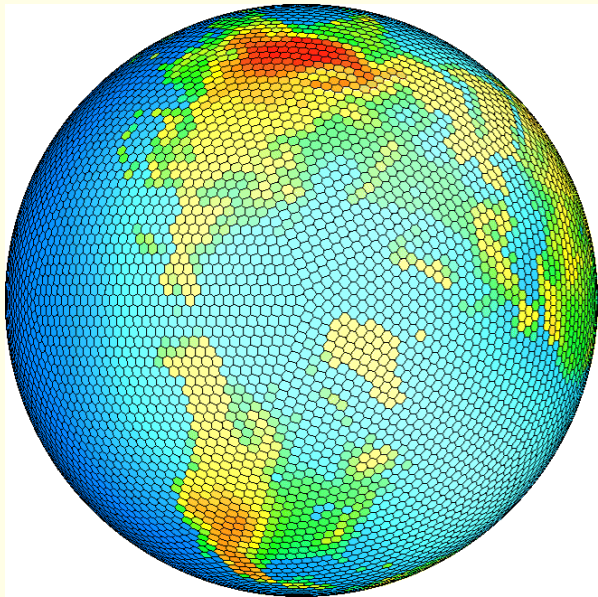
# Performance and Scaling of the NIM Global NWP Model on GPUs

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

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



# Outline

- A bit about GPUs
- A wee bit about the Non-hydrostatic Icosahedral Model (NIM)
- 2011:  Westmere vs. Fermi
- 2012:  Sandybridge vs. Kepler
- NIM scaling
- Commercial GPU compilers
- Future directions

# Moore's Law "Free Ride" is Over

- CPU clock frequencies have stalled
- CPU vendors moving to more cores per chip to increase performance
- Computational Accelerators are back…
  - GPGPU (NVIDIA, AMD)
    - General-Purpose Graphics Processing Unit
    - 1000s of simple cores
    - Already on HPC top-500 list
    - Initial NWP work by Michalakes
  - MIC (Intel)
    - Many Integrated Core
    - 10s (?) of cores
    - Under development
- Common theme: *exploit fine-grained parallelism*

# GPU Fine-Grained Parallelism

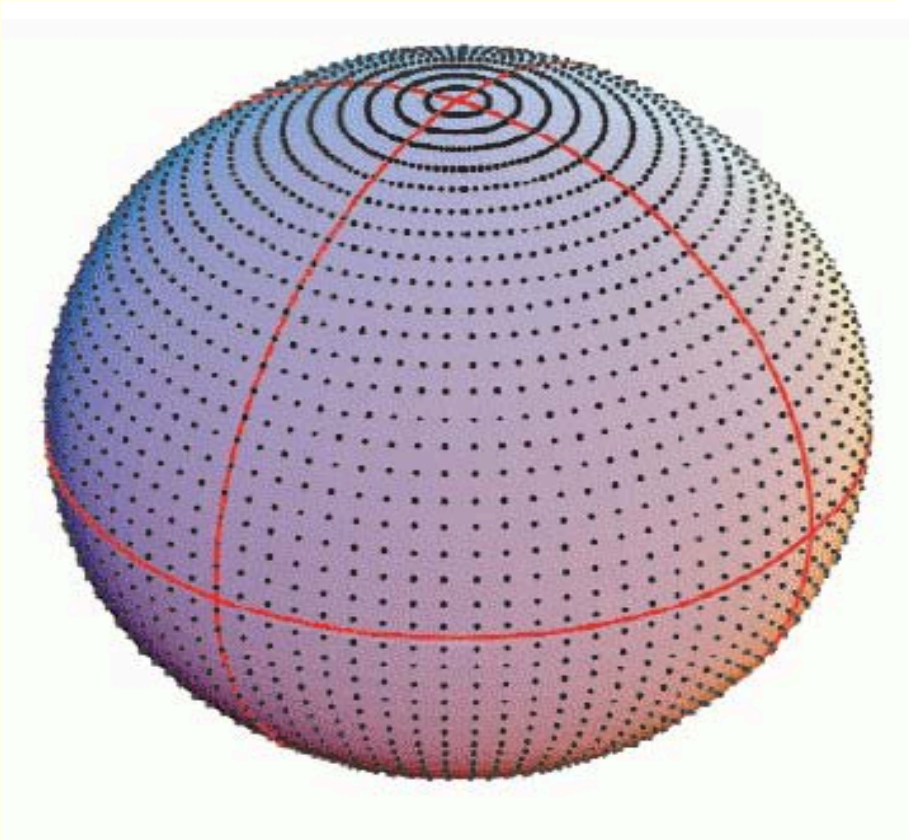
- Large on-chip ("*global*") memory
  - Higher bandwidth than CPUs
  - High latency (100s of cycles)
    - Need lots of threads to hide memory latency
    - Code must vectorize well
- Slow data transfers between CPU & GPU
  - *User must efficiently manage transfers*
- Limited per-thread fast ("*shared*") memory & registers
  - ***User manages memory hierarchy***

# For NWP CPU is a "Communication Accelerator"

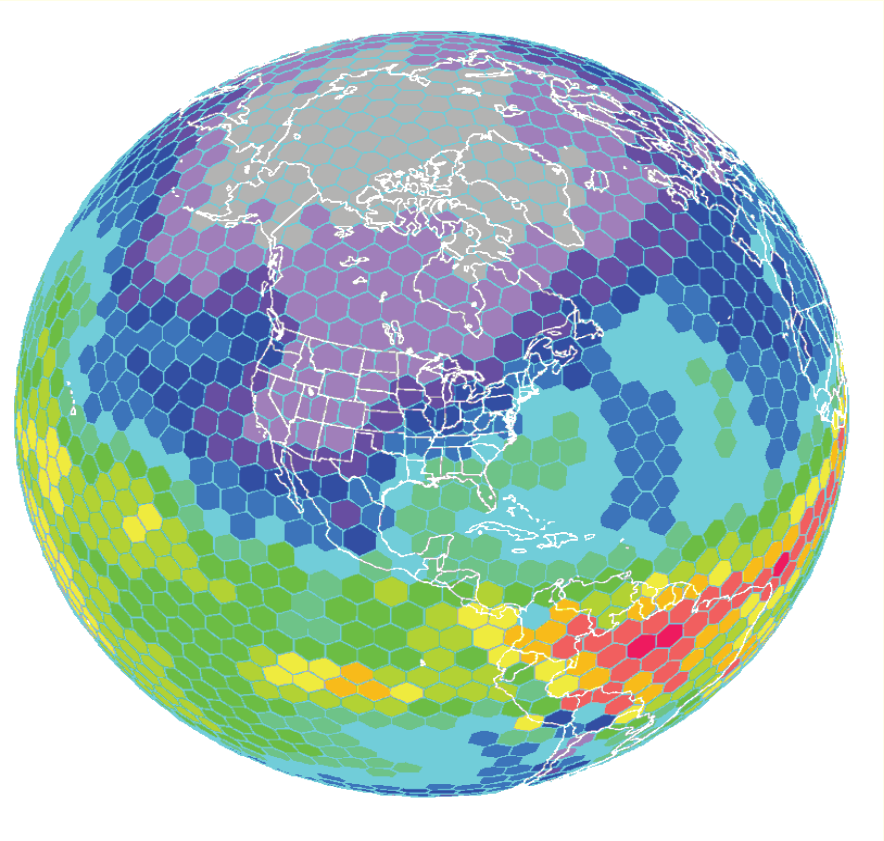
- Invert traditional "GPU-as-accelerator" model
  - Model state lives on GPU
  - Initial data read by the CPU and passed to the GPU
  - Data passed back to the CPU only for output & message-passing
  - GPU performs all computations
    - Fine-grained parallelism
  - CPU controls high level program flow
    - Coarse-grained parallelism
- Minimizes overhead of data movement between CPU & GPU

# Icosahedral (Geodesic) Grid: A Soccer Ball on Steroids

**Lat/Lon Model**



**Icosahedral Model**



- **Near constant resolution over the globe**
- **12 pentagons + lots of hexagons**



(slide courtesy Dr. Jin Lee)

# NIM NWP Dynamical Core

- NIM = "Non-Hydrostatic Icosahedral Model"
  - New NWP dynamical core
  - Target: global "cloud-permitting" resolutions ~3km (42 million columns)
  - Rapidly evolving code base
- "GPU-friendly" (also good for CPU)
  - Single-precision floating-point computations
  - Computations structured as simple vector ops with indirect addressing and inner vertical loop
- Coarse-grained parallelism via Scalable Modeling System (SMS)
  - Directive-based approach to distributed-memory parallelism built upon MPI

# NIM Requirements

- Must maintain single source code for all desired execution modes
  - Single and multiple CPU
  - Single and multiple GPU
  - Prefer a directive-based Fortran approach for GPU
    - Use F2C-ACC and commercial compilers
- Can tolerate less stable HPC platforms for research
  - “Opportunity” FLOPS

# 2011 Performance Results

- 225km test case
  - 10242 columns, 96 levels, 1000 time steps
  - Expect similar number of columns on each GPU at ~3km target resolution
- CPU = Intel Westmere (2.8GHz)
- GPU = NVIDIA C2050 “Fermi”
- Optimize for both CPU and GPU
  - Some code divergence
  - Always use fastest code

# Good Performance on CPU

- Used PAPI to count flops (Intel compiler)
  - Requires –O1 (no vectorization) to be accurate!
  - 2nd run with –O3 (vectorization) to get wallclock

$$\frac{2.8 * 10^{12}}{940} = 2.98 Gflops/\sec$$

~27% of peak on Westmere 2.8 GHz

# 2011: Fermi GPU vs. Westmere CPU, 225km 96-level

| NIM routine | CPU 1-core Time (sec) | CPU 6-core Time (sec) | C2050 GPU Time (sec) | C2050 GPU Speedup vs. 6-core CPU |
|---|---|---|---|---|
| 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 |

# 2012 NIM Changes

- New model code ready ~3 weeks ago
  - Some changes slowed GPU
- Rapid development to port to GPU
- Significant performance tuning on CPU
  - Sped up “vdmint*” by 47% by improving re-use
  - GPU ***cannot*** take full advantage of this due to per-thread resource limitations
    - Working to resolve this…

# 2012 CPU & GPU Changes

- Bitwise-exact comparison now possible between GPU & CPU!
  - nvcc compiler flags: “-ftz=true –fmad=false”
  - Avoid library functions including “pow” (**)
  - Greatly speeds up debugging
    - NIM now has a run-time switch to run “**” operations on CPU for automated testing
  - Do not underestimate the value of bitwise-exact comparisons!
  - *Will it last?*

# 2012 CPU & GPU Changes

- Sandybridge memory bandwidth boost
- Access to new Kepler GPU ~1 week ago
  - K10 GPUs via NVIDIA
    - 3x more cores per socket (1536)
    - Slower clock frequency
    - 2x more FLOPs per Watt
  - Limited performance tuning on K10
    - Valuable assistance from NVIDIA's Paulius Micikevicius
      - Only about half of Paulius' performance improvements incorporated so far

# 2012: CPUs vs. GPUs vs. 225km, 96-level

| 6-core Westmere 1 socket | 16-core Sandybridge 2 sockets | M2090 Fermi GPU 1 socket | K10 Kepler GPU 1 socket* | K10 Kepler GPU 2 sockets (estimated) |
|---|---|---|---|---|
| 94.7 | 29.8 | 20.2 | 28.8 | ~16 |

- Short 100-time-step runs
  - I/O ***not*** included
- Power consumption of one M2090 socket matches power consumption of two K10 sockets
- 2-socket K10 performance estimated

# 2012: CPUs vs. GPUs

- Good fraction of peak performance on CPU
  - 29% of peak FLOPs on single Westmere core
    - Peak is 11.2 GFLOPs
- Much more difficult to get close to peak FLOPs on GPU
  - 2% of peak FLOPs on one K10 GPU
    - Peak is 2288 GFLOPs

# NIM Scaling

- F2C-ACC + SMS directives
  - Identical results using different numbers of GPUs
  - Scaling is worse because compute has sped up but communication has not
  - Working on communication optimizations
    - Naïve decomposition…
- *Demonstrates that single source code can be used for single/multiple CPU/GPU runs*

# NIM Performance Summary

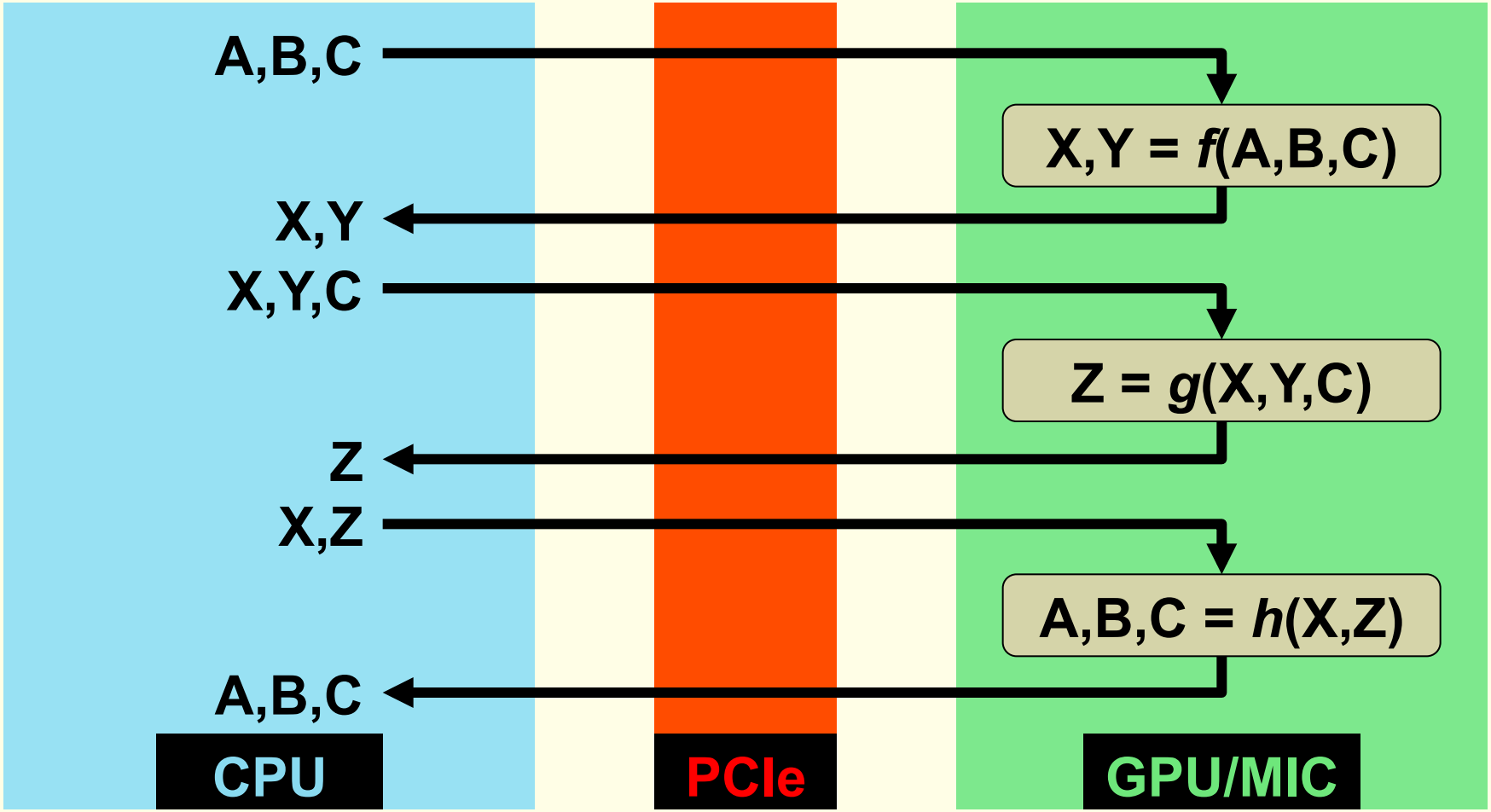
- 2xK10 is ~6x faster than 6-core Westmere
- 2xK10 is ~2x faster than 16-core Sandybridge
  - ~3x is possible with further tuning on K10
  - ~4x is not likely
  - Power consumption?
- Debugging and validation are more difficult on GPUs although this has greatly improved
- NIM scaling will improve with better grid decomposition

# Commercial GPU Compilers

- Fortran directive-based products from CAPS (HMPP), PGI (Accelerator), Cray (beta), [Pathscale]
  - Converging on OpenACC/OpenMP
- We are using F2C-ACC to demonstrate features needed by NIM & FIM
  - Vendors are listening and responding
  - Commercial compilers are approaching performance of F2C-ACC on NIM
  - Next whiny complaint: compilers should hide data transfer complexity…

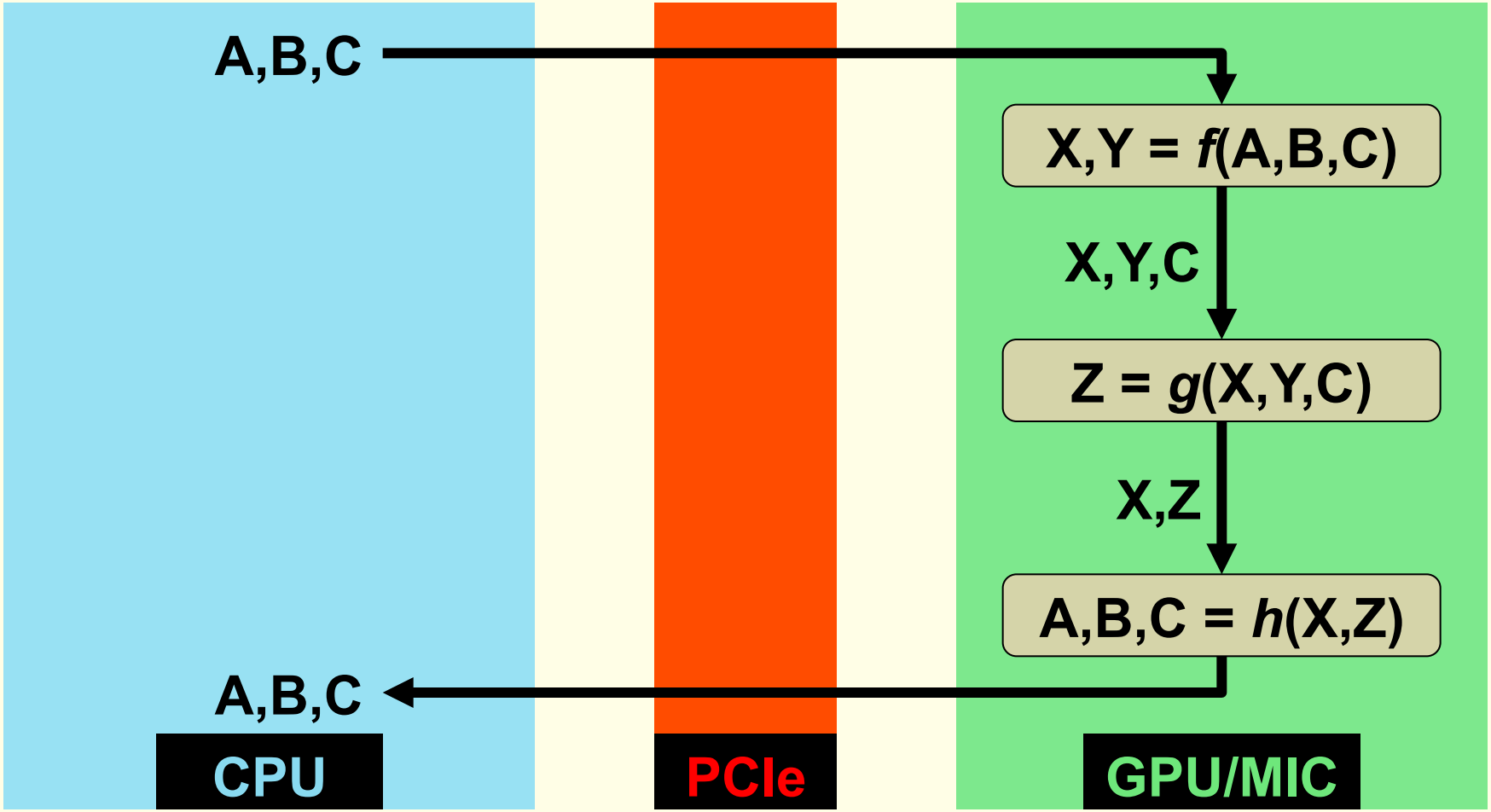
# Host-Device Data Transfers

- "Accelerator" model is well-supported



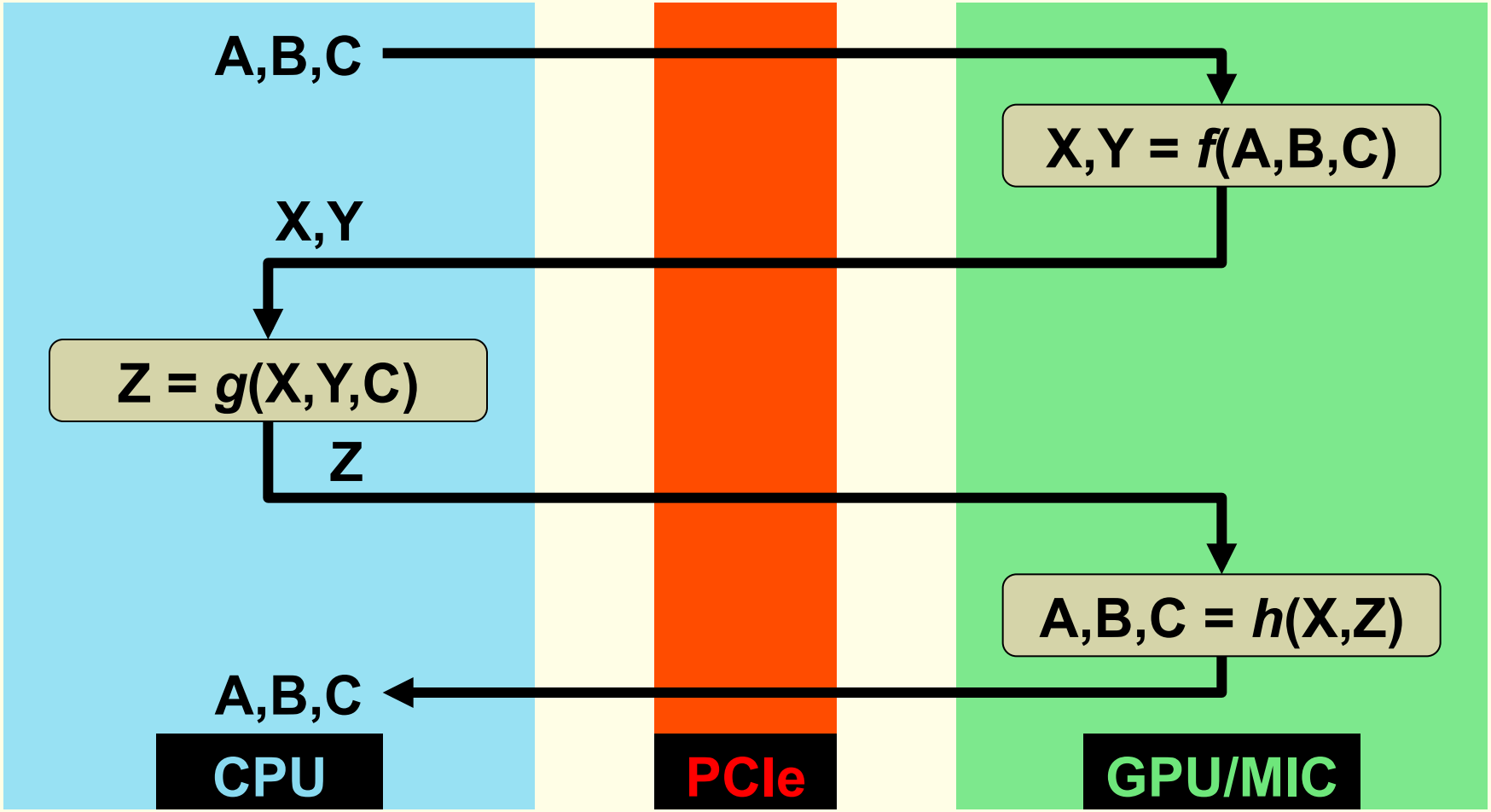
# Host-Device Data Transfers

- "State on Accelerator" is a bit harder



# Host-Device Data Transfers

- Per-kernel validation is painful!



# Near-Future Directions

- Continue to improve GPU performance
- Improve multi-GPU scaling
- Continue to work with compiler vendors
- K20, ORNL “Titan”

# Thanks to…

- Francois Bodin, Guillaume Poirier, and others at CAPS for assistance with HMPP
- Pete Johnsen at Cray for assistance with Cray OpenACC GPU compiler
- Dave Norton at PGI for assistance with PGI Accelerator
- Paulius Micikevicius at NVIDIA
- We want to see multiple successful commercial directive-based Fortran compilers for GPU/MIC

# Thank You