An Update on Fine-Grain Computing Activities at NOAA's Earth System Research Laboratory

Mark Govett

Tom Henderson, Jim Rosinksi

Jacques Middlecoff, and Paul Madden

Future of Model Development

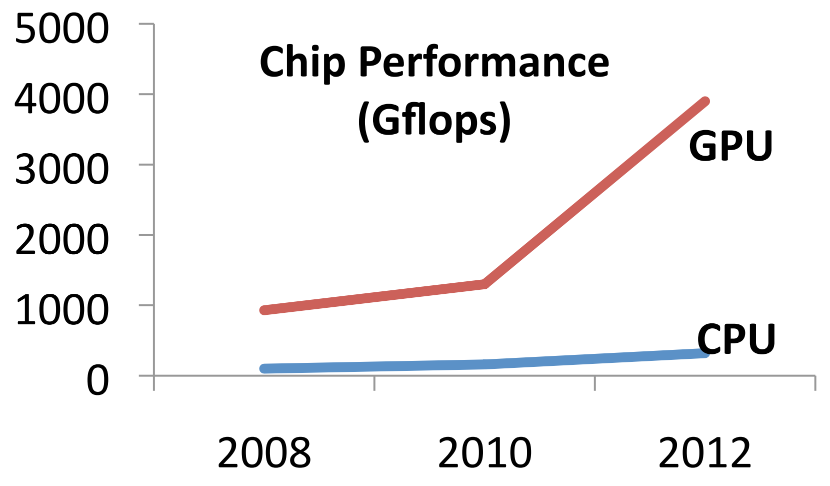
- Increasingly dependent on computing
  - Higher resolution, more ensembles, advanced data assimilation

Model	Resolution	Frequency	CPU	60 members	GPU
HRRR - CONUS	3 KM	hourly	1,000	60,000	
HRRR - CONUS	1.5 KM	hourly	8,000	480,000	
NIM - Global	4.0 KM	6 hours	200,000	---	4000

Energy Efficient Super-Computing

I. GPU: NVIDIA
- Fermi (2010)
  - **512 cores 236W 1.3 Tflops**
- Kepler (Q4 2012)
  - **3072 cores 225W 3.9 Tflops**

II. Intel Many Integrated Core (MIC)
- 32-64 cores



Application Performance

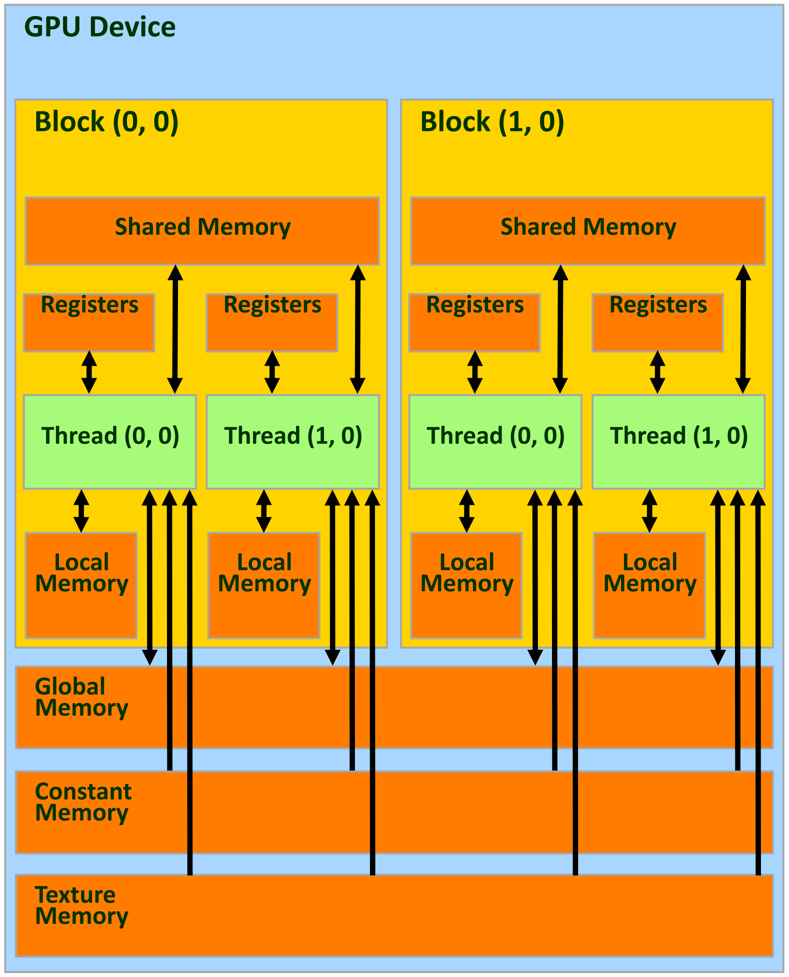
- Efficient use of memory is critical to good performance
  - ~10 cycles to access registers
  - Slightly more to access shared memory
  - Hundreds of cycles to access global memory

Memory	Tesla	Fermi
Registers/SM	1K	4K
Shared/SM	16K	64K
Global	1-2GB	4-6GB

Code may require changes to use memory efficiently

- Re-organize arrays
- Restructure calculations

GPU Multi-layer Memory



Overview

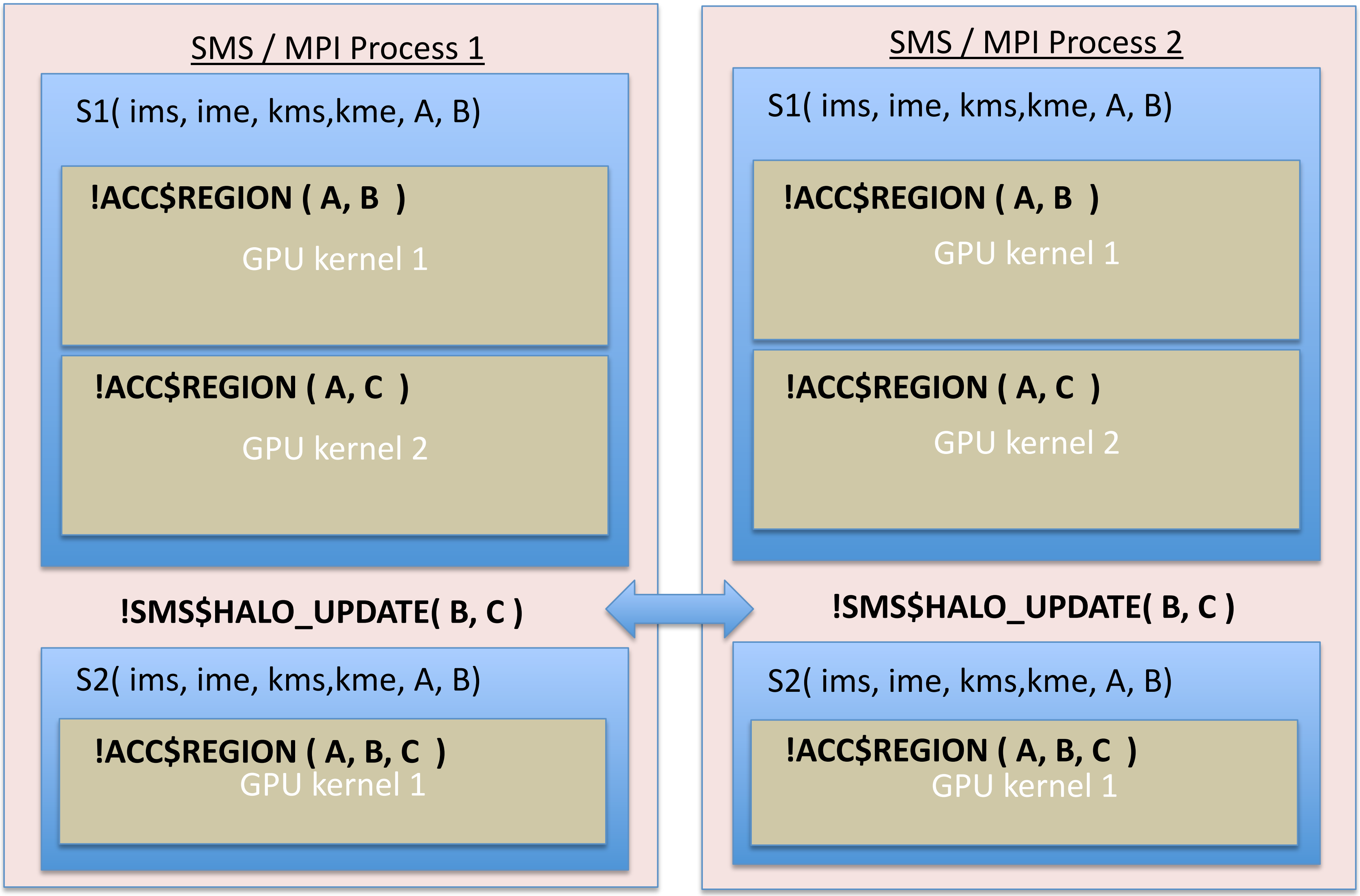
- Fine-Grain Parallelization
  - NIM
    - Designed for Fine-grain architectures
    - Significant developments in the last 6 months
  - FIM
    - Well established code, operationally ready
    - Legacy Code: Designed for CPU architectures
- Performance and optimization
    - NIM: CPU, GPU
    - FIM: CPU GPU, MIC

NIM & FIM Fine-Grain Parallelization

- Approach
  - **Single Fortran source code**
    - GPU, MIC, CPU, serial, parallel
  - Directives used for parallelization
- Directive-based Compilers
  - GPU: F2C-ACC, CAPS, PGI, CRAY
  - MIC: OMP + extensions
  - SMS: Scalable Modeling System relies on MPI
    - Developed in ESRL, used for 2 decades
- Code optimization and comparisons
  - Some architecture specific optimizations explored for CPU and GPU
  - Mostly the codes are identical, results are bit-for bit exact

OpenACC Standard
!$ACC directive

Directive-based Parallelization for FIM and NIM



NIM GPU Development Timeline

- 2010/2011: dynamics, no physics
  - 240 KM resolution test case (10,242 points)
  - 4.5x speedup (6 core Westmere vs. single Fermi GPU)
  - waiting for science to progress
- 2012: dynamics + physics
  - Runs at 30 KM resolution with YSU or GFS physics
  - F2C-ACC parallelization of dynamics took two weeks
  - Multi-GPU runs made on small cluster
    - DOE TitanDev no longer available
- 2013:
  - NIM runs @ 3.5KM resolution on 4000 GPUs
    - INCITE proposal for DOE Titan resources

F2C-ACC Compiler

- Directive-based Compiler !ACC$<directive>
- Generates CUDA or C
- Developed in 2009 to speed code conversion of NIM
- Single source code that runs on CPU & GPU
  - Important for code developers (scientists)
  - Reduces development time
  - Allows for direct performance comparisons between CPU, GPU, MIC
- Used to parallelize
  - NIM, FIM dynamics and WRF / YSU physics
- Working with the GPU compiler vendors
  - CAPS, PGI, CRAY

Recent F2C-ACC Improvements

- Ease of Use
  - Automatic generation of data movement
- Bit-for-bit correctness with CPU
  - Improvements to CUDA, F2C-ACC compilers
  - Variable Promotion: Add a thread or block dimension
- Performance
  - Variable Demotion: Remove array dimensions
  - Control of global, local, shared and register memory
  - Options for increasing thread level parallelism

Bit-for-bit Correctness

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

```
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
- NVCC V4.2 option: –fmad=false
  – No truncation of operation to 32 bits
  – FIM, NIM runs are bitwise exact compared to the CPU
    - Sped up parallelization of the codes

F2C-ACC Code Example

• ACC$REGION	- defines the kernel, number of threads & blocks
• ACC$DO PARALLEL	- indicates BLOCK level parallelism
• ACC$DO THREAD	- indicates THREAD level parallelism

```
!ACC$REGION(<nvl>,<ime-ims+1>) BEGIN
!ACC$DO PARALLEL(1)
do ipn=ims,ime
  do edgcount=1,nedge(ipn)          ! loop through edges
    edg = permedge(edgcount,ipn)
!ACC$DO VECTOR(1)
    do k=1,nvl
      vnorm(k,edg,ipn) = sidevec_e(2,edg,ipn)*                  &
          u_edg(k,edg,ipn) - sidevec_e(1,edg,ipn)*              &
          v_edg(k,edg,ipn)
    end do
  end do
end do
!ACC$REGION END
```

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)

!ACC$REGION (<nz>,<nip>,                                    &
!ACC$> <rhsu, rhsv:none,global,promote(1:block)> ) BEGIN
!ACC$DO PARALLEL(1)
do ipn=1, nip
!ACC$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 >
  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)

!ACC$REGION (<nz>,<(ipe-ips+1)>,                                         &
!ACC$>                      <rhsu,rhsv: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 >
  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

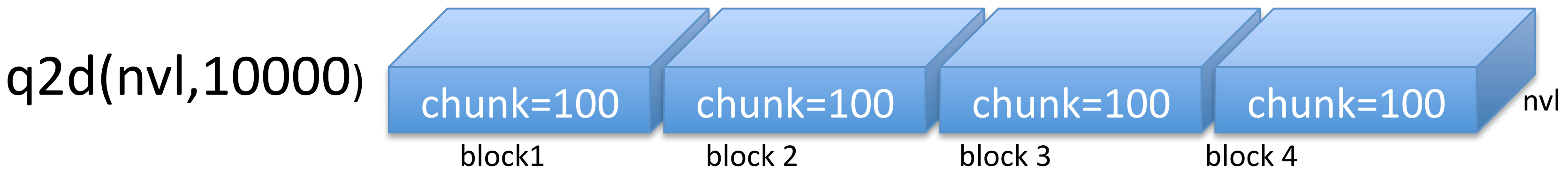
F2C V4: - Demote variables to use register memory

```
!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
```

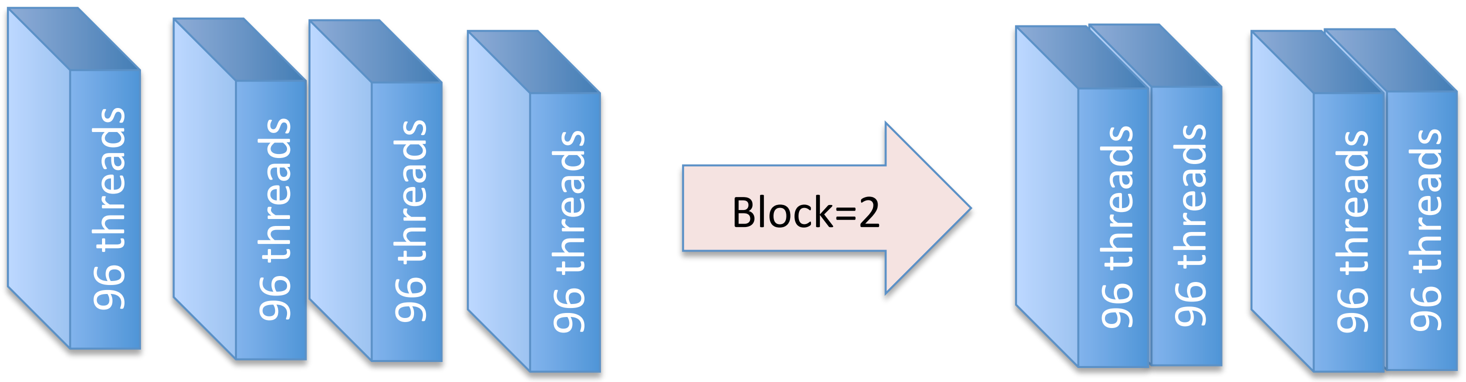
Performance: 1.8x faster than global memory / cache

Optimizations to increase parallelism

- Chunking
  - Only have one parallel dimension, need 2 for GPU
  - Assign threads and blocks to the same dimension
  - !ACC$REGION(<100:chunk>,<(ime-ims+1)/100>) BEGIN



- Blocking
  - Increasing the number of threads per block
  - !ACC$REGION(<32: block=2>,<ime-ims+1>) BEGIN



Standalone Tests: Speedup

FIM: 1 GPU versus 1 CPU socket (6 cores) with OpenMP

Routine	GPU - F2C 1 socket GLOBAL MEMORY	GPU – F2C 1 socket SHARED MEMORY	GPU – F2C 1 socket Shared + Demotion	GPU – F2C 1 socket BLOCK or CHUNKING	GPU – F2C 1 socket BEST	CPU Westmere 1 socket runtime
Trcadv - 64	5.3	6.5	6.8	7.6	9.1	11.66
cnuity - 64	0.9	3.7		3.6	4.3	4.48
Momtum - 64	8.2		9.0		11.4	4.68

FIM: 1 Socket GPU versus 1 CPU CORE

Routine	1 socket GLOBAL MEMORY	1 socket SHARED MEMORY	1 socket Shared + Demotion	1 socket BLOCK or CHUNKING	1 socket BEST	1 core runtime
Vdmintv - 32	4.7	9.2		13.6	16.9	58.7*
wrf_pbl - 70	0.7			12.8	12.8	39.0*

- Explicit use of GPU memories was always better than GLOBAL memory with cache.

Software Challenges

- Architectures are diverse and continue to evolve
  - Optimizations can differ depending on the chip
- Retain performance portability with a single source code

GPU Chip	Tesla (2008) C1060	Fermi (2010) C2050/70	Fermi (2011) C2090	Kepler (2012) K10	Intel MIC
Cores	240	448	512	2 x 1536	> 50
- Clock Speed	1.15 GHz	1.15 GHz	1.3 GHz	0.74 GHz	[redacted]
- Flops SP	0.9 TF	1.0 TF	1.3 TF	4.58 TF	[redacted]
Memory	2 GB	3-6 GB	6 GB	8 GB	[redacted]
- Bandwidth	102 GB/sec	144 GB/sec	177 GB/sec	320 GB/sec	[redacted]
- Shared/L1	16K	64 KB	64 KB	64 KB	[redacted]
Power	188 W	238 W	225 W	225 W	[redacted]
New Programming Features	CUDA	Cache, ECC Memory		Dynamic Parallelism	[redacted]

Summary

- F2C-ACC has been essential for FIM, NIM
- Majority of time is spent preparing code
  - Similar code changes for MIC or GPU
- GPU parallelization is quite easy
- Debugging is harder
  - Data movement between CPU & GPU memories
  - Parallelism & synchronization
- Bit-for-bit accuracy between CPU & GPU speeds parallelization