :: qci( its:ite, kts:kte)
real :: qts( its:ite, kts:kte)

!ACC$REGION(<128:chunk,<((ite-its+1)/128+1),<kte-kts+1> ) BEGIN
!ACC$DO PARALLEL(2)
!!$acc loop gang !not supported in OpenACC
do k = kts,kte !Loop over vertical dimension

!ACC$DO PARALLEL(1,its:ite)
!!$acc loop gang vector
do i = its,ite ! Loop over horizontal
    qci(i,k) = q2(i,k,1)
    qts(i,k) = q2(i,k,2)
end do
end do
! many more calculations in 6 REGIONS
```
Performance Analysis: Dynamics

• No attempt to compare generated code
  – Difficult to read
    • Cray (ptx), PGI (cuda)
  – Vendors are engaged and doing this

• Backed out F2C directive-based optimizations
  – Parallelism (chunking, blocking)
  – Memory Management (promotion, demotion)

\[
\text{ACC$REGION(<96>,<10242>,<\text{varA: none, local, demote(1)>})
\]
  – float VarA(96) = float varA
  – thread local array

Runtimes shown WITHOUT using demotion, promotion, blocking and chunking versus F2C-opt

<table>
<thead>
<tr>
<th>Routine</th>
<th>F2C - opt</th>
<th>Demotion</th>
<th>Blocking</th>
<th>Chunk</th>
<th>Promotion</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdmints</td>
<td>7.12</td>
<td>429.88</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vdmintv</td>
<td>3.59</td>
<td>180.95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flux</td>
<td>1.06</td>
<td>1.35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>diag</td>
<td>0.81</td>
<td>1.29</td>
<td>1.25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Memory Management: Dynamics

- User defined shared memory
  - GPU cache is used by default (dependent on compiler)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Vdmints</th>
<th>Vdmintv</th>
<th>Flux</th>
<th>Diag</th>
</tr>
</thead>
<tbody>
<tr>
<td>global</td>
<td>7.12</td>
<td>3.59</td>
<td>1.06</td>
<td>0.81</td>
</tr>
<tr>
<td>shared</td>
<td>8.23 (1.2)</td>
<td>3.75 (1.1)</td>
<td>1.35 (1.3)</td>
<td>1.69 (2.1)</td>
</tr>
</tbody>
</table>

Comparison of runtimes, for F2C routines using shared memory explicitly versus the default

- Register usage

<table>
<thead>
<tr>
<th>Routine / # registers</th>
<th>F2C-ACC (sec)</th>
<th>Cray</th>
<th>PGI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vdmints</td>
<td>58 (12.4)</td>
<td>128 (28.4)</td>
<td>63 (30.6)</td>
</tr>
<tr>
<td>vdmintv</td>
<td>81 (8.9)</td>
<td>128 (13.7)</td>
<td>63 (17.9)</td>
</tr>
<tr>
<td>flux</td>
<td>38 (2.6)</td>
<td>70 (3.8)</td>
<td>63 (4.7)</td>
</tr>
<tr>
<td>diag</td>
<td>30 (1.6)</td>
<td>45 (1.8)</td>
<td>63 (3.9)</td>
</tr>
</tbody>
</table>

- Some correlation between high usage (Cray) and runtime
- PGI use of 63 registers for each routine needs to be investigated
Concerns about OpenACC

- To get good performance and sometimes correct results, we were forced to use:

```c
#ifdef PGIOPENACC
    - PGI openACC directive
#endif
#else
    - CRAY openACC directive
#endif
```

NCSA BlueWaters WebSite:  https://bluewaters.ncsa.illinois.edu/openacc

NCSA recommends using only one programming environment for your code development and testing. At this time, the OpenACC standard is not yet mature enough to ensure complete portability between the Cray and PGI compilers. The Cray compiler tends to adhere very strictly to the standard while the PGI compiler allows for more flexibility in mixing directives that are not explicitly stated to work together.
OpenACC Directives in NIM

• Data Management: keeping resident on GPU
  – PGI: !$acc declare create(var1, var2)
  – Cray: !$acc enter data pcopyin (var1, var2)

• Kernel parallelism
  – PGI: !$acc parallel vector_length(96)
  – Cray: !$acc parallel num_workers(3) vector_length(32)

For now, using:
– Cray: !$acc parallel vector_length (96)
  • Gives a warning message and uses 64 threads
Summary

• Briefly described NIM performance
  – Single source code
    • no CPP directives (except for Cray, PGI openACC)
  – Symmetric mode improves performance, cost benefit, power consumption
  – Further details in talks tomorrow
• OpenACC compiler performance improvements needed
  – Not clear where problems are
    • Memory management: local, shared, demotion
    • Increasing parallelism
  – Have not looked at generated code
    • PGI (CUDA), Cray (PTX)
  – Results shared with vendors in May 2014
    • Vendors are working on improvements
• Concerns about OpenACC standard
  – Deprecate redundancies or similarities in directives
  – More specificity may be needed until compilers mature