

# GPGPUs for High Performance Computing

Mark Govett

National Oceanic and Atmospheric Administration  
Earth System Research Laboratory



# Outline

- Background
- GPU Hardware
- GPU Programming Environment
- GPU Parallelization of NIM



# Organizational Structure

- NOAA



**OPERATIONS**

- National Weather Service

- National Centers for Environmental Prediction (NCEP)

- Oceanic & Atmospheric Research

- Earth System Research Laboratory
  - Global Systems Division
    - » Advanced Computing Section



**RESEARCH**

- NOAA Environmental Satellite Data and Information Service (NESDIS)

- National Ocean Service



# What is Numerical Weather Prediction?

- Weather vs. Climate
  - Is it going to rain tomorrow?
  - Will the next 10 years be warmer than the last 10 years?
- NWP = computer simulations behind weather forecasts
  - National Weather Service forecasters use NWP models for guidance
- Slow but steady forecast improvements since the “birth” of NWP in the 1950s



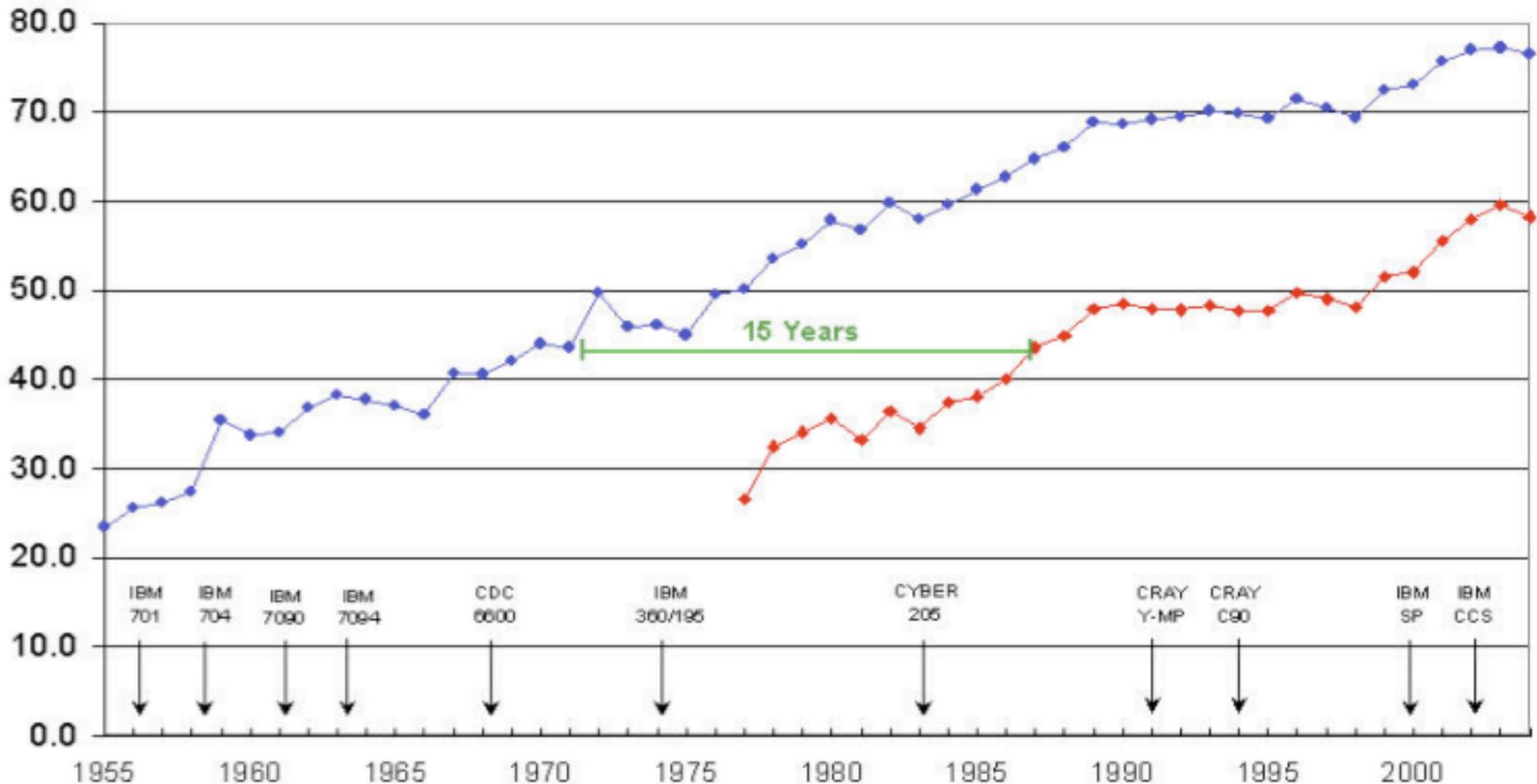


# NCEP Operational Forecast Skill

## 36 and 72 Hour Forecasts @ 500 MB over North America

[100 \* (1-S1/70) Method]

—●— 36 Hour Forecast      —●— 72 Hour Forecast



Thanks to Bruce Webster, NOAA

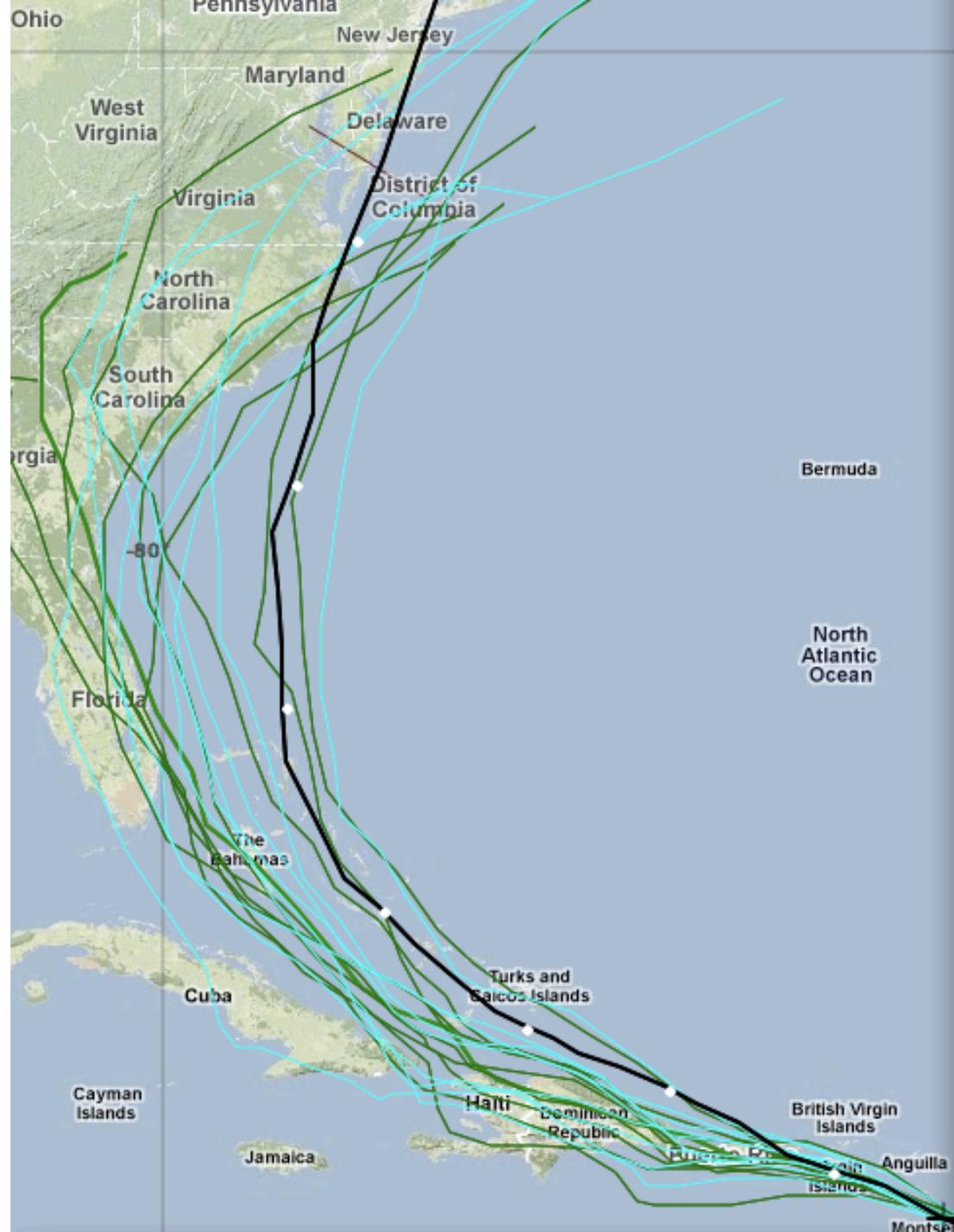
# What is Numerical Weather Prediction?

- Many scientific challenges
  - Hurricane track forecast good but not hurricane intensity forecast
  - Aviation weather
    - Regional to local forecasts
    - Impact on flight delays
  - Fire weather
    - Understanding local, short range weather



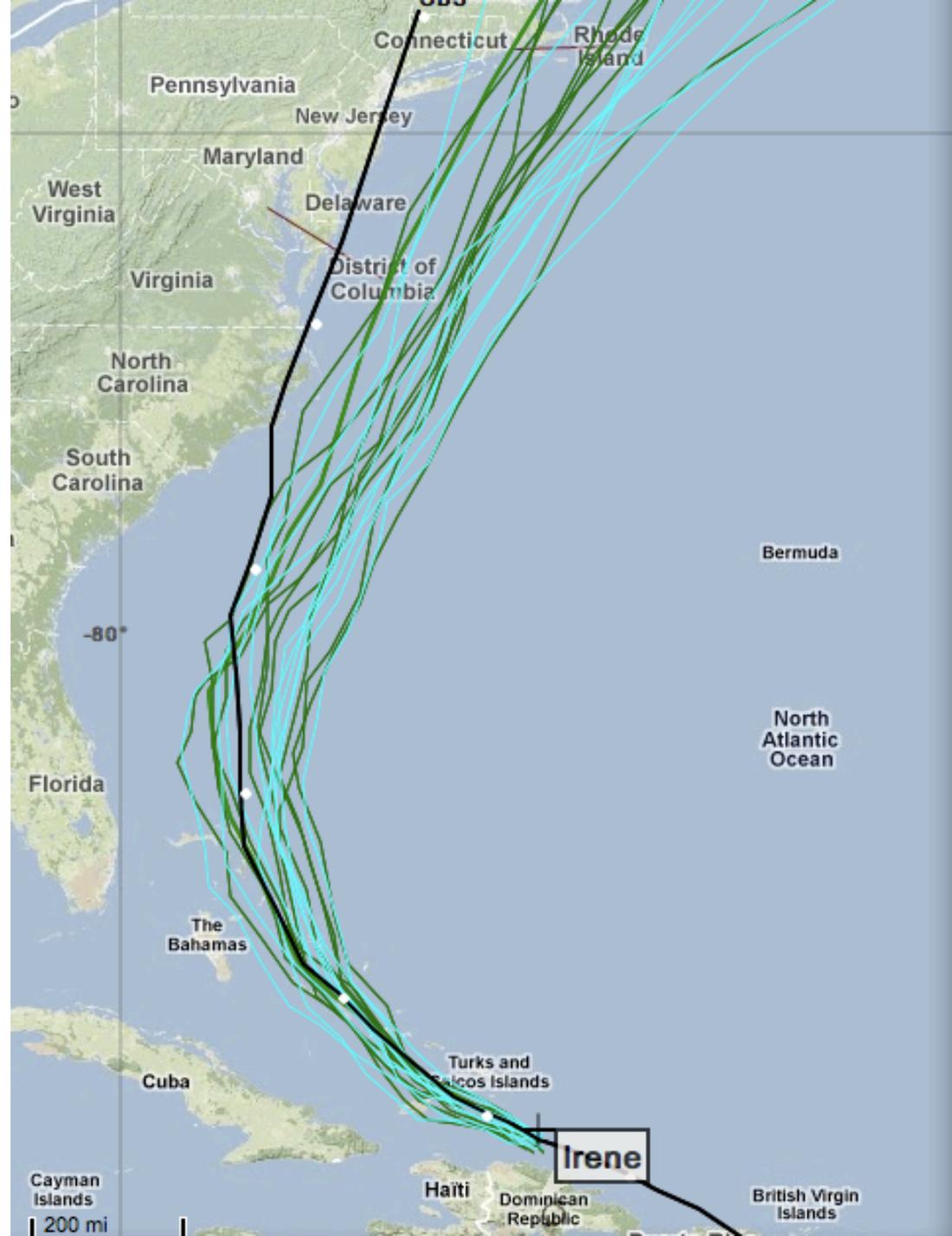
# Hurricane Irene 7-Day Forecast

- Forecast valid @ 8/21/2011 12Z UTC
- Black line is observed track
  - White diamonds are storm location at 00z UTC
- Green & cyan lines are forecast tracks from NWP models



# Hurricane Irene 5-Day Forecast

- Forecast valid @ 8/23/2011 12Z UTC



# Hurricane Irene 3-Day Forecast

- Forecast valid @ 8/25/2011 12Z UTC
- Forecast uncertainty (estimated from spread of forecast tracks) already smaller than the hurricane



# What is Numerical Weather Prediction?

- NWP is a big consumer of HPC
  - Forecast models must run 50x faster than real-time
- “Global” vs. “regional”
  - Regional = higher resolution over smaller domain
    - Because computers are never big enough
- “Dynamics” vs. “Physics”
  - Dynamics predicts the evolution of the explicitly resolved flow
  - Physics estimates the effects of subgrid-scale processes (e.g. clouds, convection at traditional global resolutions > 10km)





# The Rapid Refresh (**RR**) and High Resolution Rapid Refresh (**HRRR**):

## Short-range guidance for high impact weather

**Stan Benjamin**

**Steve Weygandt, Ming Hu,  
Tanya Smirnova, Geoff Manikin,  
Kevin Brundage, John Brown,  
Bill Moninger, Georg Grell,  
Brian Jamison, Steven Peckham,  
Patrick Hofmann, Eric James**

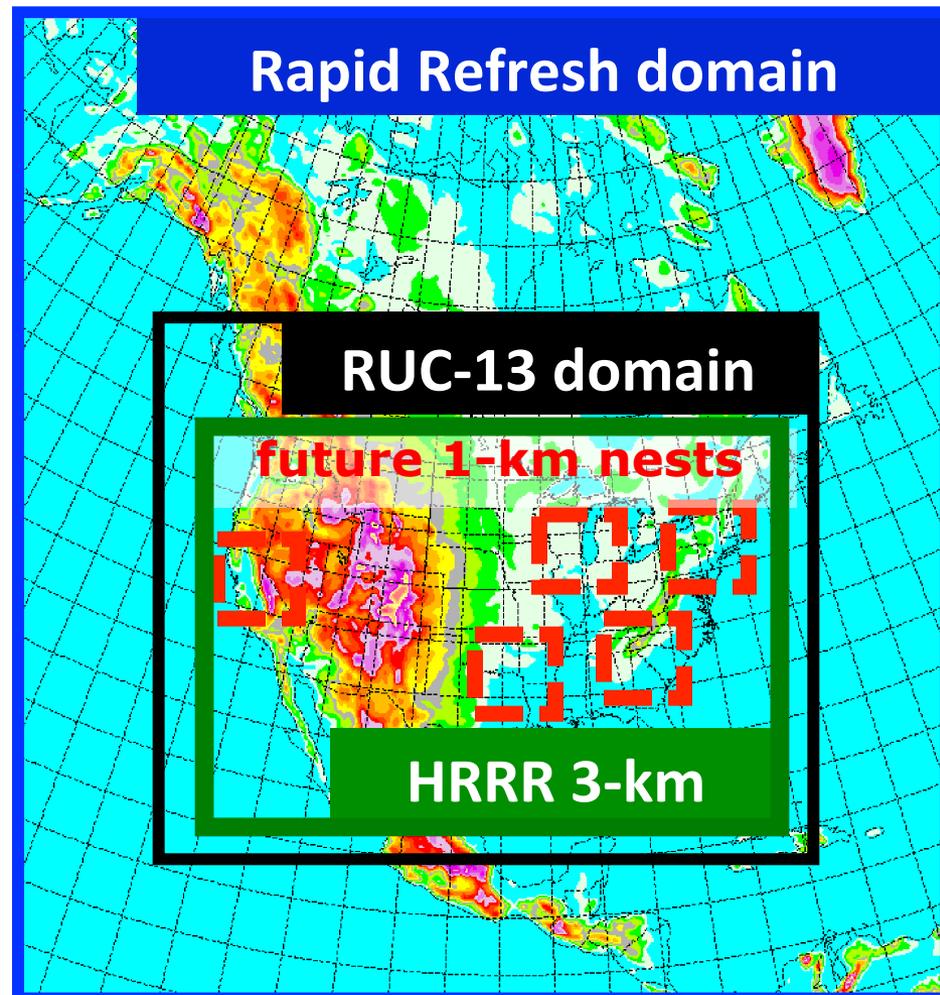
**Assimilation and Modeling Branch  
Global Systems Division**

[rapidrefresh.noaa.gov](http://rapidrefresh.noaa.gov)



**Earth System Research Laboratory**

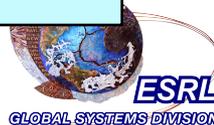
SCIENCE, SERVICE & STEWARDSHIP



# Why have a **Rapid Refresh** or **HRRR**?

- Provide high-frequency (hourly) mesoscale analyses, short range model forecasts
- Assimilate (“merge”) all available observations into single physically consistent 3-d grid such that forecasts are improved
- **Focus on sensible weather, aviation and other hazards:**
  - Thunderstorms, severe weather, winter storms
  - Detailed surface temperature, dewpoint, winds
  - Icing, ceiling and visibility, turbulence
  - Upper-level winds
- **Users:**
  - NWS and other forecasters
  - aviation and other transportation
  - severe weather forecasting
  - hydrology, energy (load, renewable)

***“Situational  
Awareness  
Model”***



# Model Development Activities

- Regional, Local Models (1-5 KM)
  - NOAA HRRR, WRF-ARW, WRF-NMM, etc
    - Hurricanes, Aviation, Fires, Chemistry / Ash
  - Ensembles (15-30KM)
- Global Models (10-30 KM)
  - NOAA FIM model
  - Improved hurricane forecasts

## Computing Requirements

- 3000 cores:
  - 15KM Global FIM
- 126,000 cores
  - 21 member 30 KM ensemble



# Global Cloud Resolving Models (GCRM)

< 2KM horizontal resolution

- Benefits
  - Weather and climate scales (5-100 day forecasts)
    - Improved hurricane track and intensity prediction
    - Regional climate impacts
- Active developments in the research community
  - NICAM: University of Tokyo
  - GCRM: Colorado State University
  - GFDL: Finite Volume Cubed Sphere
  - NIM: NOAA Earth System Research Laboratory
- Computing Requirements
  - CSU's 4KM GCRM was run on 80,000 cores of DOE Jaguar
    - Simulations ran at ~50 percent of real-time



# Why FORTRAN?

- De-facto language of NWP
  - # of atmospheric modelers >> # of SEs in domain
- Good language for HPC
  - “Formula-translation”
    - Especially multi-dimensional arrays
  - 50+ years of automatic optimizations
  - Strong support for efficient floating-point operations



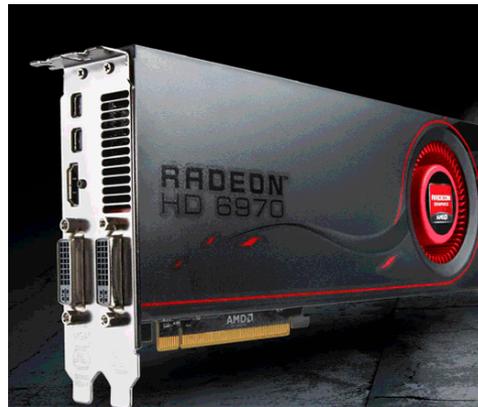
# Outline

- Background
- **GPU Hardware**
- GPU Programming Environment
- GPU Parallelization of NIM



# Graphics Processing Unit (GPU)

- CPU co-processor for compute intensive calculations
  - Focused on supporting video game applications
  - NVIDIA sold over 100 million GPUs in 2010
    - Low cost, high performance
  - Appear on most desktops and laptop systems
- Supports hundreds to thousands of lightweight CPU cores on a single chip



AMD Radeon



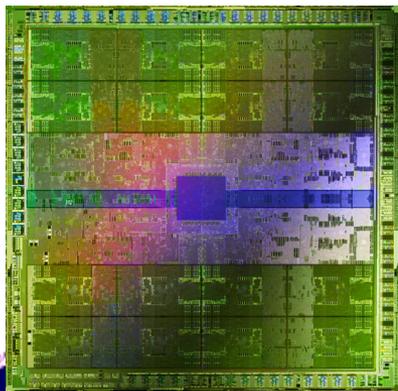
Xbox 360



# GPU / Multi-core Technology

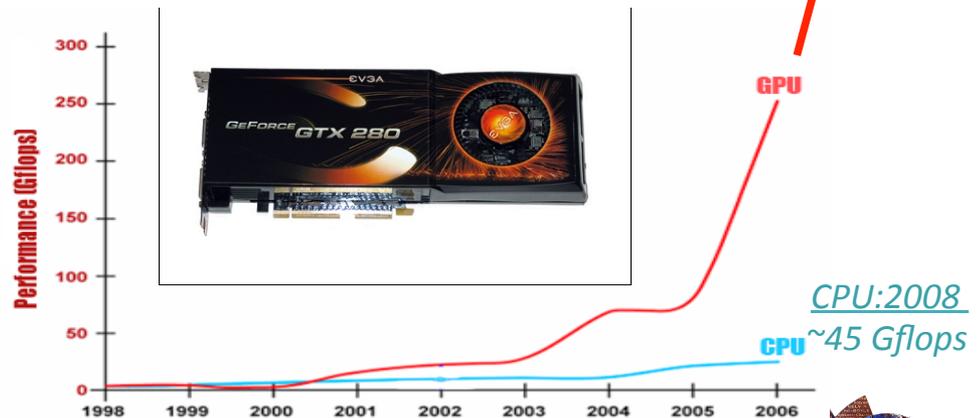
- NVIDIA: Fermi chip first to support HPC
  - Formed partnerships with Cray, IBM on HPC systems
  - #1, #3 systems on TOP500 (Fermi, China)
- AMD/ATI: Primarily graphics currently
  - #7 system on TOP500 (AMD-Radeon, China)
  - Fusion chip in 2011 (5 TeraFlops)
- Intel: Many Integrated Core (2012), 32-64 cores

NVIDIA: Fermi (2010)



- ◇ 1.2 TeraFlops
- ◇ 8x increase in double precision
- ◇ 2x increase in memory bandwidth
- ◇ Error correcting memory

NVIDIA: Tesla (2008)





The desire for visually complex and realistic games is driving this market, not HPC

Super Mario Galaxy Screenshot - <http://top10kid.com/2009/08/13/top-10-most-anticipated-games-of-2010/>



# CPU / GPU Hardware Comparison (by the numbers)

- GPU is ~8x faster, requires 2x more power

Card	year	Cores	FlopsSP (GF)	FlopsDP (GF)	MemBW (GB/s)	Memory GB	ECC?	Peak Power (W)
AMD 6970	2010	1536	2700	685	176	2	Partial	250
Nvidia GTX 280	2008	240	933	90	141.7	1	No	236
Nvidia C2050	2010	448	1030	515	144	3	Yes	238
Nvidia C2090	2011	512	1331	665	177	6	Yes	225
Intel MIC	2012	32	> 1000	??	??	1-2	Unk	???
Intel Westmere	2010	6	160	80	26.4	48	Yes	120



# PetaFlop Computing

## DOE Jaguar System

- 2.3 PetaFlops
- 250,000 CPUs
- 284 cabinets
- 7-10 MW power
- Cost: ~ \$100 million
- Building: \$75 million



**Reliability in hours**

## Equivalent GPU System

- 2.3 PetaFlop
- 2000 Fermi GPUs
- 20 cabinets
- 1.0 MW power
- Cost: ~ \$10 million
- Building: \$5 million
- **Reliability in weeks**

- Large CPU systems (>100 thousand cores) are unrealistic for operational weather forecasting
  - Power & Cooling: ~ 10x
  - Reliability: hours versus weeks
  - Cost: ~ 10x
    - ~15x including facilities cost



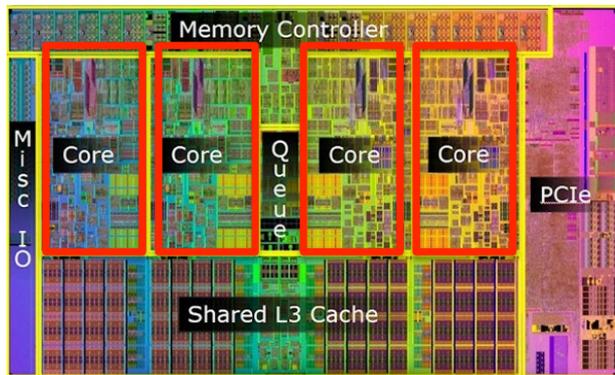
Valmont  
Power Plant  
~200 MegaWatts  
Boulder, CO



# CPU – GPU Comparison

- CPUs focus on per-core performance
  - Chip real estate devoted to cache, speculative logic
  - Westmere: 6 cores, 160 Gflops, 120 Watts (~1 GFlop /Watt)
- GPUs focus on parallel execution
  - Fermi: 512 cores, 1300 Gflops, 225 Watts ( ~5 Gflops / Watt)

CPU: Nehalem (2009)



GPU: Fermi (2010)



# Outline

- Background
- GPU Hardware
- **GPU Programming Environment**
- GPU Parallelization of NIM



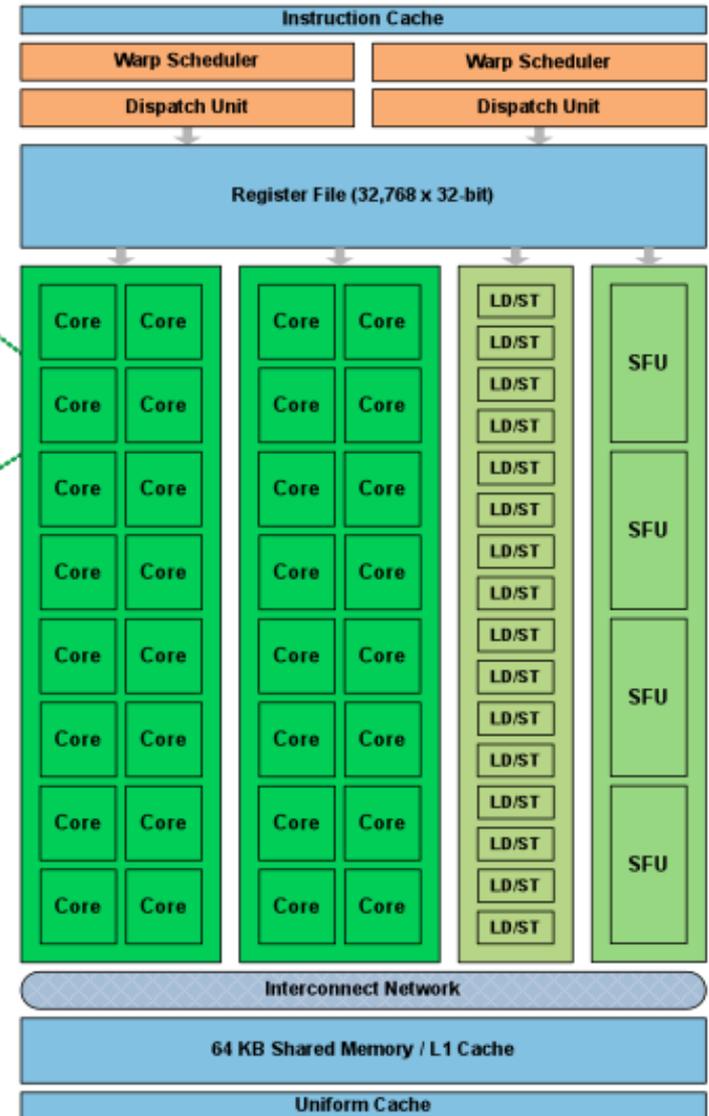
# GPU Programming Environment

- Graphic Processing Unit
  - Co-processor to support high-end graphics
    - Video games, Xbox, etc
    - Millions sold annually
  - Languages tailored for video graphics operations
    - Shaders (Direct3D, OpenGL)
    - Difficult to adapt for scientific computing
- General Purpose Graphics Processor Unit (GPGPU)
  - NVIDIA developed CUDA
    - Enable more realistic games in terms of physical calculations
      - Eg debris, smoke, fire, fluids other special effects
    - High level language based on C + GPU extensions
      - Copying data between CPU and GPU
      - Invoking GPU device
      - Defining parallelism



# Fermi Computing Environment

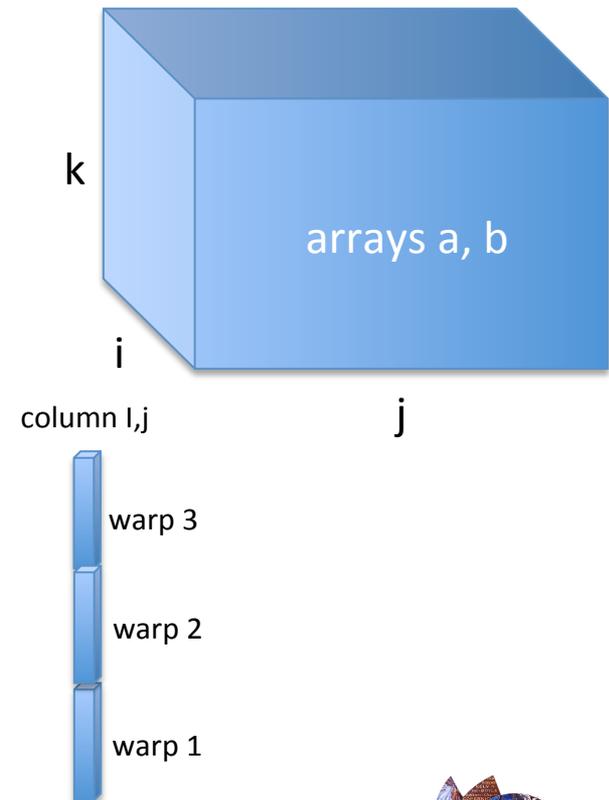
- Work Scheduler:
  - Divides up blocks of work, sends to SMs
- 16 Streaming Multiprocessors (SM)
  - Executes one or more blocks of work
- 1-8 blocks can execute almost simultaneously
  - Dependent on finite resources available on the SM
- Blocks are divided into warps
  - Unit of execution
    - 32 cores execute in lock step
  - Fast (1-2 cycles) context switching between “ready” warps
    - Effective at hiding memory latency



# Execution Example

- Each point in A,B are updated almost simultaneously by 16 SMs, operating on 1/16<sup>th</sup> of the work
  - Blocking over horizontal
  - Threading over vertical
    - 3 warps of 32 threads each

```
do i = 1, 100      !blocking
do j = 1, 10000   !blocking
do k = 1, 96      !threading
  a(k,i,j) = b(k,i,j)
enddo
enddo
enddo
```



# Execution Example (CUDA)

## Fortran

```
real a(96,100,10000)
real b(96,100,10000)

do i = 1, 100
do j = 1, 10000
do k = 1, 96
  a(k,i,j) = b(k,i,j)
enddo
enddo
enddo
```

## CUDA

```
float a[96*100*10000]
float b[96*100*10000]

i = blockIdx.x + 1;
j = blockIdx.y + 1;
k = threadIdx.x + 1;
a(MACRO(i,j,k)) =
    b(MACRO(k,i,j))
```

NOTE: MACRO refers to a C pre-preprocessing macro that collapses the multi-dimensional array reference to a single dimension



# SM Resources

- Limits the number of blocks that can be stored and queued for execution in the SM (max of 12)
  - 32K 32-bit registers
  - 64K shared memory / cache

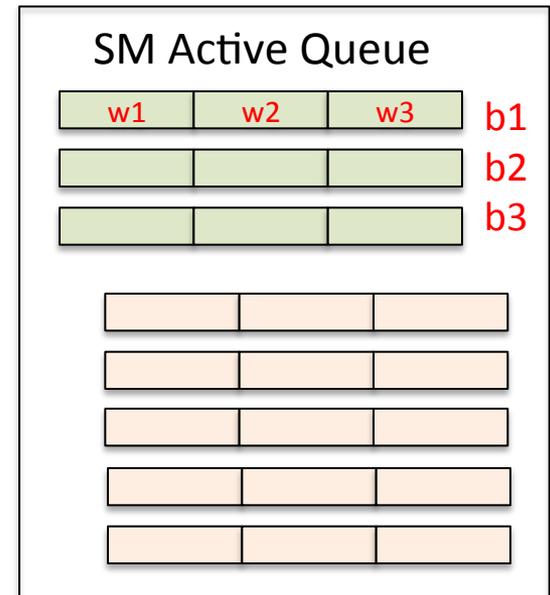
## GPU Kernel Resources

16K shared  
100 registers / thread  
96 vertical levels

## # Blocks for each SM

4

$$32000 / (100 * 96) \Rightarrow \mathbf{3}$$



- Solution: Smaller, simpler GPU kernels reduce resource use

- Data reuse must be considered too

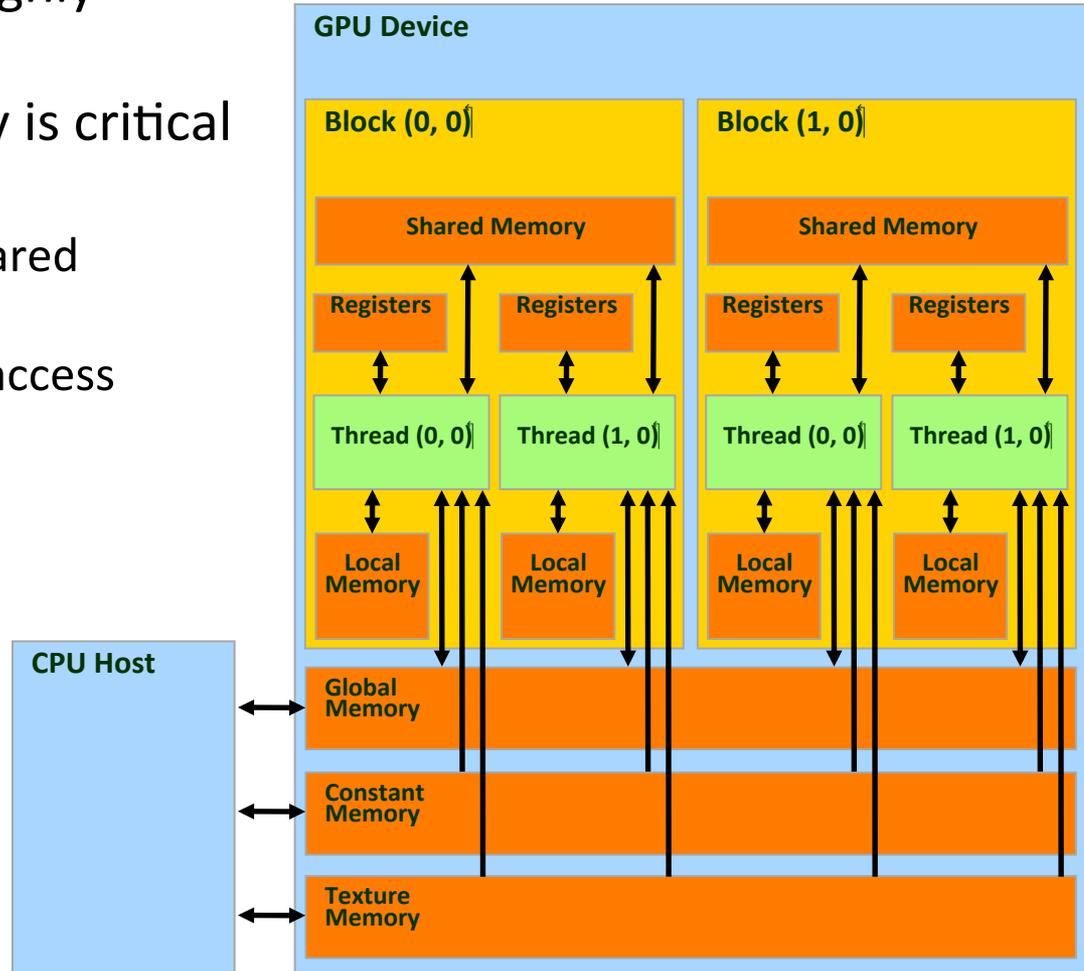


# Application Performance

- 20-50x is possible on highly scalable codes
- Efficient use of memory is critical to good performance
  - 1-2 cycles to access shared memory & registers
  - Hundreds of cycles to access global memory

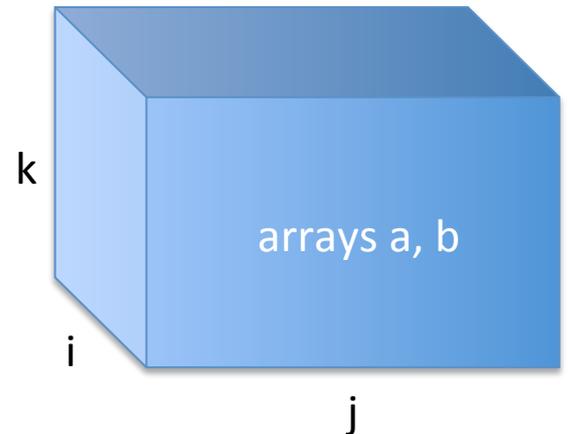
Memory	Tesla	Fermi
Shared	16K	64K
Constant	16K	64K
Global	1-2GB	4-6GB

GPU Multi-layer Memory



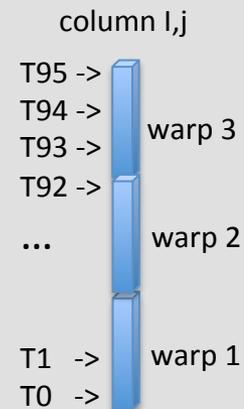
# Efficient Memory Access

- Organizing data according to how the gpu will access and use it is very important
- Coalesced Loads and Stores
  - For global memory access
  - Data that is contiguous in memory required by adjacent threads



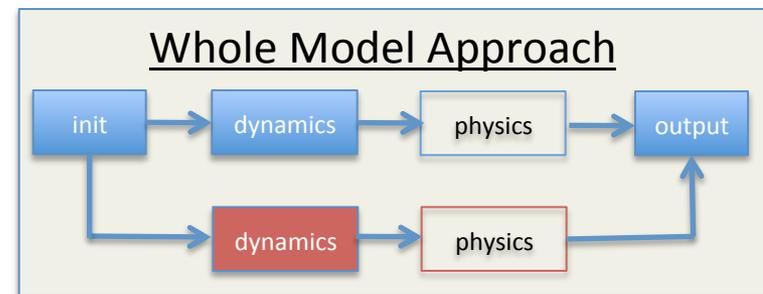
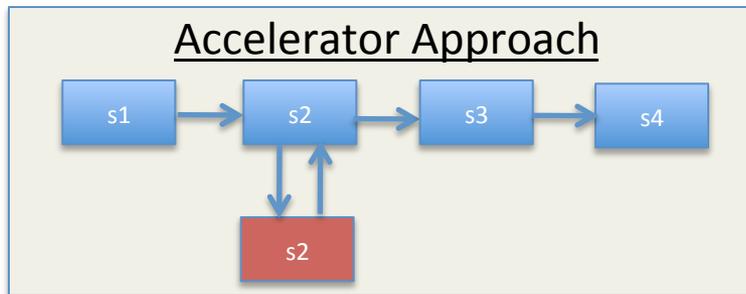
## Example

Threading over  $k$ , and referencing arrays as  $a(k, i, j)$  rather than  $a(i, j, k)$  allows global memory loads to be coalesced which can yield more than a 10x performance boost.



# GPU Parallelization Approaches

- Accelerator Approach
  - Target select routines for acceleration
  - Copy between CPU and GPU can be significant
  - Can be a step-wise approach to parallelization



- Whole Model Approach
  - Keep data resident on the GPU
    - Communications only needed for input, output and for inter-process communications
  - May requires code conversion, code restructuring or rewriting

# GPU Programming Approaches

- Language Approach
  - CUDA, OpenCL, CUDA Fortran, Python, Matlab
  - 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



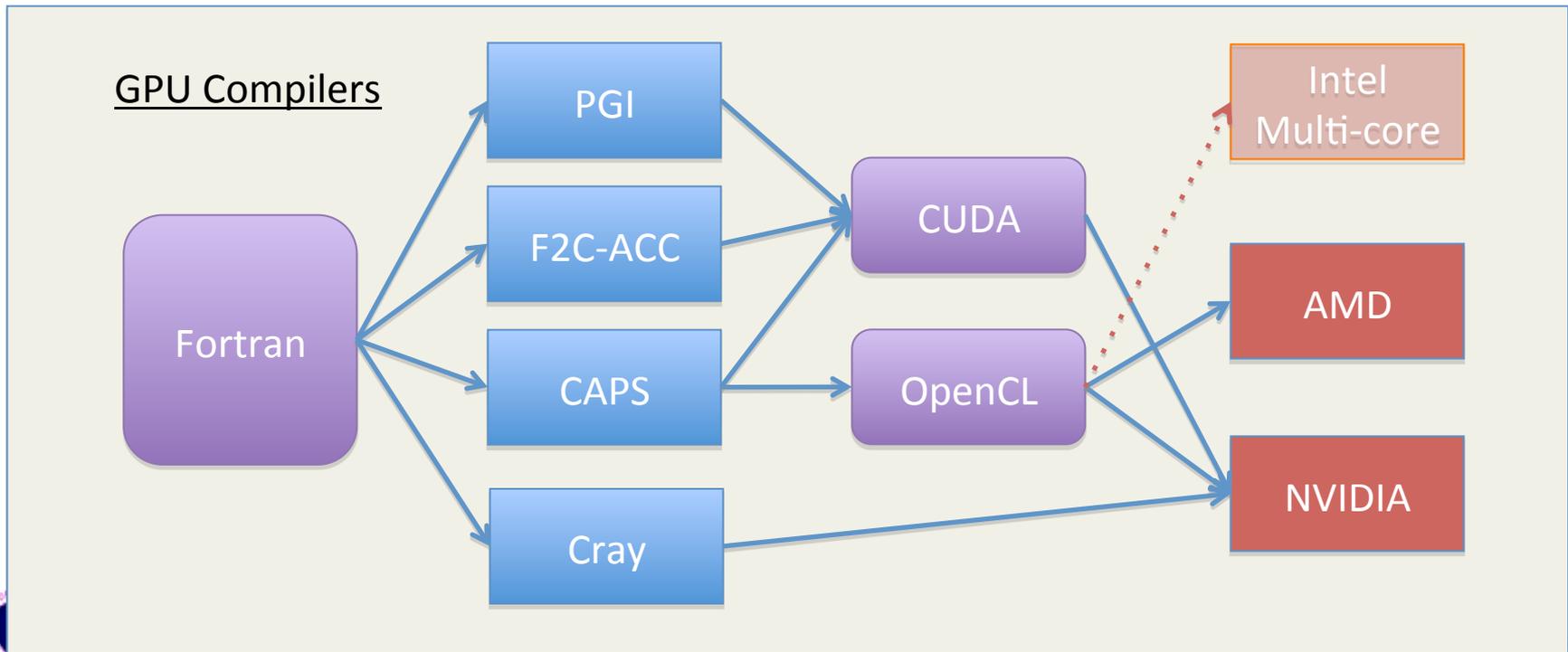
# Goal to Maintain a Single Source

- 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”



# Fortran GPU Compilers

- General Features
  - Do not support all Fortran language constructs
  - Converts Fortran into CUDA for further compilation
- CAPS – HMPP
  - Extensive set of parallelization directives to guide compiler analysis and optimization
  - Optionally generates OpenCL
- PGI
  - **ACCELERATOR** – directive-based accelerator
  - **CUDA Fortran** – Fortran + language extensions to support Kernel calls, GPU memory, etc
- F2C-ACC
  - Developed at NOAA for our models
  - Requires hand tuning for optimal performance



# CAPS-HMPP Compiler

- Multi-core Fortran
  - CUDA, OpenCL code generation
- Extensive set of directives
  - Parallelization
  - Optimization
- A minimal set of directives to get a working code
  - Compiler will do what it safely can and provide diagnostic information
- Our evaluation began ~4 months ago
  - Good documentation & support



The slide features the HMPP logo (a stylized 'H' in a circle) and the CAPS logo in the top right. A blue header reads 'Manycore portable programming'. The main content is organized into several sections: 'Benefit from the performance of GPU accelerated systems while reducing your development efforts', 'Portability' (with bullet points on investment protection, hardware independence, vendor API, and standard compilers), 'Scalability' (with bullet points on computation distribution, hardware interoperability, and OpenMP/MPI), 'Rapidly develop GPU accelerated applications with a source to source tool', 'A hybrid compiler with powerful CUDA - OpenCL generators' (with a flowchart showing the compilation pipeline from source to runtime on various hardware), 'HMPP, the best way to master performance', 'HMPP directives: a high level abstraction for manycore programming', and 'Dynamic application scaling'. A footer contains the website URL and the date 'May 2010'.

**Benefit from the performance of GPU accelerated systems while reducing your development efforts**

With the HMPP™ target generators, instantaneously prototype and evaluate the performance of the hardware-accelerated critical functions.

The code generators are specifically designed to extract the most of data parallelism from your C and Fortran kernels and translate them into NVIDIA® CUDA™ and OpenCL.

HMPP Workbench includes a C and Fortran compiler, code generators and a runtime that seamlessly integrate in your environment and make use of the NVIDIA CUDA, and OpenCL development tools and drivers.

**Portability**

- Protect your software investment
- Remain independent from the hardware platform
- Do not lock to a vendor specific API
- Work with C and Fortran standard compilers

**Scalability**

- Distribute computations between CPU and accelerators
- Provide hardware interoperability
- Complementary to OpenMP™ and MPI

**Rapidly develop GPU accelerated applications with a source to source tool**

**A hybrid compiler with powerful CUDA - OpenCL generators**

GPU programming and tuning directives

C/Fortran source-to-source compiler

HMPP preprocessor | CUDA back-end | OpenCL back-end

CPU Source | CUDA source | OpenCL source

C/Fortran Std. Compiler | CUDA compiler | OpenCL compilers

HMPP Runtime | CUDA driver | OpenCL drivers

Multicore CPU | NVIDIA TESLA | AMD ATI FireStream | NVIDIA TESLA

**HMPP, the best way to master performance**

**HMPP directives: a high level abstraction for manycore programming**

Based on a set of OpenMP™-like directives that preserve legacy codes, HMPP fully leverages the performance offered by most of today's stream processors and vector units. HMPP offers a standardized interface between your scientific algorithm and last evolving target code

by insulating hardware specific implementation of functions from your legacy code.

Complementary to OpenMP and MPI, HMPP lets you develop parallel hybrid applications that mix the best of today's available parallel tools.

**Dynamic application scaling**

By providing different target versions of computations that are offloaded to the available hardware compute units, an HMPP application dynamically adapts its execution to multi-GPUs systems and platform configuration. This guarantees the scalability and interoperability of your application.

**Developers**  
Rapidly integrate hardware accelerators in your application while preserving its portability.

**OEMs**  
Provide software developers with standard programming tools that keep their applications interoperable with your evolutive hybrid platforms.

**ISVs**  
Embed HMPP in your product and offer a single software that leverages the computing power of all the different configurations of your customers manycore platforms.

May 2010

<http://www.caps-entreprise.com>



# PGI Directives

- Directives are placed directly in the code body
  - Define an accelerated region
    - !\$acc region( [ copy | copyout | copyin ]) begin
    - !\$acc region end
  - User defines loop level parallelism
    - !\$acc do [ vector | parallel | unroll ]  
vector = thread, parallel = threadblock
  - Define data resident on the GPU
    - !\$acc data region ( copy | copyin | copyout )



# F2C-ACC GPU Compiler

- Developed to speed parallelization of NIM
  - Commercial compilers were not available in 2008
  - 25x speedup over 1 CPU core in 2009
- 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 & diagnostics
    - 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>



# F2C-ACC Translation to CUDA (Input Fortran Source)

```
subroutine SaveFlux(nz,ims,ime,ips,ipe,ur,vr,wr,trp,rp,urs,vrs,wrs,trs,rps)
implicit none
<input argument declarations>

!ACC$REGION(<nz>,<ipe-ips+1>,<ur,vr,wr,trp,rp,urs,vrs,wrs,trs,rps:none>) BEGIN
!ACC$DO PARALLEL(1)
do ipn=ips,ipe
!ACC$DO VECTOR(1)
  do k=1,nz
    urs(k,ipn) = ur (k,ipn)
    vrs(k,ipn) = vr (k,ipn)
    trs(k,ipn) = trp(k,ipn)
    rps(k,ipn) = rp (k,ipn)
  end do !k loop
!ACC$THREAD(0)
  wrs(0,ipn) = wr(0,ipn)
!ACC$DO VECTOR(1)
  do k=1,nz
    wrs(k,ipn) = wr(k,ipn)
  end do !k loop
end do !ipn loop
!ACC$REGION END

return
end subroutine SaveFlux
```



# F2C-ACC Translated Code (Host)

```
extern "C" void saveflux_ (int *nz__G,int *ims__G,int *ime__G,int *ips__G,int *ipe__G,float *ur,float
    *vr,float *wr,float *trp,float *rp,float *urs,float *vrs,float *wrs,float *trs,float *rps) {

    int nz=*nz__G;
    int ims=*ims__G;
    int ime=*ime__G;
    int ips=*ips__G;
    int ipe=*ipe__G;

    dim3 cuda_threads1(nz);
    dim3 cuda_grids1(ipe-ips+1);

    extern float *d_ur;
    extern float *d_vr;
    < Other declarations >

    saveflux_Kernel1<<< cuda_grids1, cuda_threads1 >>>
        (nz,ims,ime,ips,ipe,d_ur,d_vr,d_wr,d_trp,d_rp,d_urs,d_vrs,d_wrs,d_trs,d_rps);
    cudaThreadSynchronize();
    // check if kernel execution generated an error
    CUT_CHECK_ERROR("Kernel execution failed");
    return;
```



# F2C-ACC Translated Code (Device)

```
#include "ftnmacros.h"

//!ACC$REGION(<nz>,<ipe-ips+1>,<ur,vr,wr,trp,rp,urs,vrs,wrs,trs,rps:none>) BEGIN
__global__ void saveflux_Kernell(int nz,int ims,int ime,int ips,int ipe,float *ur,float *vr,float
    *wr,float *trp,float *rp,float
    *urs,float *vrs,float *wrs,float *trs,float *rps) {
    int ipn;
    int k;
    //!ACC$DO PARALLEL(1)
        ipn = blockIdx.x+ips;
    // for (ipn=ips;ipn<=ipe;ipn++) {
    //!ACC$DO VECTOR(1)
        k = threadIdx.x+1;
    // for (k=1;k<=nz;k++) {
        urs[FTNREF2D(k,ipn,nz,1,ims)] = ur[FTNREF2D(k,ipn,nz,1,ims)];
        vrs[FTNREF2D(k,ipn,nz,1,ims)] = vr[FTNREF2D(k,ipn,nz,1,ims)];
        trs[FTNREF2D(k,ipn,nz,1,ims)] = trp[FTNREF2D(k,ipn,nz,1,ims)];
        rps[FTNREF2D(k,ipn,nz,1,ims)] = rp[FTNREF2D(k,ipn,nz,1,ims)];
    // }
    //!ACC$THREAD(0)
    if (threadIdx.x == 0) {
        wrs[FTNREF2D(0,ipn,nz-0+1,0,ims)] = wr[FTNREF2D(0,ipn,nz-0+1,0,ims)];
    }
    //!ACC$DO VECTOR(1)
        k = threadIdx.x+1;
    // for (k=1;k<=nz;k++) {
        wrs[FTNREF2D(k,ipn,nz-0+1,0,ims)] = wr[FTNREF2D(k,ipn,nz-0+1,0,ims)];
    // }
    // }
    return;
}
//!ACC$REGION END
```

Key Feature: Translated  
CUDA code is human-  
readable!



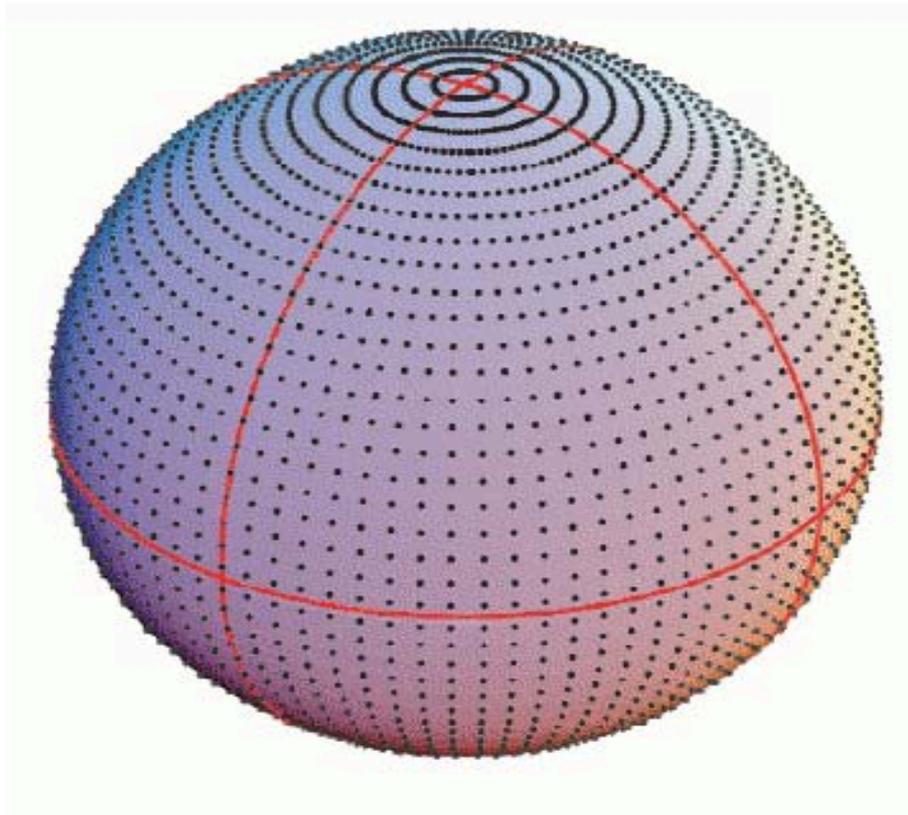
# Outline

- Background
- GPU Hardware
- GPU Programming Environment
- **GPU Parallelization of NIM**

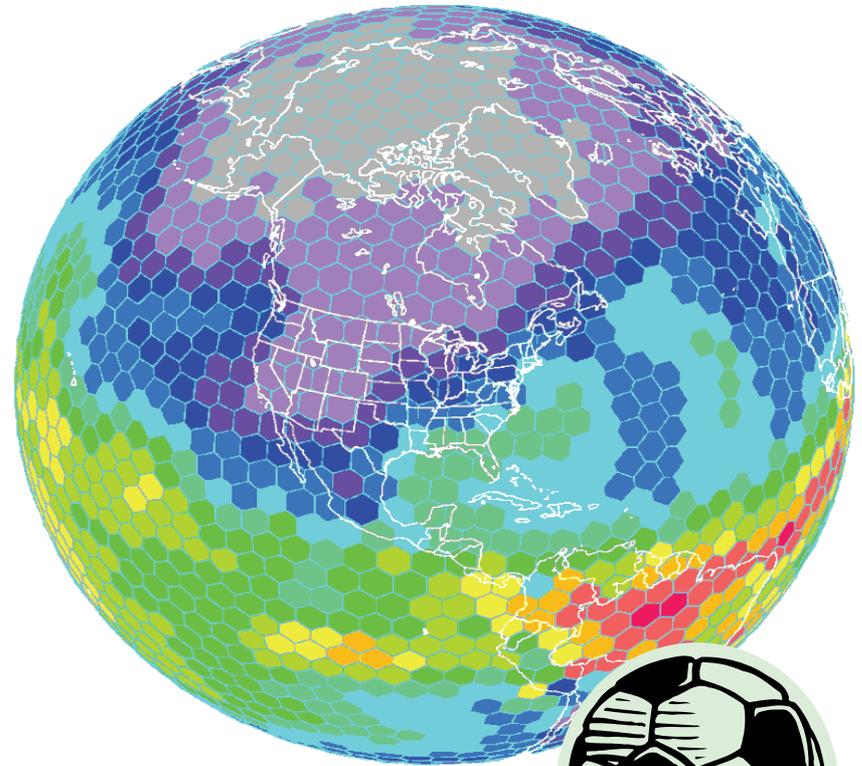


# Non-hydrostatic Icosahedral Model (NIM)

Lat/Lon Model



Icosahedral Model



- Near constant resolution over the globe
- Efficient high resolution simulations

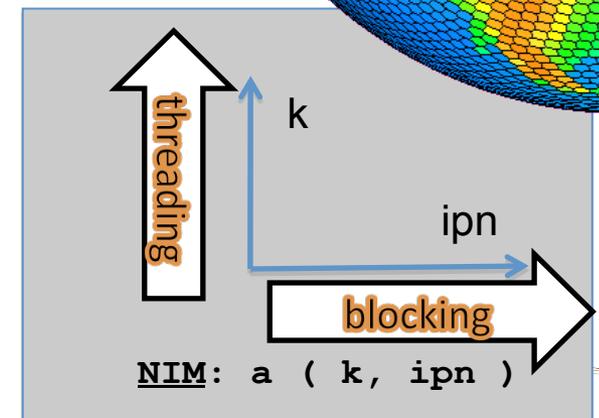
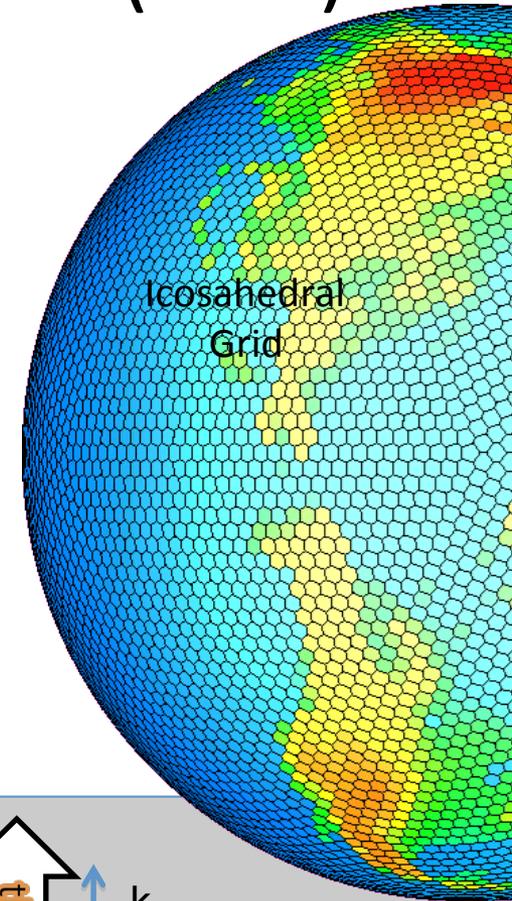
(slide courtesy Dr. Jin Lee)



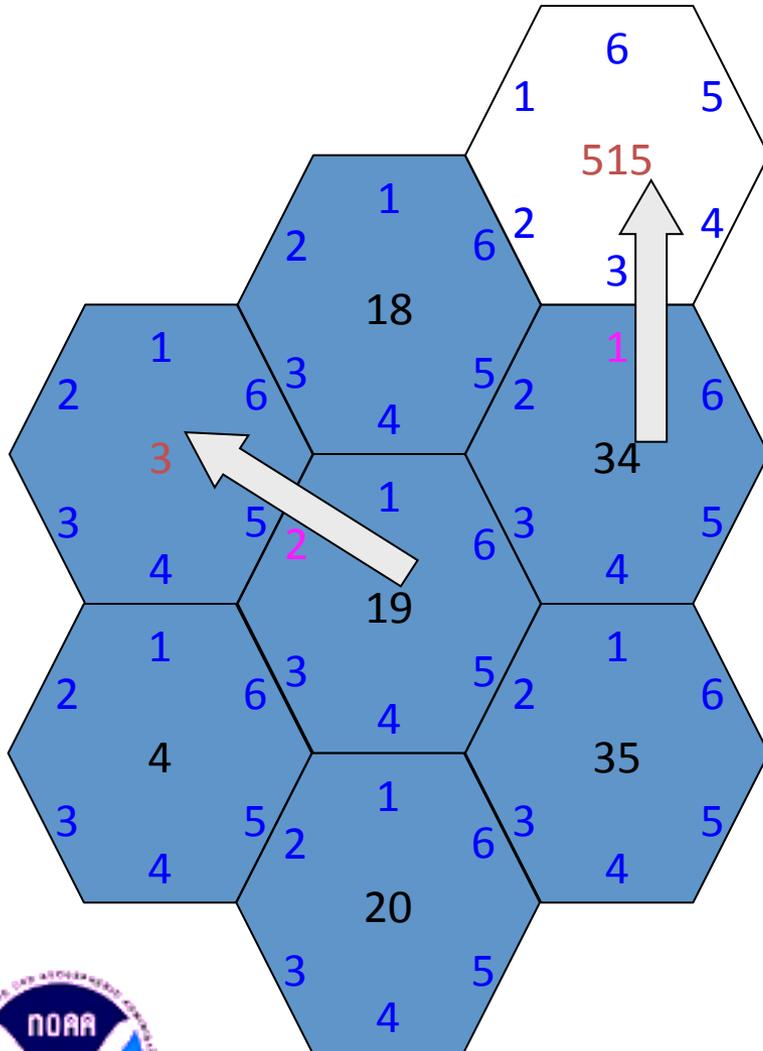
# Non-hydrostatic Icosahedral Model (NIM)

(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
- Plans to run at 4KM resolution on GPUs
- Dynamics is running entirely on GPUs
  - Horizontal data dependencies
  - 2D arrays (vertical, horizontal)
  - GPU threading across the vertical
    - 32, 96 points - - - > 192 points
  - Physics (scientific) integration in progress



# 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%



(slide courtesy Tom Henderson)



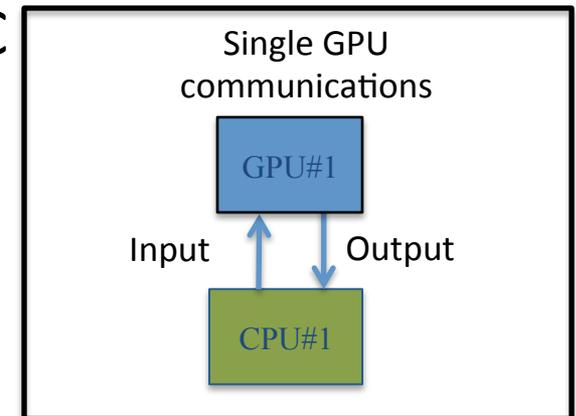
# Resolution nomenclature

!	glvl	Number of grid points	Linear Scale (km)
!	0	12	7,071
!	1	42	3,779
!	2	162	1,174
!	3	642	891
!	4	2562	446
!	5	10,242	223
!	6	40,962	111
!	7	163,842	56
!	8	655,352	28
!	9	2,611,442	14
!	10	10,445,762	7
!	11	41,943,042	3.5
!	12	167,772,162	1.75



# Code Parallelization (2009)

- Developed the Fortran-to-CUDA compiler (F2C-ACC)
  - Commercial compilers were not available in 2008
  - Converts Fortran 90 into C or CUDA-C
  - Some hand tuning was necessary
- Parallelized NIM model dynamics
  - Tesla Chip, Intel Harpertown (2008)
  - Result for a single GPU
  - Communications only needed for I/O

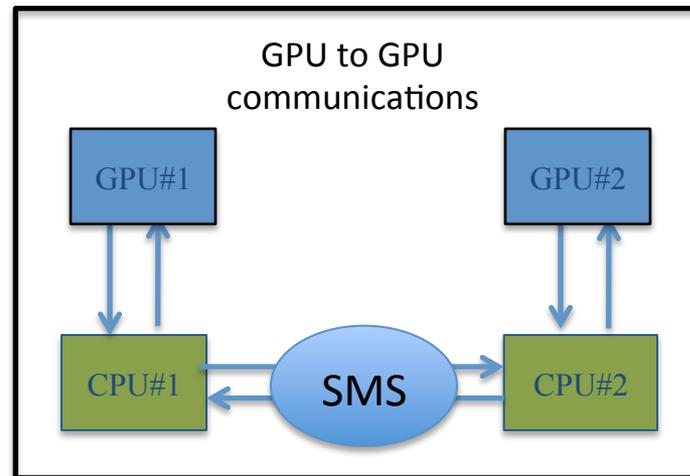


NIM Dynamics (version 160)

Resolution	HorizPts	Harpertown	Tesla	Nehalem	Fermi
G4-480km	2562	2.13	0.079 (26.9)	1.45	0.054 (26.7)
G5-240km	10242	8.81	0.262 (33.5)	5.38	0.205 (26.2)

# Model Parallelization (2010)

- Updated NIM Model Parallelization
  - Active model development
  - Code optimizations on-going
- Evaluate Fortran GPU compilers
  - Use F2C results as benchmark
- Evaluate Fermi
- Run on Multiple GPUs
  - Modified F2C-ACC GPU compiler
  - Uses MPI-based Scalable Modeling System (SMS)
  - Testing on 10 Tesla & 10 Fermi GPUs



# Achieving High Performance

## #1 Optimize the CPU code

- Modifications often help the GPU performance too
  - Performance Profiling identified a matrix solver that accounted for 40 percent of the runtime
    - Called 150,000 per timestep (G4), 3.6 million for 3.5KM
    - Replaced BLAS routine with hand-coded solution resulting in a 3x performance increase for the entire model
    - Loop unrolling improved overall performance by another 40 percent
      - » Developed test kernels to study CPU & GPU performance
- Organize arrays so the inner dimension will be used for thread calculations
  - Improve loads & stores from GPU global memory
- Re-order calculations to improve data reuse



# Achieving High Performance

## #2 Optimize GPU Kernels (major issues)

- Use the performance profiler to identify bottlenecks
  - Occupancy
    - Largely determined by registers and shared memory usage
  - Coalesced Loads and Stores
    - Data alignment with adjacent threads can yield big performance gains
      - » Eg. Threading on “k”, for” a(k,l,j)”
  - Use Shared memory or registers where there is data reuse
    - Must be at least  $> 3$  to have benefit (2 loads & stores are needed to move data from global to local memory)



# Achieving High Performance

## #2 Optimize GPU Kernels (major issues)

- Use the performance profiler to identify bottlenecks
  - Occupancy
    - Largely determined by registers and shared memory usage
  - Coalesced Loads and Stores
    - Data alignment with adjacent threads can yield big performance gains
      - » Eg. Threading on “k”, for” a(k,l,j)”
  - Use Shared memory or registers where there is data reuse
    - Must be at least  $> 3$  to have benefit (2 loads & stores are needed to move data from global to local memory)



# Fortran GPU Compiler Comparison (2009)

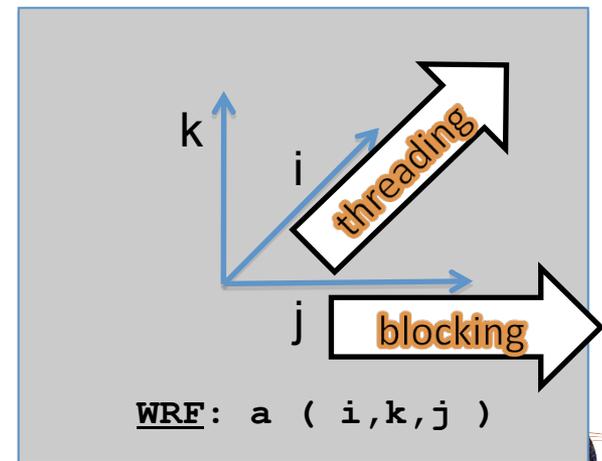
Using NIM G4 - 2562 horizontal points, 96 vertical levels  
 Tesla GPU vs. Intel Harpertown CPU Core

	Harpertown CPU Time	F2C-ACC CUDA-C Tesla GPU Speedup	HMPP Tesla GPU Speedup	PGI Tesla GPU Speedup
vdmints	88.86	34	37	11
vdmintv	37.73	38	38	24
flux	17.97	25	17	24
vdn	12.77	22	17	--
diag	5.13	29	60	52
force	5.34	46	28	12
trisol	8.46	4	6	--
<b>TOTAL</b>	<b>194.25</b>	<b>( 7.48 ) 26</b>	<b>(7.49) 26</b>	



# WRF Physics

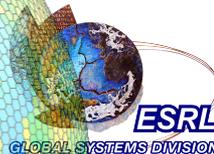
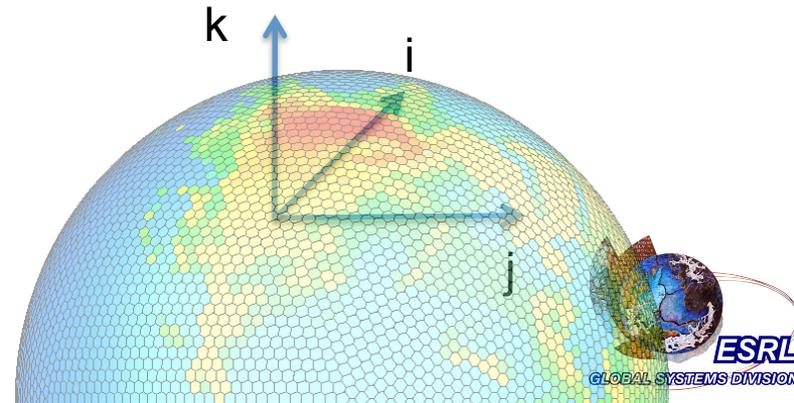
- Community Model used worldwide for more than a decade
  - Significant number of collaborators, contributors
- Used in WRF-ARW, WRF-NMM, WRF-RR, WRF-CHEM, HWRF, etc.
- Traditional cartesian grid
  - 3D arrays (horizontal, vertical, horizontal) == > array3D( i, k, j )
- Designed for CPU architectures
- **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



# Parallelization Factors for NIM

- Code design a dominant factor in performance
  - Weather codes typically have a high memory access to compute ratio
    - Implies lots of accesses, few computations
  - Data alignment led to a 10x improvement
- Data dependencies guide parallelization
  - Dynamics are in the horizontal
    - $a [ \text{vert}, \text{horiz} ]$
  - Physics are in the vertical column
    - $a [ \text{horiz}, \text{vert} ]$
  - Transpose needed to optimize memory accesses

```
lat-lon a ( k, i, j )  
NIM:      a [ k, indx)
```



# Successes

- Parallelization of NIM
  - 25x performance speedup of NIM dynamics
  - Continues to be used for NIM
- 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



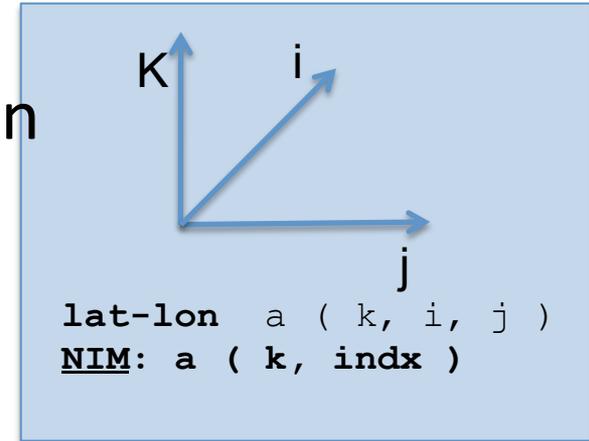
# Challenges: Validating Results

- Results vary depending on the computer architecture
  - CPU, GPU, Intel, IBM, etc
  - If you are running on the same computer, will Fortran and C generate the same exact result?
    - How are intrinsics, constants are calculated? (SP or DP?)
    - Is there a set of compiler options that will give the same answer
- Model numerics
  - Physics is more sensitive than dynamics
  - How do you determine acceptable results?
    - Run the model or routine, compare results after XX number of time steps
      - Bitwise exact result ( all digits )
        - » Fortran, C on the same machine?
        - » Fortran, CUDA on the same machine?
      - “Nearly” bitwise exact ( 5, 6 digits )
        - » C and CUDA on different architectures?
        - » Fortran and CUDA on CPU and GPU?
      - Diverging results? ( 2,3 digits )
        - » Depends ... need scientific guidance



# Final thoughts

- Code design a dominant factor in performance
  - Data alignment led to a 10x improvement
- Data dependencies guide parallelization
- Performance tools are improving
  - Help to determine speedup potential
- Debugging is challenging
  - Potentially millions of threads to manage
- Commercial Fortran GPU compilers are maturing
  - F2C-ACC is largely a language translator, limited analysis
  - As CAPS, PGI mature, they should do more analysis and optimization



# Opportunities at NOAA

- Good science being done
  - Weather, climate, chemistry, land surface, remote sensing, instruments, high performance computing
- Hollings Scholarships
  - Graduate and undergraduate students
    - summer

Email: [mark.w.govett@noaa.gov](mailto:mark.w.govett@noaa.gov)



Quick Links  
mark.w.govett@noaa.gov

CUDA ZONE  
programmers guide  
software  
tools  
links

