

Successes and Challenges Using GPUs for Weather and Climate Models

Mark Govett

Tom Henderson, Jacques Middlecoff,
Jim Rosinski

NOAA Earth System Research Laboratory



GPU Programming Approaches

- Language Approach
 - CUDA, OpenCL, CUDA Fortran, etc.
 - User control over coding and optimizations
 - Vendor specific optimizations
 - May not be portable across architectures
 - Requires that separate versions be maintained
 - In practice this rarely works – too costly, difficult
- Directive-based Approach
 - Appear as comments in the source
 - !ACC\$DO VECTOR (1)
 - Compilers can analyze and (hopefully) generate efficient code
 - Dependent on maturity



Directive-Based Approach

- Maintain a single source code (Fortran)
 - Same code for CPU, GPU, serial, and parallel
 - Models used for research and operations
 - Community models continue to be developed
 - Scientists are the “keepers” of the code
 - Owners of the science
 - Expected to modify & maintain their code
 - Software engineers improve performance, find parallel bugs, port codes, etc
- User must insert directives into their code



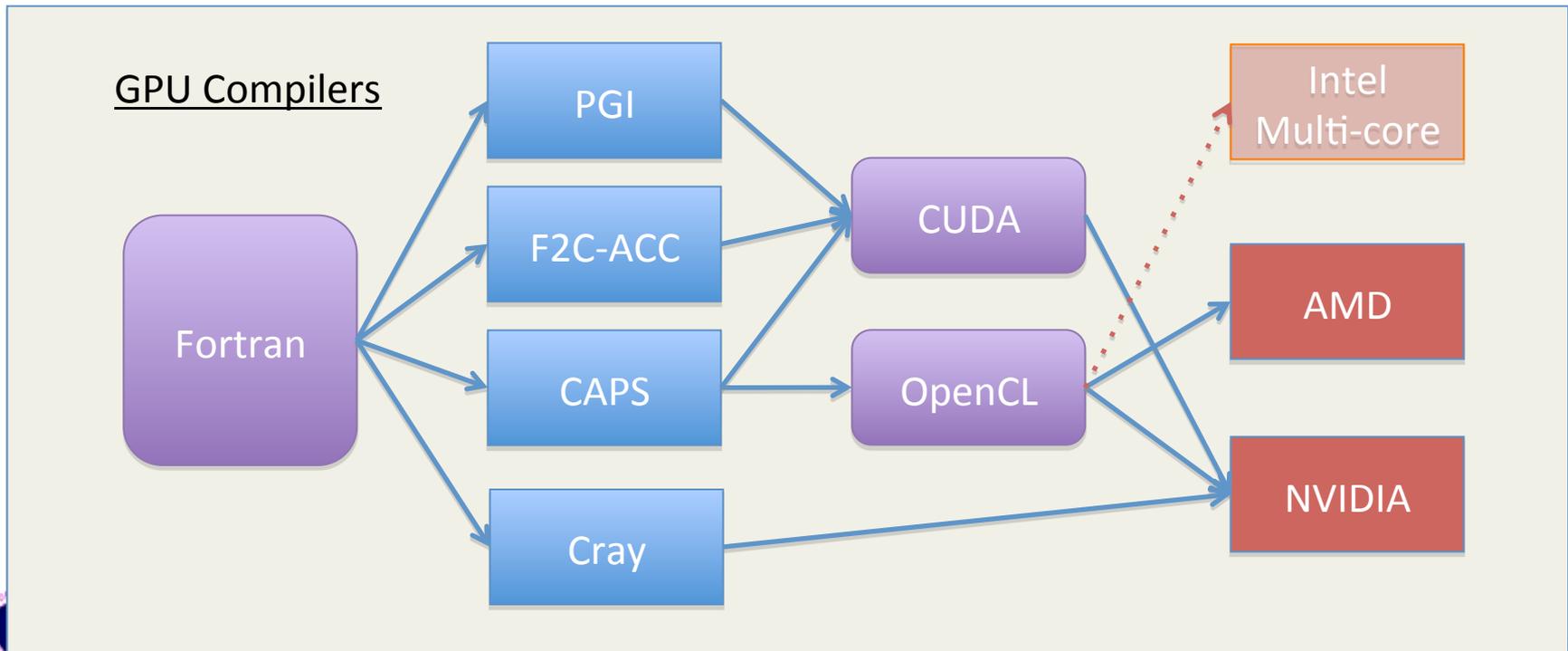
Goal to Maintain a Single Source

- Significant challenge
 - GPU, CPU architectures are quite different
 - Memory hierarchy
 - Loop level versus block level parallelism
 - Inter-GPU communications
- Heavy reliance on compilers and other tools
 - Code analysis
 - parallelization diagnostics
 - Loop reordering
 - Data management (register, constant, shared, global, etc)
- To what extent will algorithm changes be needed to extract independent, fine grain parallelism
 - Will the solution be performance portable



Directive-Based Fortran GPU Compilers and Portability

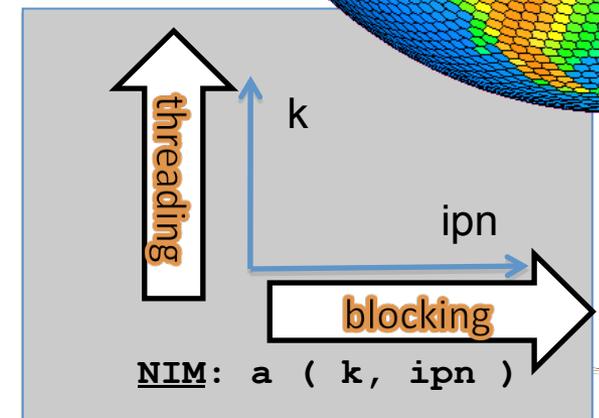
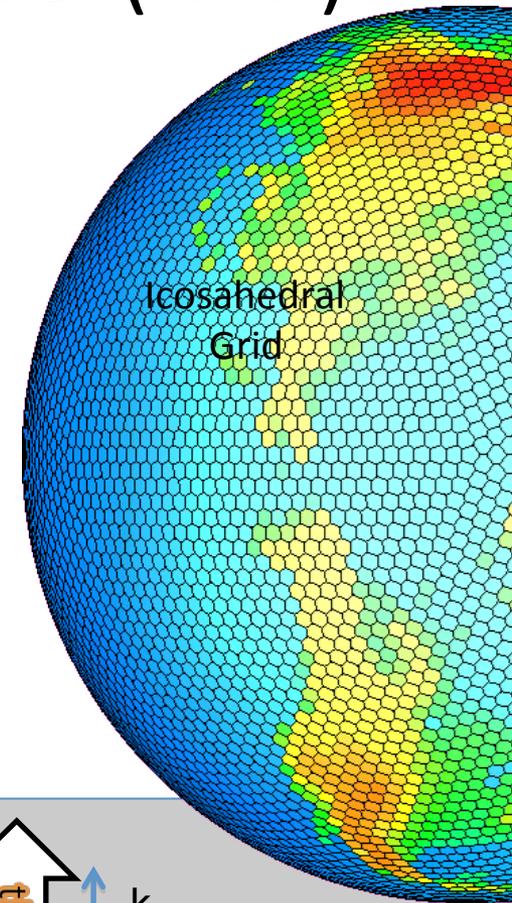
- No industry standard for directives (yet)
 - PGI: Accel
 - F2C-ACC: OpenSource
 - CAPS: HMPP
 - Cray: OMP “like”



Non-hydrostatic Icosahedral Model (NIM)

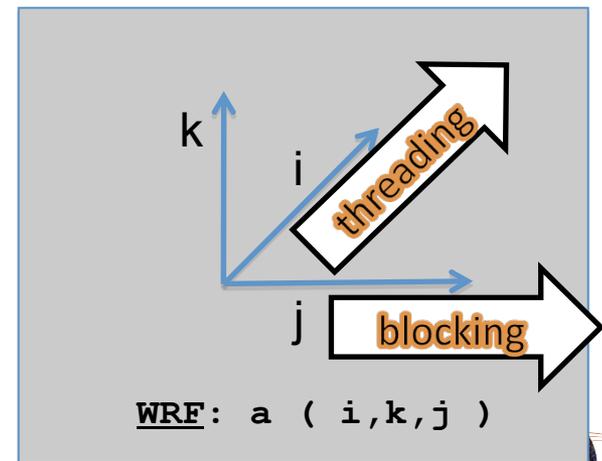
(Lee, MacDonald)

- Global Weather Forecast Model
- Developed at ESRL
- Uniform, hexagonal-based, icosahedral grid
- Novel indirect addressing scheme permits concise, efficient code
- Designed and optimized for CPUs and GPUs
 - Very good performance on both
 - Luxury of changing code for optimal performance
- Dynamics is running entirely on GPUs
 - 25x speedup (5x, socket to socket)
 - Horizontal data dependencies
 - 2D arrays (vertical, horizontal)
 - GPU threading across the vertical
 - 32, 96 points - - - > 192 points
 - Physics integration in progress



WRF Physics

- Community Model used worldwide for more than a decade
 - Significant number of collaborators, contributors
- Used in WRF-ARW, WRF-NMM, WRF-RR, WRF-CHEM, HWRF, etc.
- Traditional cartesian grid
 - 3D arrays (horizontal, vertical, horizontal) == > array3D(i, k, j)
- Designed for CPU architectures
- **Limited ability to change the code**
 - **Must continue to be performance portable**
- GPU parallelization
 - In progress – select routines
 - Dependencies in vertical
 - GPU: threading in horizontal dimensions



F2C-ACC GPU Compiler

- Developed to speed parallelization of NIM
 - Commercial compilers were not available in 2008
 - 25x speedup over 1 Intel Westmere core, ~ 5x socket to socket
- Translates Fortran to C or CUDA
 - Many (but not all) language features supported
 - Generates readable, debuggable code with original comments retained
 - No code optimizations are being done
- Continues to be developed
 - Used at NOAA and by research groups worldwide
 - Improving code analysis & diagnostic messages
 - Variable usage within and between GPU kernels
 - GPU data management



F2C-ACC Directives

- Define GPU Kernels

```
ACC$REGION (< THDs > , < BLKs > ) BEGIN  
ACC$REGION END
```

- Define loop level parallelism

```
ACC$DO VECTOR( dim [ range] )      - thread  
ACC$DO PARALLEL (dim [ range] )    - blocks
```

- Data movement

```
ACC$DATA (<var: intent, type>] ) BEGIN
```

```
intent: in, out
```

```
type: shared, constant, global memory
```

```
ACC$REGION( < THD > , < BLK > , <var: intent, scope > )
```

```
intent: in, out, inout, none
```

```
scope: global, local, extern, none
```



F2C-ACC Directives

- Restricting Thread Ops

ACC\$THREAD (dim, [range])

ACC\$DO PARALLEL \ VECTOR (dim, [range])

- Thread Synchronization

ACC\$SYNC

- Translation Limitations

ACC\$INSERT, ACC\$INSERTC, ACC\$REMOVE

Available to the community

Users guide

Limited support

<http://www.esrl.noaa.gov/gsd/ad/ac/Accelerators.html>



Successes

- Parallelization of NIM
 - ~5x socket-to-socket speedup of NIM dynamics
- Development of F2C-ACC
 - **Useful for comparisons to commercial compilers**
 - Establish performance benchmarks
 - Ease of use: readability of generated code
 - Directives that support our weather, climate codes
 - Validate correctness of results
 - Feedback to vendors
 - Communicate needs in the weather and climate community
 - Helped improved Fortran GPU compilers



Challenges: Validating Results

- Results depend on:
 - Computer architecture – different machines give different results
 - Compiler options and language
 - How intrinsics are calculated, constants defined? (in SP or DP)
- To validate results, run the model or routine & compare results after XX number of time steps
 - Bitwise exact result (all digits)
 - For Fortran, C on the same machine - F2C-ACC validates, not PGI
 - Fortran, CUDA on the same machine - compiler options required
 - “Nearly” bitwise exact (5, 6 digits)
 - C and CUDA on different architectures?
 - Fortran and CUDA on CPU and GPU?
 - Diverging results (2,3 digits)
 - Numerics could be correct but scientists need to approve
 - Physics may be acceptable
 - Probably not acceptable for dynamics
- How do you determine acceptable results?



Challenges: Performance Portability

- CPU: PBL declares 3D arrays, but uses 2D slices for inner loop calculations
 - Array3D (i, k, j) == > Array2D(i,k)
 - Done to improve cache utilization
- GPU: 2D arrays in YSU had to be promoted to 3D where,
 - Data dependencies are in “k”
 - Threading over the “i” dimension
 - Blocking over the “j” dimension
 - Needed for correctness and good performance on the GPU
 - Resulted in a 50 percent performance penalty on the CPU

Jim Rosinski’s talk will discuss a “chunking” solution for NIM



Application Requirements

- Restriction on the number of formal arguments
 - CUDA has a 256 bytes limit (64 – 4 byte words)

`NVCC: Formal parameter space overflowed
in function ...`

- Typical for physics routines
 - WRF - PBL has ~ 50 subroutine arguments
 - 25-30 local arrays are also passed to GPU via argument list
- Solution via F2C-ACC
 - Declare and use constant memory
 - Analysis local variables and scalars to reduce number of arguments being passed into each kernel



Application Requirements

- User control over thread and block dimensions
 - Assigning multiple columns to a thread block
 - Small array dimensions (eg. < 64) mean less parallelism
 - array2D(64, 5000) - where nThreads=64, nBlocks = 5000
 - Better performance by striding across block dim
 - » Eg. nThreads = 4 * 64, nBlocks = 500/4
 - Assigning threads and blocks to the same array dimension
 - Single large array dimension with block and thread parallelism
 - array2D (5000, 64), where 2nd dimension cannot be parallel
 - Contiguous memory references for threading
 - Stride across thread dimension for each block



Application Requirements

- Directives to support
 - Promotion of variables
 - Added dimension needed for block or thread parallelization
 - 2D array + + + > 3D array
 - Needed for correctness & performance
 - Demotion of variables
 - 1D arrays - - -> GPU register variables
 - 2D arrays - - -> shared memory arrays
 - Results in improved performance



Conclusion

- Committed to a single source
 - We anticipate significant challenges for legacy codes
- We will continue to use and develop F2C-ACC
 - Until commercial compilers mature
 - Intend to use in tandem with commercial compilers for now
 - Provide Fortran-to-Fortran code transformations where there are gaps in the commercial compilers
- More diagnostics will simplify parallelization
 - Compilers state when parallelization is done
 - They don't state why parallelization could not be done

