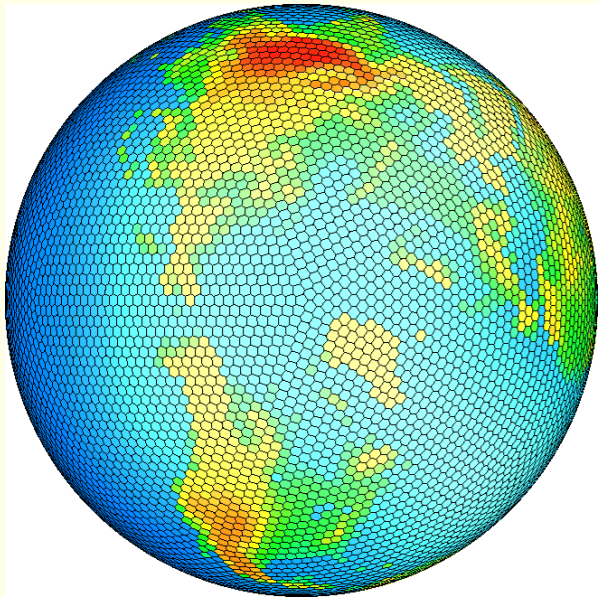
Earth System Research Laboratory
Global Systems Division

Experience Applying Fortran GPU Compilers to Numerical Weather Prediction

Tom Henderson
NOAA Global Systems Division
Thomas.B.Henderson@noaa.gov

Mark Govett, Jacques Middlecoff
Paul Madden, James Rosinski,
Craig Tierney



Outline

- Motivation for GPU investigation
- The Non-hydrostatic Icosahedral Model (NIM)
- GPU software design issues
- Commercial directive-based Fortran GPU compilers
- Step-wise approach
- Initial performance comparisons
- Conclusions and future directions

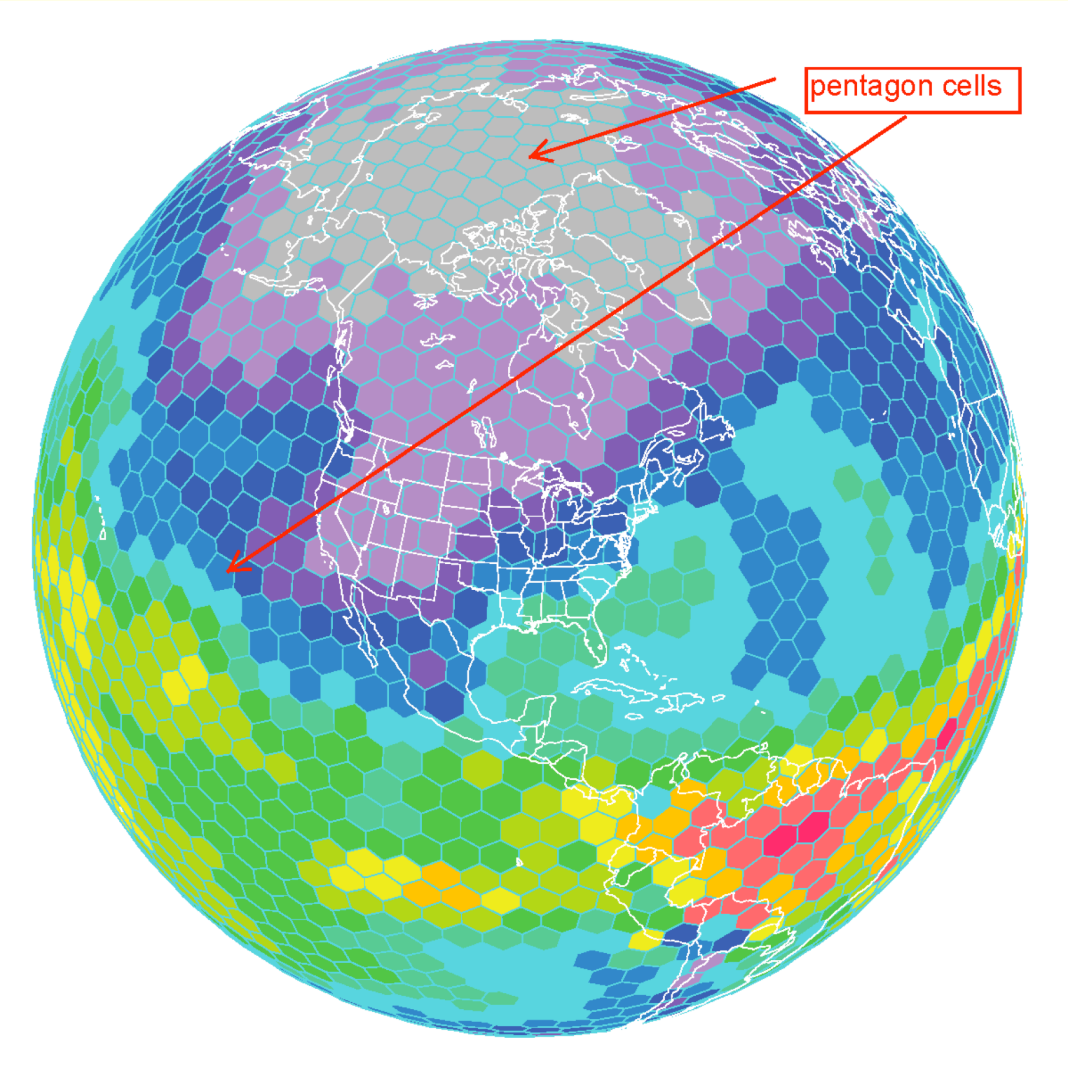
Motivation for NOAA GPU Investigation

- Run global NWP and climate models at higher resolution with more sophisticated physical parameterizations to improve forecast skill
  - Research: shop for FLOPs
  - Operations: technology must be mature
- Michalakes early GPU work
- Continue to maintain single source code for all desired execution modes
  - Single and multiple CPU
  - Single and multiple GPU
  - Prefer a directive-based Fortran approach

NIM NWP Dynamical Core

- NIM = “Non-Hydrostatic Icosahedral Model”
  - NWP dynamical core prototype
  - Global “cloud-permitting” resolutions < 3km (42 million columns)
- 32-bit floating-point computations
- Computations structured as simple vector ops with indirect addressing and inner vertical loop
  - “GPU-friendly”, also good for CPU

Icosahedral Grid

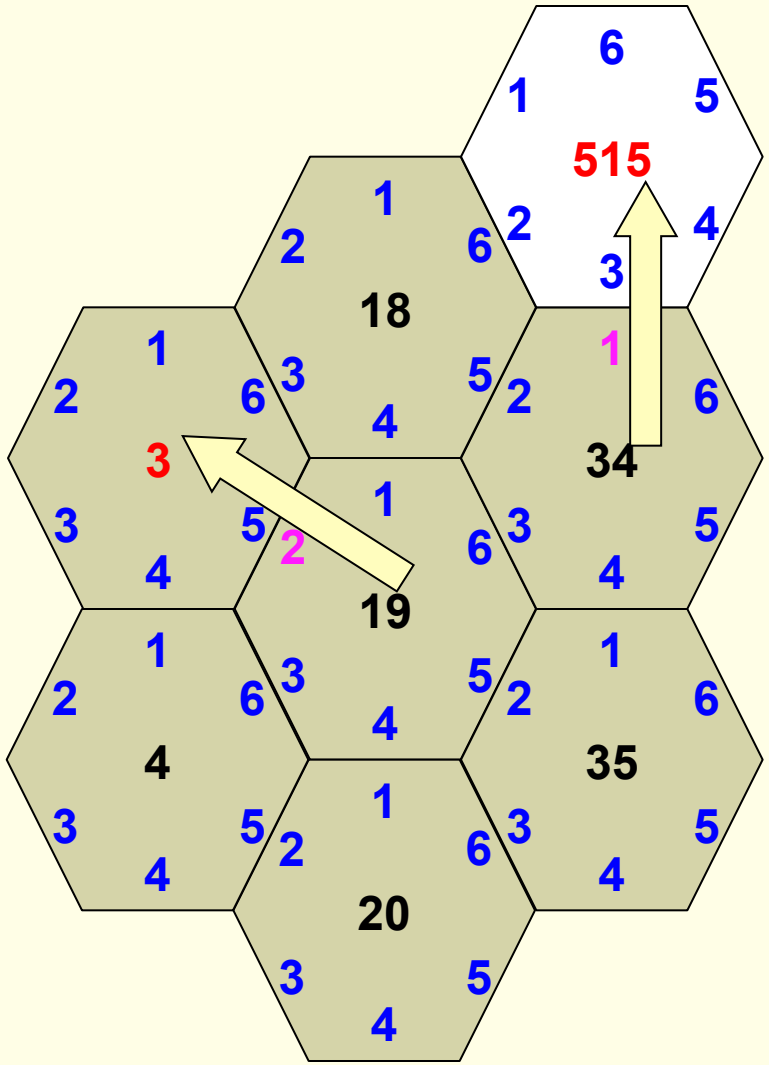


450km
2562 columns

Always 12 pentagons

Good geometric properties compared to traditional latitude/longitude grid

NIM/FIM Indirect Addressing (MacDonald, Middlecoff)



- Single horizontal index
- Store number of sides (5 or 6) in “nprox” array
  - nprox(34) = 6
- Store neighbor indices in “prox” array
  - prox(1,34) = 515
  - prox(2,19) = 3
- Place directly-addressed vertical dimension fastest-varying for speed
- Very compact code
- Indirect addressing costs <1%

Simple Loop With Indirect Addressing

- Compute sum of all horizontal neighbors
  - nip = number of columns
  - nvl = number of vertical levels

```
xnsum = 0.0
do ipn=1,nip                ! Horizontal loop
  do isn=1,nprox(ipn)       ! Loop over edges (sides, 5 or 6)
    ipp = prox(isn,ipn)     ! Index of neighbor across side "isn"
    do k=1,nvl              ! Vertical loop
      xnsum(k,ipn) = xnsum(k,ipn) + x(k,ipp)
    enddo
  enddo
enddo
```

GPU Software Design Issues

- CPU controls high level program flow
  - I/O, message passing, coarse-grained parallelism
    - MPI parallelism via the Scalable Modeling System (SMS)
      - Directive-based
- GPU performs all computations
  - Fine-grained parallelism
  - Implemented by a GPU compiler
- Model data is resident on the GPU
  - Invert traditional “GPU-as-accelerator” model
    - Initial data read by the CPU and passed to the GPU
    - Data passed back to the CPU only for output & message-passing
    - Minimizes overhead of data movement between CPU & GPU

GPU Software Design Issues

- Massive fine-grained parallelism
  - GPU “global memory” is slow
    - Need lots of threads to hide memory latency
    - Key trade-off: number of threads vs. thread resources
- Coalesced loads
  - Multiple memory accesses in one load
  - Adjacent threads must access adjacent data
    - Use unit stride access in loops that will be threaded

GPU Software Design Issues

- Reduce branching
  - Code should vectorize well using CPU compilers
- Redesign algorithms if needed
- Optimize inter-CPU (MPI) communication
  - Computation time is a much smaller fraction of total run time when multiple GPUs are used
  - Overlap communication with computation
  - Use redundant computation to eliminate communication

GPU Fortran Compilers

- Commercial directive-based compilers
  - CAPS HMPP 2.3.5
    - Generates CUDA-C and OpenCL
    - Supports NVIDIA and AMD GPUs
  - Portland Group PGI Accelerator 11.7
    - Supports NVIDIA GPUs
    - Previously used to accelerate WRF physics packages
- F2C-ACC (Govett, 2008) directive-based compiler
  - “Application-specific” Fortran->CUDA-C compiler for performance evaluation
- Other directive-based compilers
  - Cray (beta)

Current GPU Compiler Limitations

- Limited support for Fortran language features such as modules, derived types
- Support not yet strong for automatic inlining, __device__ routines
- Both PGI and HMPP prefer “tightly nested outer loops” (not a limitation for F2C-ACC)

```
! This is OK
do ipn=1,nip
  do k=1,nvl
    <statements>
  enddo
enddo
```

```
! This is NOT OK
do ipn=1,nip
  <statements>
  do k=1,nvl
    <statements>
  enddo
enddo
```

Directive Comparison: Loops

HMPP

```
!$hmppcg parallel
do ipn=1,nip
!$hmppcg parallel
 do k=1,nvl
  do isn=1,nprox(ipn)
    xnsum(k,ipn) = xnsum(k,ipn) + x(k,ipp)
  enddo
 enddo
enddo
```

PGI

```
!$acc do parallel
do ipn=1,nip
!$acc do vector
 do k=1,nvl
  do isn=1,nprox(ipn)
    xnsum(k,ipn) = xnsum(k,ipn) + x(k,ipp)
  enddo
 enddo
enddo
```

Directive Comparison: Array Declarations

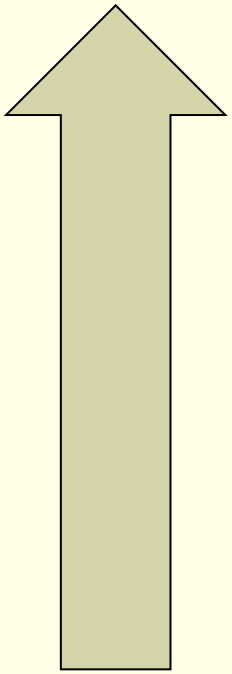
Original Code

```
real :: u(nvl,nip)
…
call diag(u, …)
call vd(u, …)
call diag(u, …)
…
subroutine vd(fin, …)
…
subroutine diag(u, …)
…
```

Directive Comparison: Array Declarations

HMPP

```
real :: u(nvl,nip)
!$hmpp map, args[vd::fin;diag1::u;diag2::u]
…
!$hmpp diag1 callsite
call diag(u, …)
!$hmpp vd callsite
call vd(u, …)
!$hmpp diag2 callsite
call diag(u, …)
…
!$hmpp vd codelet
subroutine vd(fin, …)
…
!$hmpp diag1 codelet
!$hmpp diag2 codelet
subroutine diag(u, …)
…
```



Directive Comparison: Array Declarations

PGI

```
!$acc mirror (u)
real :: u(nvl,nip)

! Must make interfaces explicit via interface
! block or use association
include "interfaces.h"
…
call diag(u, …)
call vd(u, …)
call diag(u, …)
…
subroutine vd(fin, …)
!$acc reflected (fin, …)
…
subroutine diag(u, …)
!$acc reflected (u, …)
…
```

Directive Comparison: Explicit CPU-GPU Data Transfers

HMPP

```
!$hmpp diag1 advancedLoad, args[u]
…
!$hmpp diag2 delegatedStore, args[u]
```

PGI

```
!$acc update device(u)
…
!$acc update host(u)
```

Step-Wise Approach to GPU Parallelization

- GPU tools and debuggers are still relatively primitive
  - Bugs can be difficult to diagnose
- Create test cases and establish tolerances
  - Match tolerances observed from CPU compiler optimization changes
- Make small changes and test after each
  - Much easier to find and fix errors

Step-Wise Approach to GPU Parallelization

- Compare HMPP and PGI output and performance with F2C-ACC compiler
  - Use F2C-ACC to prove existence of bugs in commercial compilers
  - Use F2C-ACC to prove that performance of commercial compilers can be improved
- Both HMPP and F2C-ACC generate “readable” CUDA code
  - Re-use function and variable names from original Fortran code
  - Allows straightforward use of CUDA profiler
  - Eases detection and analysis of compiler correctness and performance bugs

Initial Performance Results

- Optimize for both CPU and GPU
  - Some code divergence
  - Always use fastest code
- CPU = Intel Nehalem (2.8GHz) or Intel Westmere (2.66GHz)
- GPU = NVIDIA GTX280 “Tesla” or C2050 “Fermi”
- Work in-progress…

Initial Performance Results

- Small “G4-L96” test case
  - 2562 columns, 96 levels, 50 time steps
    - Fraction of time spent in init/input is unrealistically large
- Large “G5-L96” test case
  - 10242 columns, 96 levels, 1000 time steps
  - Newer version of NIM code
- Update to newer NIM code turned out to be non-trivial!
- Many GPU optimizations remain untried

Run Times for Single GPU vs. Single Nehalem Core, “G4-L96”

NIM routine	Nehalem CPU Time (sec)	F2C-ACC Tesla GPU Time (sec)	HMPP Tesla GPU Time (sec)	PGI Tesla GPU Time (sec)
Total	106.6	10.8	10.3	--
vdmints	50.6	2.5	2.3	4.6
vdmintv	23.3	0.93	0.99	0.93
flux	10.4	1.15	1.05	0.43
vdn	4.6	0.58	0.73	--
diag	4.0	0.093	0.085	0.12
force	3.4	0.11	0.19	0.09
trisol	2.0	1.9	1.4	--

Estimated GFLOPS for GPU and Single Nehalem Core "G4-L96"

NIM routine	Nehalem 1-core CPU GFLOPS	F2C-ACC CUDA-C Tesla GPU GFLOPS	HMPP Tesla GPU GFLOPS	Computational Intensity
Total	3.2	31	32	1.68
vdmints	3.8	77	85	1.96
vdmintv	3.9	99	95	1.85
flux	2.3	21	23	1.11
vdn	1.0	9	6	0.89
diag	1.3	57	62	1.12
force	1.9	61	35	1.41
trisol	2.3	2.2	3	1.10

- Used PAPI performance counters on CPU (GPTL)
  - *Estimated* ~29% of peak (11.2 GFLOPS) on CPU

Fermi GPU vs. Single/Multiple Westmere CPU cores, “G5-L96”

NIM routine	Westmere CPU 1-core Time (sec)	Westmere CPU 6-core Time (sec)	F2C-ACC Fermi GPU Time (sec)	Fermi Speedup vs. 6-core CPU (1 socket ea.)
Total	8654	2068	449	4.6
vdmints	4559	1062	196	5.4
vdmintv	2119	446	91	4.9
flux	964	175	26	6.7
vdn	131	86	18	4.8
diag	389	74	42	1.8
force	80	33	7	4.7
trisol	119	38	31	1.2

“G5-96” with PGI and HMPP

- HMPP:
  - Each kernel passes correctness tests in isolation
  - Run times of individual kernels very close to F2C-ACC
  - Unresolved error in “map”/data transfers
- PGI:
  - Started with PGI 11.7 six days ago
  - Entire model runs but does not pass correctness tests
  - Run times of most expensive kernels very close to F2C-ACC
  - Data transfers appear to be correct
  - Likely error in one (or more) kernel(s)

Ongoing Work With WRF Physics

- Legacy codes not designed with GPU in mind
- Much more difficult than NIM
- Initial candidate: YSU PBL scheme
- WRF physics (i,k,j) ordering good for coalesced loads (Michalakes, others)
  - Must transpose from NIM (k,ipn)
  - Transpose costs appear small
  - Memory may be an issue

Early Work With Multi-GPU Runs

- F2C-ACC + SMS directives
  - Correct results on different numbers of GPUs
  - Poor scaling because compute has sped up but communication has not
  - Working on communication optimizations
- Demonstrates that single source code can be used for single/multiple CPU/GPU runs
- Should be possible to mix HMPP/PGI directives with SMS too

Conclusions

- Some grounds for optimism
  - Fermi is ~4-5x faster than 6-core Westmere
  - Once compilers mature, expect level of effort similar to OpenMP for “GPU-friendly” codes like NIM
- HMPP strengths: more flexible low-level loop transformation directives, user-readable CUDA-C
- PGI strengths: simpler directives for making data persist in GPU memory
- This is still very much a work-in-progress

Future Directions

- Continue to improve GPU performance
  - Tuning options via commercial compilers
  - Test AMD GPU/APUs (HMPP->OpenCL)
- Address GPU scaling issues
- Cray GPU compiler
  - Working with beta releases
- Intel MIC
- OpenMP extensions?
- OpenHMPP?

Thank You