Using Fine-Grain Architectures for Weather and Climate Models

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

NOAA Earth System Research Laboratory
Boulder, Colorado

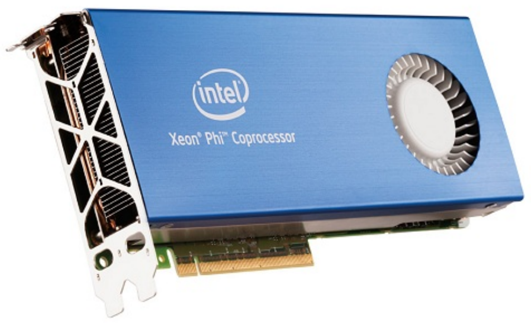
Fine-Grain Computing Hardware

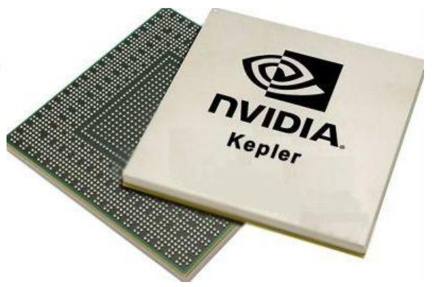
- GPU

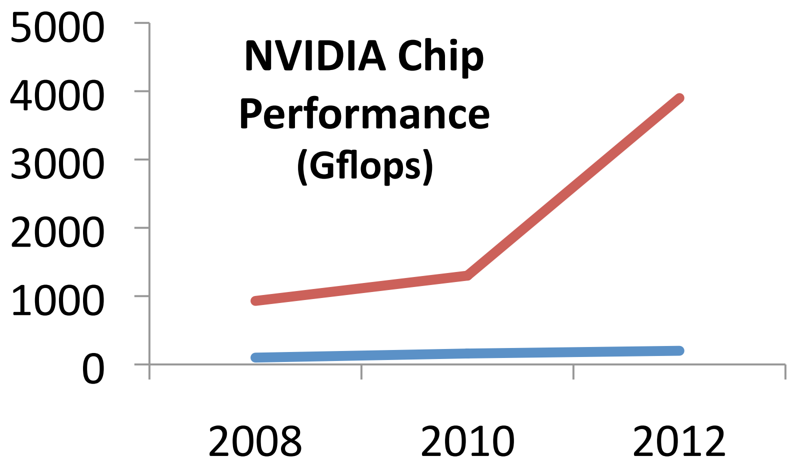
– NVIDIA Kepler	2880 cores	250w	3.9 TFlops
– AMD FireStream	~3000 cores	225w	3.23 TFlop

- Many Integrated Core

– Intel Xeon Phi	61 cores	300w	2.1 Tflops







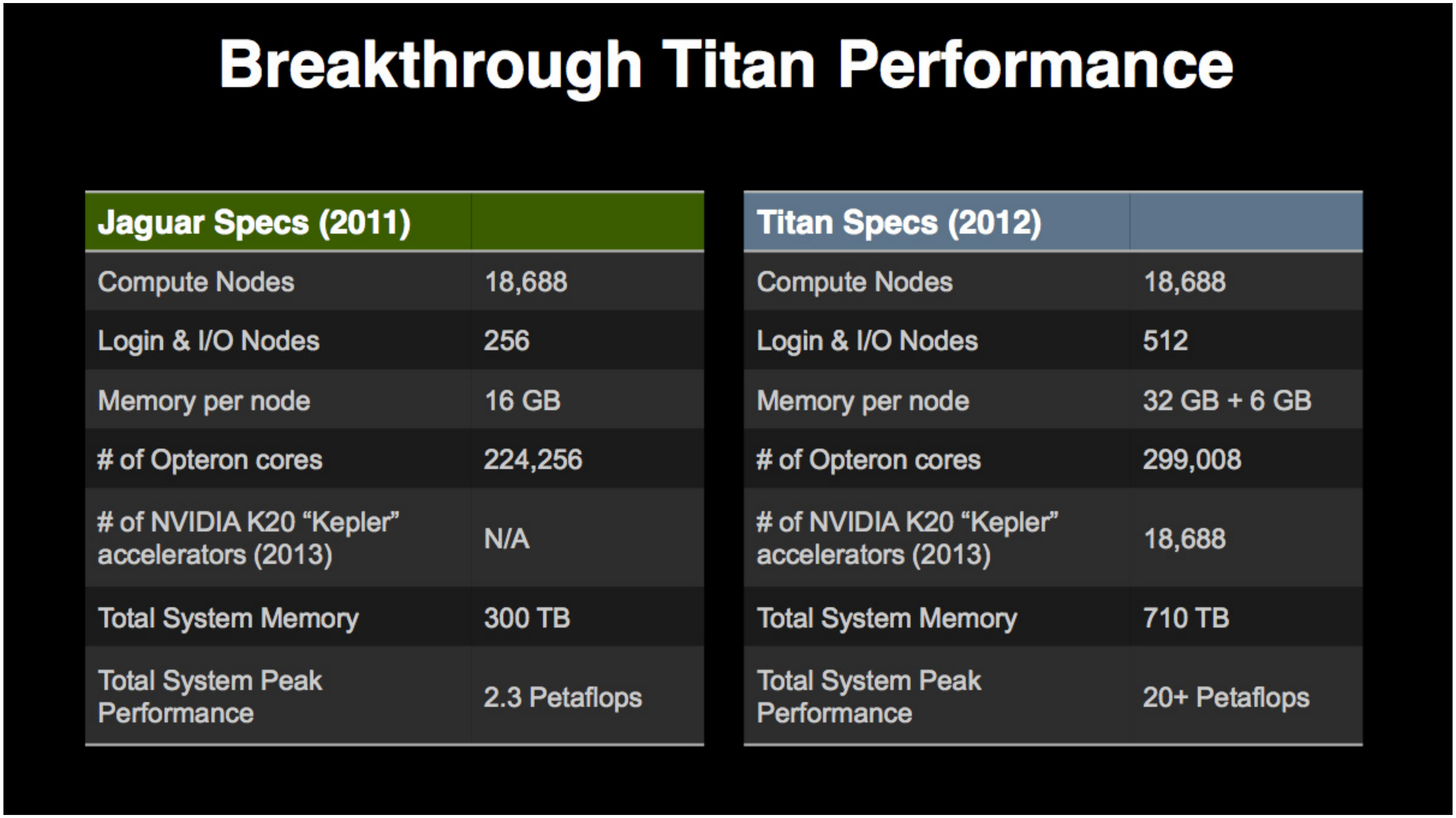
- CPU

– Intel SandyBridge	16 cores	125w	150 GFlops

Fine-Grain Architectures

- Architectures are diverse and continue to evolve
  - Identify more parallelism in the models
- Challenge to maintain single source & performance portability

GPU Chip	Fermi (2010) C2050/70	Kepler (2012) K20x	Intel MIC (2012) Xeon Phi	AMD FireStream 9370
Cores	448	2688	61	1600
- **Clock Speed**	1.15 GHz	0.73 GHz	1.91 GHz	1.2 GHz
- Flops SP	1.0 TF	3.9 TF	2.1 TF	2.6 TF
Memory	4-6 GB	6 GB	8 GB	4 GB
- **Bandwidth**	144 GB/sec	250 GB/sec	~200 GB/sec	147 GB/sec
- Shared/L1	64 KB	64 KB		
Power	238 W	235 W	300 W	225 W
Programing Features	Cache Mem	Dynamic Parallelism	Vector units	openCL



Breakthrough Titan Performance

Jaguar Specs (2011)	
Compute Nodes	18,688
Login & I/O Nodes	256
Memory per node	16 GB
# of Opteron cores	224,256
# of NVIDIA K20 "Kepler" accelerators (2013)	N/A
Total System Memory	300 TB
Total System Peak Performance	2.3 Petaflops

Titan Specs (2012)	
Compute Nodes	18,688
Login & I/O Nodes	512
Memory per node	32 GB + 6 GB
# of Opteron cores	299,008
# of NVIDIA K20 "Kepler" accelerators (2013)	18,688
Total System Memory	710 TB
Total System Peak Performance	20+ Petaflops

- Joint agreement to give NOAA 10M node hours
  - 4 km NIM expected to be tested in 2013/14
    - 4000 GPUs needed

Fine-Grain Computing @ NOAA

- Weather Codes
  - FIM, NIM — in progress
  - NMM-B, HRRR, GFS — planning stages
- Climate Codes
  - NOAA GFDL Cube-Sphere — in progress
- Computing Systems
  - Kepler — ORNL Titan, NOAA gaea
  - MIC — NFS TACC, NASA

Topics

- Maintaining Portability in Model Code
  - Single source, good design
  - Software tools & compilers
  - Validating results
- Performance & Key Optimizations
- Status and Future Work

NVIDIA GPU Architecture

Compute:

- SIMT: 32 cores, called a warp, execute in lock-step on each Streaming MultiProcessor (SM)

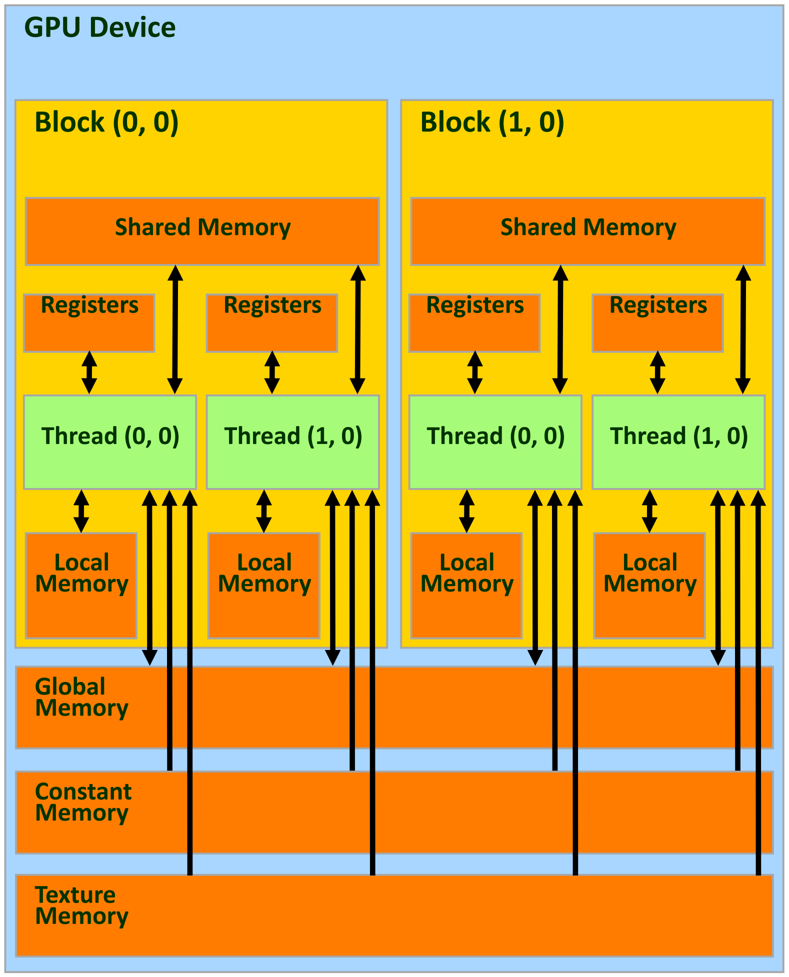
	Fermi	Kepler
SM:	32	14
cores/SM	16	192

Memory:

- 2-4 cycles to access fast memory
  - Registers, cache / shared
- Hundreds of cycles to access slow memory
  - Global, constant, texture

	Fermi	Kepler
Shared:	16/48K	16/48K
Register:	128K	256K

NVIDIA Compute / Memory



Fine-Grain Compilers

- Language–based

– CUDA	NVIDIA	C, C++	+ extensions
– OpenCL	AMD, NVIDIA	C, C++	+ extensions
– CUDA Fortran	PGI	Fortran	+ extensions

- Directive-based Compilers
  - F2C-ACC

– CAPS	OpenACC® DIRECTIVES FOR ACCELERATORS	Xeon - MIC
– PGI		AMD - GPU
– Cray		NVIDIA – GPU
– Intel	OpenMP™	Xeon - MIC

F2C-ACC Compiler

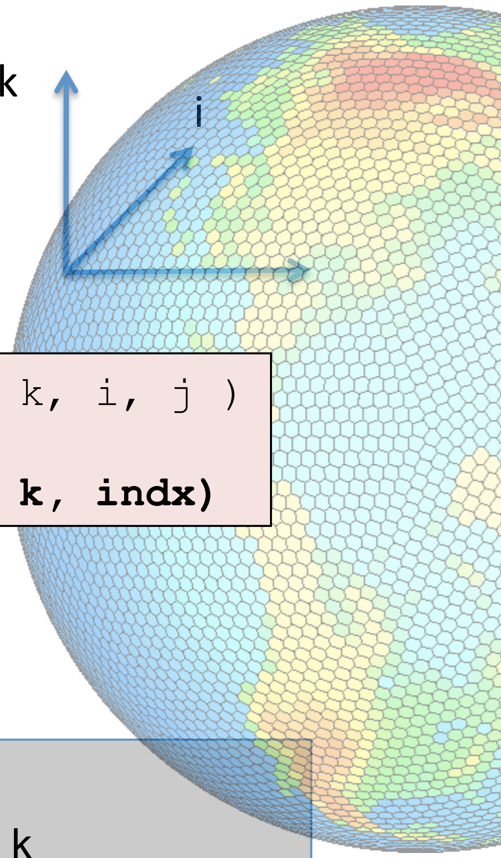
- Developed in 2009, before commercial compilers were available
  - Ten directives for parallelization: !ACC$ <directive_name>
- Used to parallelize NIM & FIM dynamics
  - **Focus on minimizing changes to preserve original code**
  - **Single source to preserve performance portability**
    - Run on CPU, GPU, serial, parallel
- Advanced Features
  - Variable promotion, demotion
  - Shared, local, constant memory
  - Increasing parallelism using “chunking” and “blocking”
- Used to evaluate commercial compilers
  - 2011: evaluated CAPS, PGI (Henderson)
  - 2012: shared stand-alone tests with all the vendors
    - performance, correctness, language support
  - Plan another evaluation in 2013
- 0

Programming for MIC (Intel Phi)

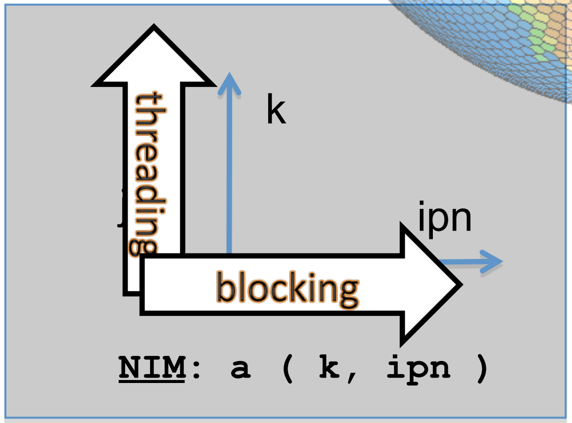
- Relies on OpenMP directives
  - Add an intel compiler flag to invoke MIC
- Two programming modes
  - Native
    - Everything runs on the coprocessor
    - NO code mods required to get it running
    - Can use multiple cores via OpenMP and/or MPI
  - Offload
    - Host offloads part of calculation to coprocessor
    - Compiler directives describe how to move data
    - Similar to GPU in handling of data transfer between 2 address spaces

NIM Parallelization for GPUs

- Uniform, hexagonal-based, icosahedral grid
- Novel indirect addressing scheme permits concise, efficient code
  - MacDonald, Middlecoff, Henderson, et al
  - No performance impact
  - Adopted by NCAR MPAS model
- Designed for fine-grain parallel in 2008
- <u>Dynamics</u>
  - Running on GPU, OpenMP, MIC in progress
- <u>Physics</u>: GFS, YSU
  - GPU, OpenMP, MIC parallelization planned
- Testing at 120, 60, 30km
  - Aqua-Planet to 300 days
- Testing at 120, 60 KM
  - real data runs



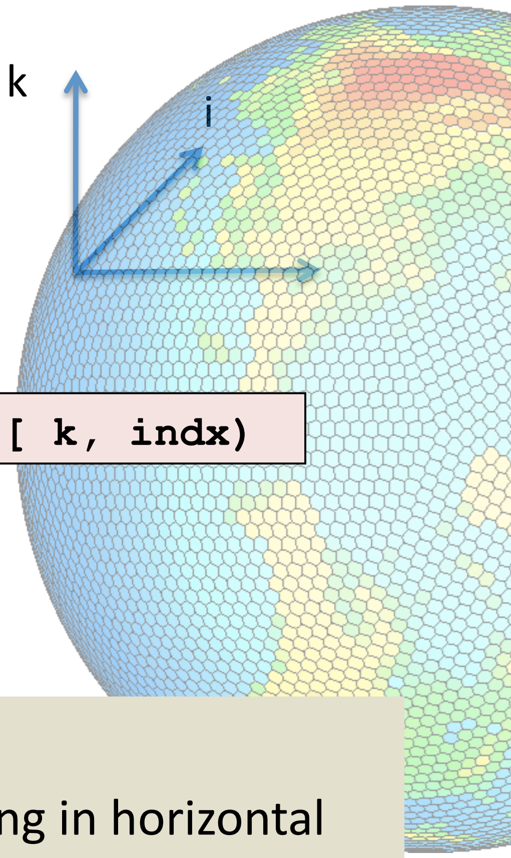
```
lat-lon a ( k, i, j )

NIM:    a [ k, indx)
```



```
NIM: a ( k, ipn )
```

FIM Fine-Grain Parallelization

- Well established code
  - Designed in 2000 for CPUs
  - Near operational status
    - Running daily at 10, 15, 30 KM resolutions
  - Multi-faceted development
    - Ensembles, chemistry, ocean
- Limited ability to change the code
  - Fortran modules
  - Performance improvements
  - No degradation in clarity of code
- No change in results is ideal
  - Otherwise scientists must evaluate



```
FIM:     a [ k, indx)
```

- GPU
  - Blocking in horizontal
  - Threading in vertical
- OpenMP, MIC
  - Threading in horizontal
  - Vector in vertical

Parallelization for GPU & MIC

- Separate CPU routines from accelerator routines
  - Compute on GPU
  - I/O, Comms on CPU
- Parallelize the leaf routines in the call tree
- Use the accelerator style initially
  - Data lives on CPU, copied to GPU to execute each kernel
- Validate output for every step
- Make data resident on device
  - Copy data to GPU
  - Run dynamics on GPU
  - Copy data back to CPU

- FIM Dynamics
  - allocstate
  - dyn_init
  - dyn_run
    - **hybgen**
      - **regrid_1d**
        - **restp_1d, equilb, inflate**
      - **remap_1d**
        - **inflate, plmadv**
    - **hystat**
    - **abstart**
    - **edgvar**
    - **cnuity**
      - **cnuity1, cnuity2, cnuit3, cnuity4, cnuity5**
    - **trcadv**
      - **trcadv1, trcadv2, trcadv3**
    - **momtum**
      - **del4prep, dissip, momtum1**
  - compare_output

Validation of Results

- Prior to CUDA v4.2, the number of digits of accuracy was used to compare FIM / NIM results between the CPU and GPU

```
Variable    Ndifs         RMS (1)              RMSE      max      DIGITS
  rublten     2228  0.1320490309E-03 0.2634E-09  0.3922E-05     5
  rvblten     2204  0.2001348128E-03 0.6318E-09  0.2077E-04     4
   exch_h     3316  0.1670498588E+02 0.8979E-05  0.8379E-05     5
     hpbl        9  0.4522379124E+03 0.2688E-03  0.1532E-04     4
 rqiblten     1082  0.2236843110E-09 0.7502E-17  0.6209E-07     7
```

  - Small differences for 1 timestep can become significant when running a model over many timesteps

- CPU and GPU results are identical (Intel versus NVIDIA)
  - When the fused multiply-add instruction is turned off
    - Eliminates truncation of arithmetic operations
  - Significantly speeds parallelization
  - Exceptions:
    - Use of **power function**, and possibly other intrinsics
      - Calculations must be done on the CPU in these cases right now

Simple OpenMP / GPU Kernel

- Placement of directives are generally in the same location for GPU directives & OMP
  - OMP: Need to have sufficient work to overcome startup overhead
- Data movement is implicit
  - F2C accelerator model assumes data resides on CPU

```
!$OMP PARALLEL DO PRIVATE (k,…) SCHEDULE (runtime)
!ACC$REGION(<nvl>,<ime-ims+1>) BEGIN

!  worka(:,:) = tr3d(:,:,1)       !F2C does not support F90 assignments

!ACC$DO PARALLEL(1)
do ipn=ips,ipe
!ACC$DO VECTOR(1)
  do k=1,nvl
    worka(k,ipn) = tr3d(k,ipn,1)
  end do
end do
!ACC$REGION END
!OMP END PARALLEL DO
```

OpenACC GPU Kernel

- Unclear how OpenACC compilers handles data movement
  - Explicitly listing each variable for each region can be tedious particularly if the region is big and complicated
- Unclear how if OpenACC compilers support Fortran 90 syntax
  - Compiler analysis could determine parallelism implicitly

```
!$ACC DATA [copyin] [copyout]
!$ACC PARALLEL  [ gang ] [ worker ] [ vector ]

!  worka(:,:) = tr3d(:,:,1)        !unclear if openACC can handle this

!$ACC LOOP [gang] [worker] [vector]
do ipn=ips,ipe
!$ACC LOOP [gang] [worker] [vector]
do k=1,nvl
    worka(k,ipn) = tr3d(k,ipn,1)
  end do
end do
!$ACC END PARALLEL
```

FIM Performance – CPU, GPU & MIC (2012)

- GPU timings used F2C-ACC compiler
  - Commercial compilers performance between Global and Optimized
- Intel Xeon Phi (MIC) – SE10x Pre-production chip
  - 61 cores, 1.091 GHz, 8GB memory
  - OpenMP + MIC extensions used for parallelization
- Kepler K20x results are from the Fermi F2C-ACC optimized code

FIM Dynamics Routines	**Fermi GPU F2C-ACC 1 socket**	**Fermi GPU Optimized 1 socket**	**Intel CPU SandyBridge 1 & (2) socket**	**Intel Xeon Phi – KNC 1 socket**	**Kepler GPU EARLY RESULTS**
trcadv	2.07	1.28	2.10 (1.80)	1.59	0.66
cnuity	5.20	1.04	1.13 (0.76)	0.74	1.15
momtum	0.57	0.41	0.71 (0.60)	2.03	0.41
hybgen		4.13	4.09 (2.06)	3.40	3.43
TOTAL	-	8.01	8.87 (5.97)	9.89	6.60

Variable Promotion for Correctness

Example: NIM vdmintv subroutine (nz=32)

F2C V4: - promote variables using GPU global memory

```
real :: rhsu(nz,nob), rhsv(nz,nob), tgtu(nz,npp), tgtv(nz,npp)

!ACC$REGION (<nz>,<(ipe-ips+1)>,                                    &
!ACC$> <rhsu,rhsv,tgtu,tgtv:none,global,promote(1:block)> ) BEGIN
!ACC$DO PARALLEL(1)
do ipn=ips,ipe
!ACC$DO VECTOR(1,1:nz-1)
  do k=1,nz-1
    rhsu(k,1) = cs(1,ipn)*u(k  ,ipp1)+sn(1,ipn)*v(k  ,ipp1) - u(k,ipn)
    rhsu(k,2) = ...
         < Similar calculations on rhsv, tgtu, tgtv >
  enddo
  call solver( ..., rhsu, rhsv, ...)
enddo
!ACC$REGION END
```

Performance: run-time w/ global memory: 12.51 ms

nvcc will use cache by default

Optimization: Shared Memory

Example: NIM vdmintv subroutine (nz=32)

F2C V4: - use GPU shared memory for rhsu,rhsv,tgtu,tgtv

```
real :: rhsu(nz,nob), rhsv(nz,nob), tgtu(nz,npp), tgtv(nz,npp)
!ACC$DATA(<rhsu,rhsv,tgtu,tgtv:none,shared>)     !declaration required
!ACC$REGION (<nz>,<(ipe-ips+1)>,                                    &
!ACC$> <rhsu,rhsv,tgtu,tgtv:none,shared> ) BEGIN
!ACC$DO PARALLEL(1)
do ipn=ips,ipe
!ACC$DO VECTOR(1,1:nz-1)
  do k=1,nz-1
    rhsu(k,1) = cs(1,ipn)*u(k  ,ipp1)+sn(1,ipn)*v(k  ,ipp1) - u(k,ipn)
    rhsu(k,2) = ...
         < Similar calculations on rhsv, tgtu, tgtv >
  enddo
  call solver( ..., rhsu, rhsv, ...)
enddo
!ACC$REGION END
```

Performance: run-time w/ shared memory: 7.30 ms

1.7x speedup over global memory w/ cache

Optimization: Variable Demotion

Example: FIM trcadv subroutine

F2C V4: - Demote variables + shared, local or register memory

```
!ACC$REGION(<nvl:block=2>,<ipe-ips+1>,                              &
!ACC$>  <s_plus,s_mnus:none,local,demote(1)>) BEGIN
!ACC$DO PARALLEL(1)
    do ipn=ips,ipe
!ACC$DO VECTOR(1)
      do k=1,nvl
        s_plus(k) = 0.
        s_mnus(k) = 0.
      end do
      do edg=1,nprox(ipn)
!ACC$DO VECTOR(1)
        do k=1,nvl
          s_plus(k) = s_plus(k) - min(0., antiflx(k,edg,ipn))
          s_mnus(k) = s_mnus(k) + max(0., antiflx(k,edg,ipn))
        end do
      end do
```

Performance: 1.8x faster than global memory / cache

NIM Performance (2013)

- 10K horizontal points, 96 vertical levels

NIM	Opteron	Westmere	SandyBridge	Fermi	K20x
1 node	75.4	86.8	31.3	25.0	20.7

- Very early weak scaling results
  - Opteron, K20x from Gaea-GPU + Gemini
  - SandyBridge + Infiniband
  - Communications issue noted on Gaea
    - Contradicts Jaguar test showing Gemini was faster

Number of Nodes	Opteron + Gemini Total (comms)	SB + Infiniband Total (comms)	K20x + Gemini Total (Comms)
4 (120 km)	79.8 (10.6)	34.0 (5.9)	46 (21.0) = 25.0
16 (60 km)	95.9 (21.5)	43.1 (11.8)	68.7 (37.1) = 31.6
64 (30 km)	83.7 (28.3)	45.9 (14.4)	60.1 (29.6) = 30.5

Conclusion & Next Steps

- Demonstrated single source, performance portability for GPU, MIC, CPU
- Bit reproducibility between GPU & Intel CPU
- Early performance on K20x is encouraging
- Next Steps
  - NIM
    - MIC parallelization of dynamics
    - MIC & GPU parallelization of physics
    - Optimize scaling from 100s to 1000s of GPUs after Titan becomes available in May
  - FIM
    - MIC & GPU parallelization of physics
    - Run 60 member ensembles on Titan