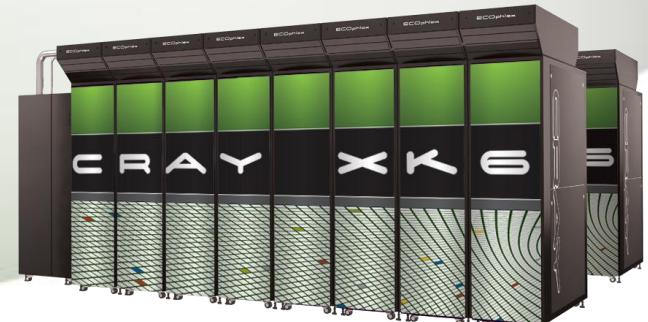


# WRF Experiments on GPU Accelerators using OpenACC

Pete Johnsen  
Jim Schwarzmeier

pjj@cray.com  
Cray, Inc.



Programming weather, climate, and earth-system models  
on heterogeneous multi-core platforms

September 12-13, 2012 at the National Center for Atmospheric Research in Boulder, Colorado

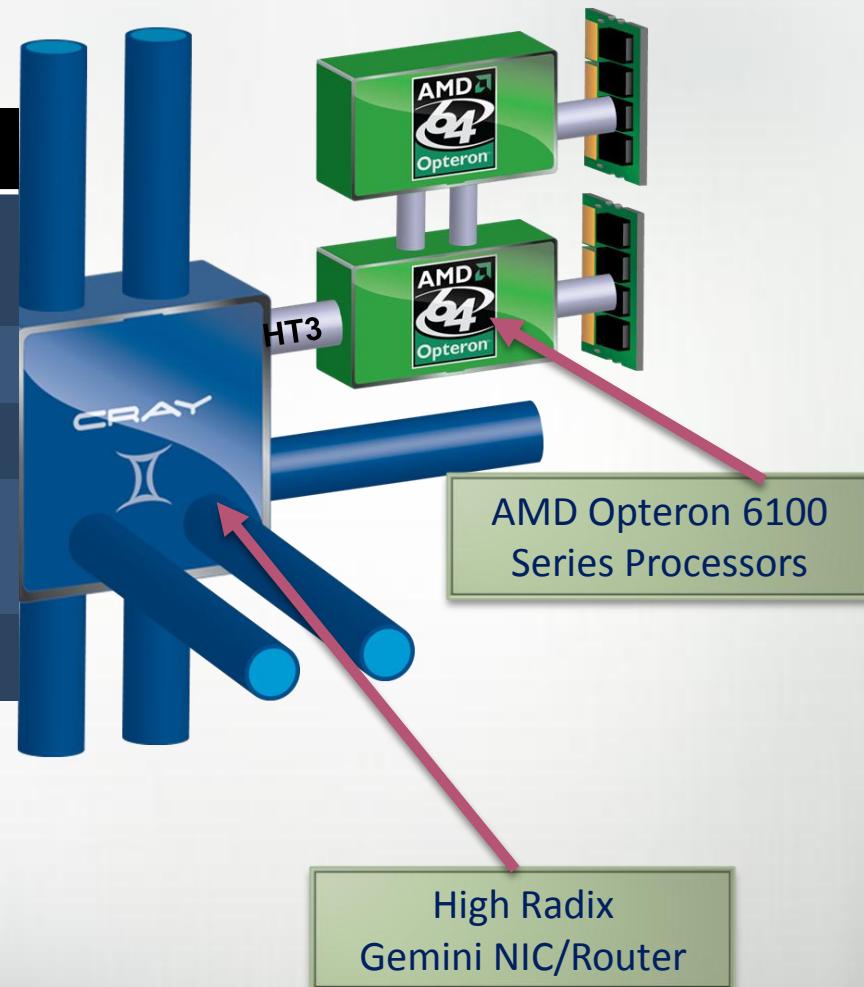
# Topics

- First, a look at using OpenACC on WRF subroutine
  - advance\_w dynamics routine
- Second, an estimate of WRF multi-node performance on Cray XK6 with GPU accelerators
  - Based on performance of WRF kernels, what can we expect if entire WRF ARW solver is ported to Cray XK6 using GPUs
  - Include dynamics and physics routines as well as halo exchanges

# Cray XE6 Compute Nodes

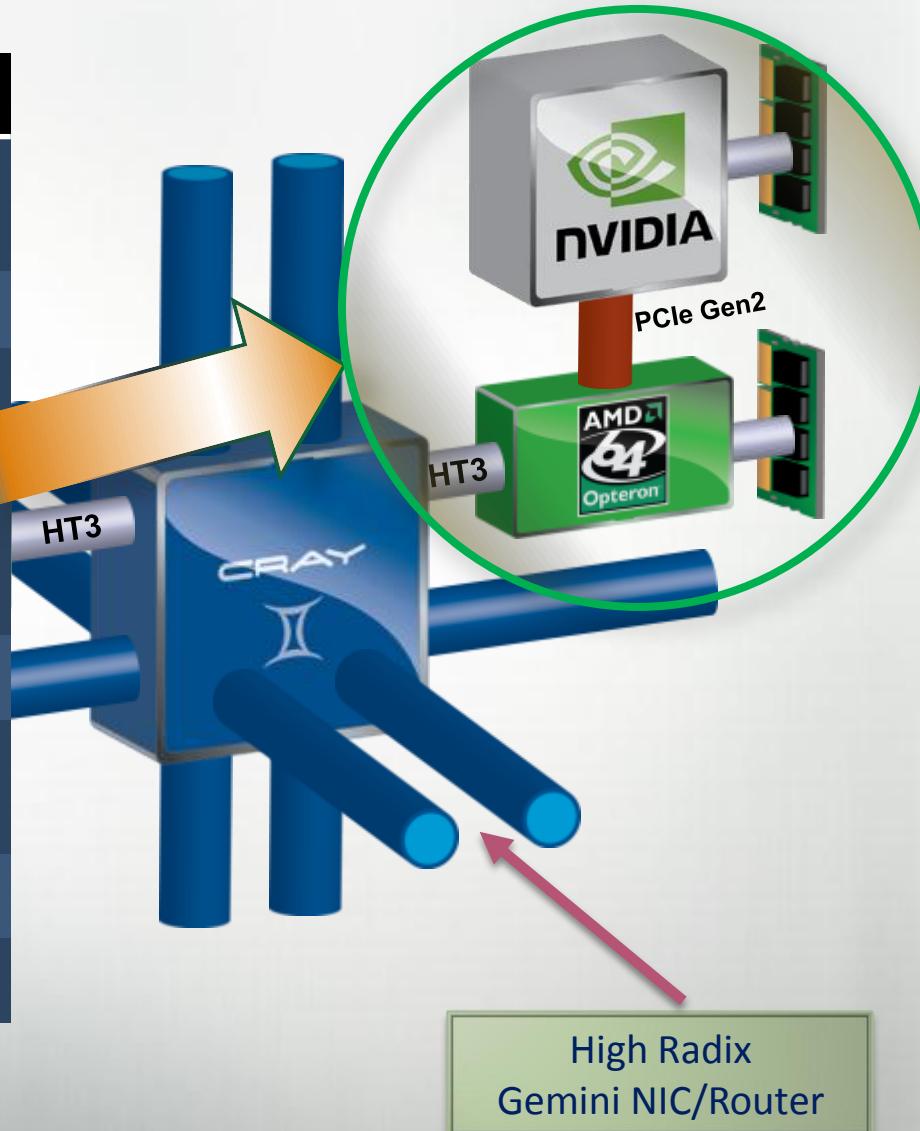
## XE6 Compute Node Characteristics

Host Processors	AMD Series 6200 (Interlagos)
Host Processors	147 Gflops
Host Processor Cores	32 (16 per socket)
Host Memory	32 or 64GB 1600 MHz DDR3
Gemini High Speed Interconnect	



# Cray XK6 Compute Node

XK6 Compute Node Characteristics	
Host Processor	AMD Series 6200 (Interlagos)
Host Processor	147 Gflops
Tesla X2090 Cores	512
Tesla X2090 Perf.	665 Gflops DP
Host Memory	16 or 32GB 1600 MHz DDR3
Host Processor Cores	16 (1 socket)
Tesla X2090 Memory	6GB GDDR5 capacity 170 GB/sec No ECC
Gemini High Speed Interconnect	
Upgradeable to KEPLER many-core processor	



# Using OpenACC – advance\_w dynamics routine

- Advances vertical velocity and geopotential
- Kernel extracted from WRF solver to speed tests
- Measure performance of 128x48x128 grid
- Compare 1 AMD Interlagos socket (16 cores) with 1 Fermi GPU
  - X86 version uses OpenMP threads and WRF tiling
  - GPU version uses OpenACC gangs (thread blocks) and vector loops (threads)

OpenMP Tile Loop (host)

```
!$OMP PARALLEL DO    &
!$OMP PRIVATE ( tile ) 
do tile=1,numtiles

call advance_w( w_2, rw_tend, ww, u_2, v_2,          &
               mu_2, mut, muave, muts,           &
               t_2save, t_2, t_save,           &
               ph_2, ph_save, phb, ph_tend,   &
               ht, c2a, cqw, alt, alb,       &
               a, alpha, gamma,             &
               rdx, rdy, dts_rk, t0, epssm,   &
               dnw, fnm, fnp, rdnw, rdn,     &
               cf1, cf2, cf3, msft,          &
               ids,ide, jds,jde, kds,kde,   &
               ims,ime, jms,jme, kms,kme,   &
               i_start(tile), i_end(tile), &
               j_start(tile), j_end(tile), &
               kds      , kde            )

enddo
!$OMP END PARALLEL DO
```

# advance\_w and OpenACC

OpenACC Parallelism defined inside advance\_w subroutine

Define GPU kernel and  
partition into thread blocks on J  
dimension

```
138. G -----< !$ACC PARALLEL LOOP PRIVATE(mut_inv, msft_inv, wdn) FIRSTPRIVATE(rhs)
139. G g-----< j_loop_w: DO j = j_start, j_end
140. G g g-----< DO i=i_start, i_end
141. G g g           mut_inv(i) = 1./mut(i,j)
142. G g g           msft_inv(i) = 1./msft(i,j)
143. G g g-----> ENDDO
144. G g
145. G g i-----< DO k=1, k_end
146. G g i ig----< DO i=i_start, i_end
147. G g i ig           t_2ave(i,k,j)=.5*((1.+epssm)*t_2(i,k,j)      &
148. G g i ig           + (1.-epssm)*t_2ave(i,k,j))
149. G g i ig           t_2ave(i,k,j)=(t_2ave(i,k,j)-mul(i,j)*t_1(i,k,j)) &
150. G g i ig           / (muts(i,j)*(t0+t_1(i,k,j)))
151. G g i ig----> ENDDO
152. G g i-----> ENDDO
153. G g
154. G g b-----< DO k=2,k_end+1
155. G g b gb----< DO i=i_start, i_end
156. G g b gb           wdn(i,k)=.5*(ww(i,k,j)+ww(i,k-1,j))*rdnw(k-1)    &
157. G g b gb           *(ph_1(i,k,j)-ph_1(i,k-1,j)+phb(i,k,j)-phb(i,k-1,j))
158. G g b gb           rhs(i,k) = dts*(ph_tend(i,k,j) + .5*g*(1.-epssm)*w(i,k,j))
159. G g b gb----> ENDDO
160. G g b-----> ENDDO
```

# advance\_w and OpenACC

OpenACC Parallelism defined inside advance\_w subroutine

```
138. G -----< !$ACC PARALLEL LOOP PRIVATE(mut_inv, msft_inv, wdn) FIRSTPRIVATE(rhs)
139. G g-----< j_loop_w: DO j = j_start, j_end
140. G g g-----< DO i=i_start, i_end
141. G g g           mut_inv(i) = 1./mut(i,j)
142. G g g           msft_inv(i) = 1./msft(i,j)
143. G g g-----> ENDDO
144. G g
145. G g i-----< DO k=1, k_end
146. G g i ig----< DO i=i_start, i_end
147. G g i ig           t_2ave(i,k,j)=.5*((1.+epssm)*t_2(i,k,j)      &
148. G g i ig           + (1.-epssm)*t_2ave(i,k,j))
149. G g i ig           t_2ave(i,k,j)=(t_2ave(i,k,j)-mul(i,j)*t_1(i,k,j)) &
150. G g i ig           / (muts(i,j)*(t0+t_1(i,k,j)))
151. G g i ig----> ENDDO
152. G g i-----> ENDDO
153. G g
154. G g b-----< DO k=2,k_end+1
155. G g b gb----< DO i=i_start, i_end
156. G g b gb           wdn(i,k)=.5*(ww(i,k,j)+ww(i,k-1,j))*rdnw(k-1)    &
157. G g b gb           *(ph_1(i,k,j)-ph_1(i,k-1,j)+phb(i,k,j)-phb(i,k-1,j))
158. G g b gb           rhs(i,k) = dts*(ph_tend(i,k,j) + .5*g*(1.-epssm)*w(i,k,j))
159. G g b gb----> ENDDO
160. G g b-----> ENDDO
```

Inner loops (I) distributed across threads within block

# advance\_w and OpenACC

OpenACC Parallelism defined inside advance\_w subroutine

```
138. G -----< !$ACC PARALLEL LOOP PRIVATE(mut_inv, msft_inv)
139. G g-----< j_loop_w: DO j = j_start, j_end
140. G g g-----<     DO i=i_start, i_end
141. G g g           mut_inv(i) = 1./mut(i,j)
142. G g g           msft_inv(i) = 1./msft(i,j)
143. G g g-----> ENDDO
144. G g
145. G g i-----<     DO k=1, k_end
146. G g i ig----<       DO i=i_start, i_end
147. G g i ig           t_2ave(i,k,j)=.5*((1.+epssm)*t_2(i,k,j)      &
148. G g i ig           + (1.-epssm)*t_2ave(i,k,j))
149. G g i ig           t_2ave(i,k,j)=(t_2ave(i,k,j)-mul(i,j)*t_1(i,k,j)) &
150. G g i ig           / (muts(i,j)*(t0+t_1(i,k,j)))
151. G g i ig----> ENDDO
152. G g i-----> ENDDO
153. G g
154. G g b-----<     DO k=2,k_end+1
155. G g b gb----<       DO i=i_start, i_end
156. G g b gb           wdn(i,k)=.5*(ww(i,k,j)+ww(i,k-1,j))*rdnw(k-1)    &
157. G g b gb           * (ph_1(i,k,j)-ph_1(i,k-1,j)+phb(i,k,j)-phb(i,k-1,j))
158. G g b gb           rhs(i,k) = dts*(ph_tend(i,k,j) + .5*g*(1.-epssm)*w(i,k,j))
159. G g b gb----> ENDDO
160. G g b-----> ENDDO
```

Cray compiler interchanges these loops – uses fewer registers (56 vs. 63) resulting in fewer spills for a 4% speedup.

# advance\_w and OpenACC

OpenACC Parallelism defined at tile loop – advance\_w caller

Define GPU kernel and partition into thread blocks at tile level. Can tile in 2 horizontal directions.

```
71. G-----< !$ACC PARALLEL LOOP
72. G g-----<      do tile=1,numtiles
73. G g
74. G g
75. G g igr4 I-->      call advance_w( w_2, rw_tend, rgrid%aww, u_2, v_2
76. G g                                mu_2, mut, muave, muts,
77. G g                                t_2save, t_2, t_save,
78. G g                                ph_2, ph_save, phb, ph_tend,
79. G g                                ht, c2a, cqw, alt, alb,
80. G g                                a, alpha, gamma,
81. G g                                rdx, rdy, dts_rk, t0, epssm,
82. G g                                dnw, fnm, fnp, rdnw, rdn,
83. G g                                cf1, cf2, cf3, msft,
84. G g                                ids,ide, jds,jde, kds,kde,
85. G g                                ims,ime, jms,jme, kms,kme,
86. G g                                i_start(tile), i_end(tile), &
87. G g                                j_start(tile), j_end(tile), &
88. G g                                kds      , kde           )
89. G g
90. G g----->      enddo
91. G-----> !$ACC END PARALLEL LOOP
```

# advance\_w and OpenACC

OpenACC Parallelism defined at tile loop – advance\_w caller

```
71. G-----< !$ACC PARALLEL LOOP
72. G g-----<      do tile=1,numtiles
73. G g
74. G g
75. G g igr4 I-->      call advance_w( w_2, rw_tend, rgrid%aww, u_2, v_2      &
76. G g                               mu_2, mut, muave, muts,
77. G g                               t_2save, t_2, t_save,
78. G g                               ph_2, ph_save, phb, ph_tend,
79. G g                               ht, c2a, cqw, alt, alb,
80. G g                               a, alpha, gamma,
81. G g                               rdx, rdy, dts_rk, t0, epssm,
82. G g                               dnw, fnm, fnp, rdnw, rdn,
83. G g                               cf1, cf2, cf3, msft,
84. G g                               ids,ide, jds,jde, kds,kde,      & ! domain dims
85. G g                               ims,ime, jms,jme, kms,kme,      & ! memory dims
86. G g                               i_start(tile), i_end(tile), &
87. G g                               j_start(tile), j_end(tile), &
88. G g                               kds      , kde
89. G g
90. G g----->      enddo
91. G-----> !$ACC END PARALLEL LOOP
```

Advance\_w subroutine inlined and loops inside are partitioned within thread block.



# advance\_w and OpenACC

Cray compiler loop mark listing messages.

```
71. G-----< !$ACC PARALLEL LOOP
72. G g-----<      do tile=1,numtiles
73. G g
74. G g
75. G g igr4 I-->      call advance_w( w_2, rw_tend, rgrid%aww, u_2, v_2      &
76. G g           mu_2, mut, msft, muts )
```

ftn-6418 ftn: ACCEL File = advance\_w\_driver.F90, Line = 71

If not already present: allocate memory and copy whole array "alpha" to accelerator, free at line 91 (acc\_copyin).

ftn-6418 ftn: ACCEL File = advance\_w\_driver.F90, Line = 71

If not already present: allocate memory and copy whole array "a" to accelerator, free at line 91 (acc\_copyin).

ftn-6418 ftn: ACCEL File = advance\_w\_driver.F90, Line = 71

If not already present: allocate memory and copy whole array "gamma" to accelerator, free at line 91 (acc\_copyin).

ftn-6424 ftn: ACCEL File = advance\_w\_driver.F90, Line = 71

Private array "mut\_inv" was allocated to shared memory.

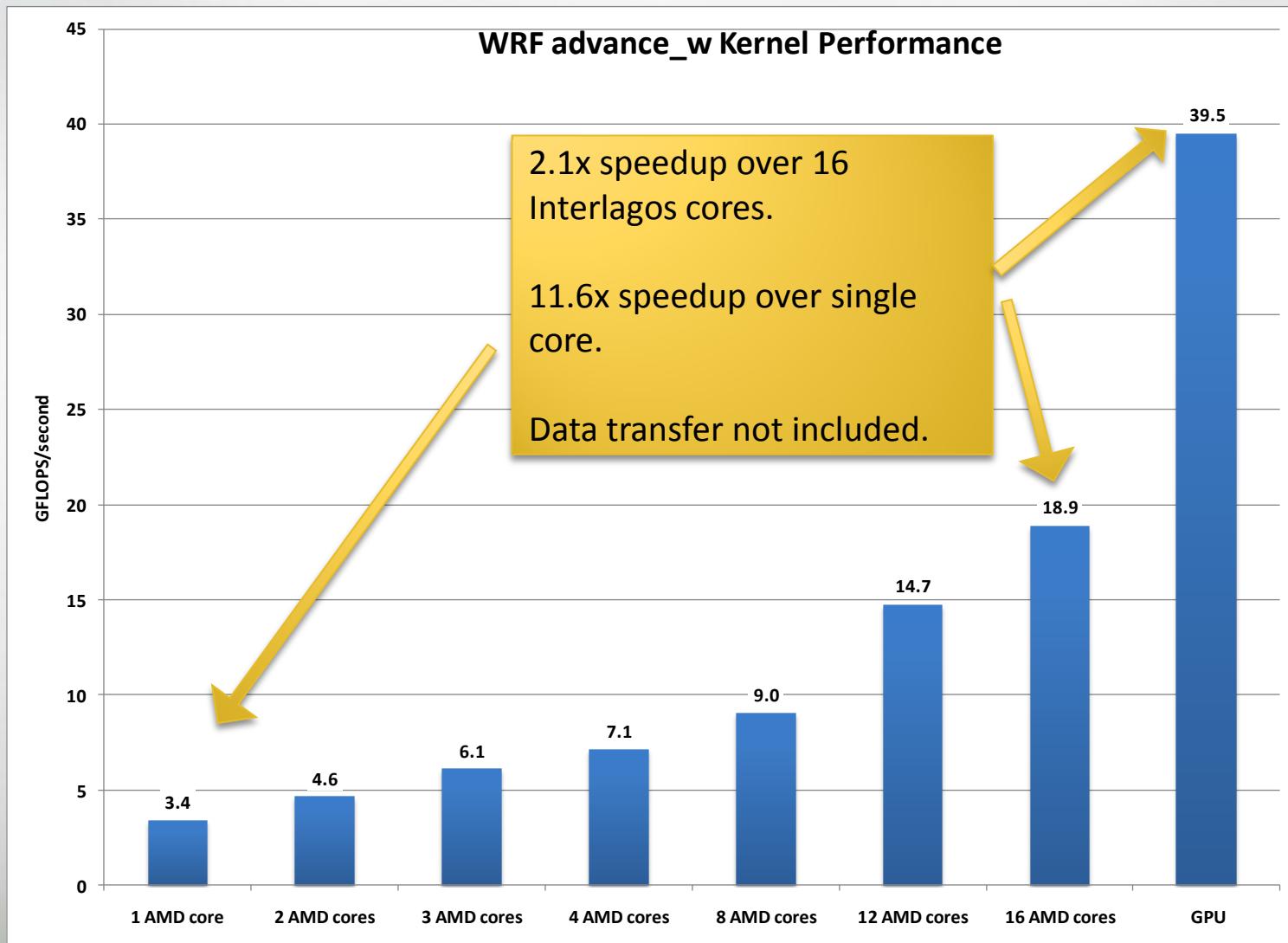
ftn-6424 ftn: ACCEL File = advance\_w\_driver.F90, Line = 71

Private array "msft\_inv" was allocated to shared memory.

ftn-6430 ftn: ACCEL File = advance\_w\_driver.F90, Line = 72

A loop starting at line 72 was partitioned across the thread blocks.

# advance\_w Results



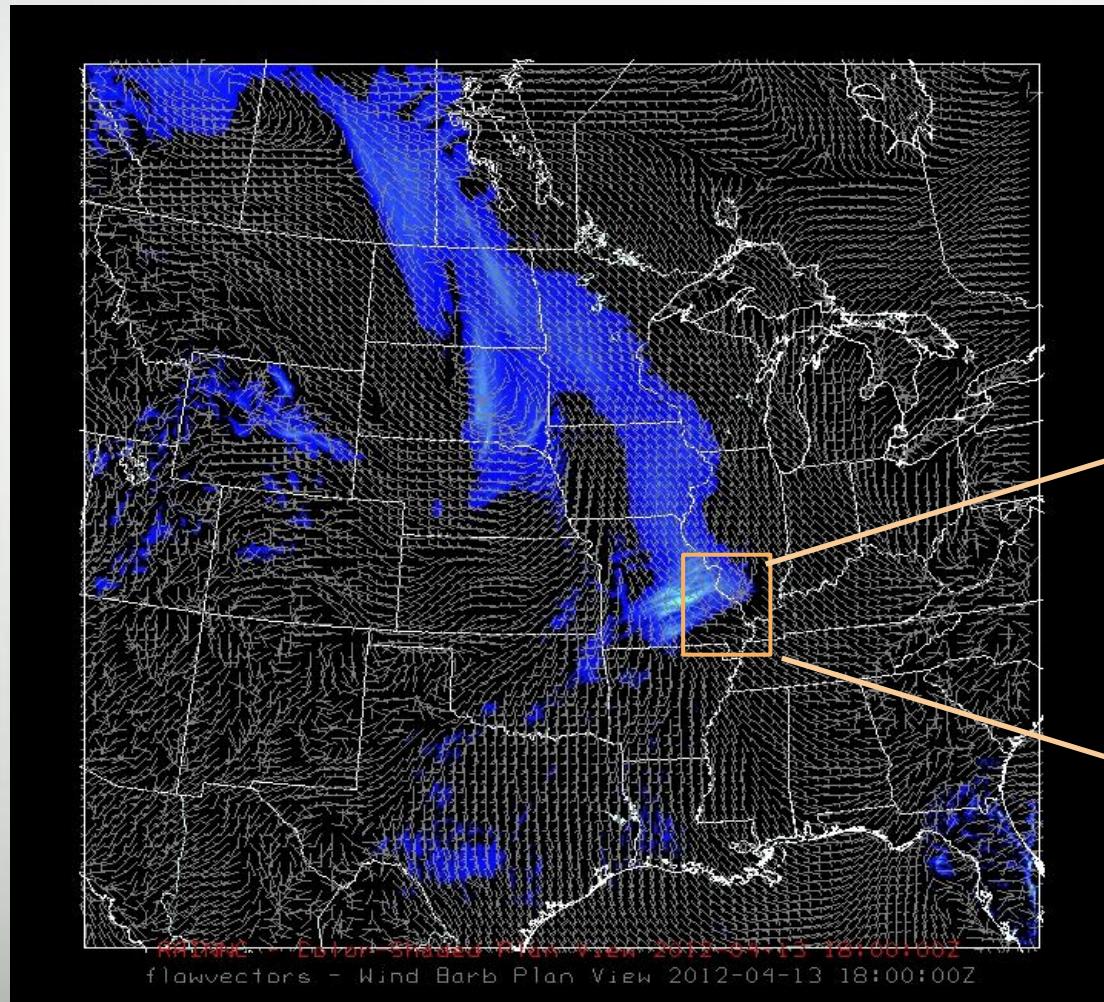
# Other WRF Experiments with GPUs

- wsm5 microphysics in CUDA C, 3.1x speedup over Nehalem quad-core, 8 threads
  - Michalakes, et al.
  - See <http://www.mmm.ucar.edu/wrf/WG2/GPU/WSM5.htm>
- RRTM radiation physics routine in CUDA Fortran, 11x speedup over single XEON X5550 Nehalem core
  - Ruetsch, et al.
  - See [http://data1.gfdl.noaa.gov/multi-core/2011/presentations/Ruetsch\\_GPU%20Acceleration%20of%20the%20Longwave%20Rapid%20Radiative%20Transfer%20Model%20in%20WRF%20Using%20CUDA%20Fortran.pdf](http://data1.gfdl.noaa.gov/multi-core/2011/presentations/Ruetsch_GPU%20Acceleration%20of%20the%20Longwave%20Rapid%20Radiative%20Transfer%20Model%20in%20WRF%20Using%20CUDA%20Fortran.pdf)
- wsm5 and wsm3 using OpenACC directives (NCAR distribution)
  - Tests underway with PGI and Cray CCE compilers on XK6
- Additional kernels for wsm6 microphysics and small\_step (acoustic) dynamics in test

- Part 2 – Estimating full WRF solver performance on GPU using results from kernel experiments on multiple nodes

# WRF Test Domain

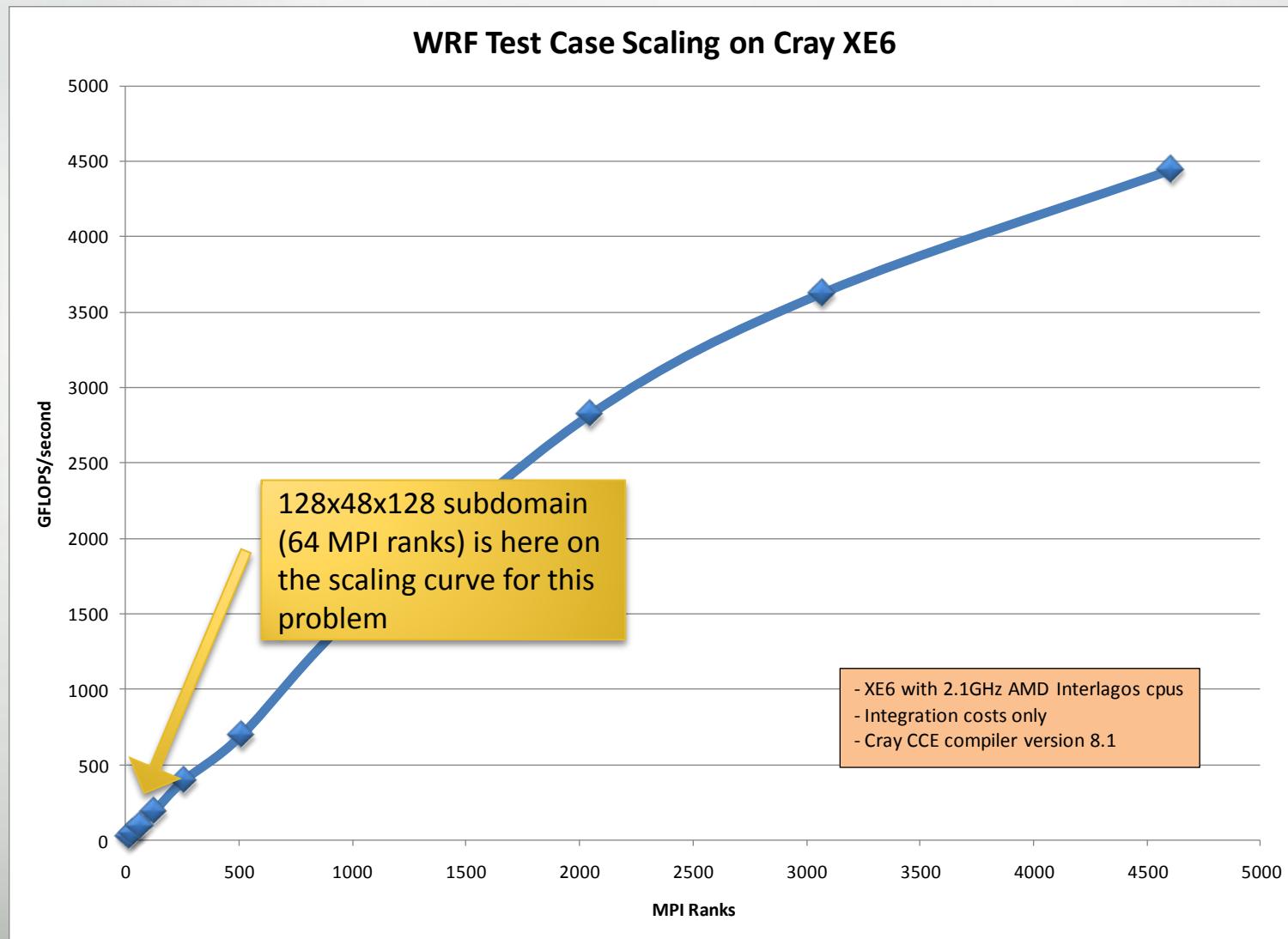
- 1024 x 48 x 1024 Global Domain Size
- Decomposed using 64 MPI ranks resulting in 128 x 48 x 128 subdomains per MPI rank



Subdomain of 128x48x128,  
1 for each GPU

Estimates based on XE6  
performance for this  
subdomain with active  
precipitation (microphysics  
work)

# WRF Test Case Scaling



# Estimating WRF Forecast on GPU

- Estimate a 3 hour forecast on multiple nodes of XK6
  - Adequately account for periodic radiation calculations
- What performance difference can we expect between XE6 Opteron nodes and XK6 Opteron + GPU nodes
- Based on WRF ARW solver (`solve_em`) results on XE6
- Include measured host <-> device data transfer times
  - WRF prognostic variables are transferred to GPU once at start, retrieved from GPU once at end
  - Halo exchange planes are packed/unpacked on GPU and transferred to/from host
- Assume all calculations (all tile loops in `solve_em`) are on GPU
- MPI communications are done on Opteron

# WRF ARW Solver

Transfer state variables to GPU



Time Step Loop (720 times for 3 hour forecast)

Required state and work variables transferred to GPU device only once for 3 hour forecast period.

Call Solve\_em

Tile Loop

    Call dynamics routines

Halo exchange

Tile Loop

    Call physics routines

Halo exchange

Etc....

End Solve\_em

End Time Step Loop

Transfer state variables to Host (for output)

# WRF ARW Solver

Transfer state variables to GPU

Time Step Loop (720 times for 3 hour forecast)

    Call Solve\_em

    Tile Loop

        Call dynamics routines

    Halo exchange

    Tile Loop

        Call physics routines

    Halo exchange

    Etc....

End Solve\_em

End Time Step Loop

Transfer state variables to Host (for output)

Time step loop. 720 iterations for a 3 hour forecast (15 second time step)



# WRF ARW Solver

Transfer state variables to GPU

Time Step Loop (720 times for 3 hour forecast)

Call Solve\_em

Tile Loop

Call dynamics routines

Halo exchange

Tile Loop

Call physics routines

Halo exchange

Etc....

End Solve\_em

End Time Step Loop

Transfer state variables to Host (for output)

Up to 73 tile loops in each pass through solve\_em. All parallelized with OpenMP (host).

Ideally, each tile loop a GPU kernel using OpenACC

# WRF ARW Solver

Transfer state variables to GPU

Time Step Loop (720 times for 3 hour forecast)

Call Solve\_em

Tile Loop

Call dynamics routines

Halo exchange

Tile Loop

Call physics routines

Halo exchange

Etc....

End Solve\_em

End Time Step Loop

Transfer state variables to Host (for output)

# WRF ARW Solver

Transfer state variables to GPU

Time Step Loop (720 times for 3 hour forecast)

Call Solve\_em

Tile Loop

Call dynamics routines

Halo exchange

Tile Loop

Call physics routines

Halo exchange

Etc....

End Solve\_em

End Time Step Loop

Transfer state variables to Host (for output)

# WRF ARW Solver

Transfer state variables to GPU

Time Step Loop (720 times for 3 hour forecast)

Call Solve\_em

Tile Loop

Call dynamics routines

Halo exchange

Tile Loop

Call physics routines

Halo exchange

Etc....

End Solve\_em

End Time Step Loop

Transfer state variables to Host (for output)

State variables moved to host only once after 3 hour forecast computed.

External boundary conditions processed here if required.

# Estimating 3 hour forecast on GPU

$$T_{total} = T_{load} + 720 * (G_{dyn} * \sum_1^{Nd} T_{dyn} + G_{phy} * \sum_1^{Np} T_{phy} + \sum_1^{45} T_{halo} + \sum_1^{45} T_{halo\_xfer}) + T_{retrieve}$$

Where

GPU Related Terms

$T_{total}$  = Total time to run a 3 hour forecast on GPU

$T_{load}$  = Time to load state variables from host to GPU device

720 = Number of time steps in 3 hour forecast

$G_{dyn}$  = Expected speedup for dynamics routines on GPU

$T_{dyn}$  = Time for dynamics routines in 1 time step on Opteron

$G_{phy}$  = Expected speedup for physics routines on GPU

$T_{phy}$  = Time for physics routines in 1 time step on Opteron

$T_{halo}$  = Time for halo exchange

$T_{halo\_xfer}$  = Time to transfer halo buffers from/to GPU

$T_{retrieve}$  = Time to retrieve state variables from GPU to host

# Expected GPU Performance for 3 Hour Forecast

	Equation Term	4 XE6 nodes with 2 Opteron Interlagos sockets (seconds)	GPU Speedup	64 XK6 nodes with 1 Opteron Interlagos socket plus 1 Fermi GPU (seconds)
Dynamics routines	$T_{dyn}$	1905	2.0	953
Physics routines	$T_{phy}$	1398	3.1	451
Halo exchanges	$T_{halo}$	307		307
Transfer state variables	$T_{load} + T_{retrieve}$			0.51
Transfer halo exchange buffers	$T_{halo\_xfer}$			3.97
Total time for 3 hour forecast	$T_{total}$	3610		1716
GPU Speedup				2.1x faster

# Further Studies

- There could be limited opportunities for asynchronous work
  - Overlapped halo exchanges and buffer moves
  - Multiple GPU kernels within a single tile loop
  - Kepler Hyper-Q allowing multiple MPI ranks and OpenMP threads to share GPU device
- Continuing Cray compiler optimizations and additional OpenACC features
- Additional WRF kernel tests underway
- Kepler performance
- Intel MIC
- Questions?

Thank You!

- Details follow

# Details for 3 hour Forecast Calculations

128x48x128 per MPI rank (l,k,j)		720 tsteps							
	gpu speedup	total time per timestep (sec) XE6	total for 3hr forecast (sec) XE6	time (sec) pinned	average bytes	occurrences per timestep	total time per timestep (sec) XK6	total for 3hr forecast (sec) XK6	
host to gpu 1 3d var				8.14E-04	3871488	300	0.2442	0.244	
host to gpu 1 2dvar				2.10E-05	80656	350	0.0073	0.007	
halo to host ( avg vars)				6.13E-05	241253	45	0.0028	1.986	
halo to gpu (avg vars)				6.13E-05	241253	45	0.0028	1.986	
halo exchanges (avg, mpi + pack/unpack)		0.428	307.800	9.50E-03	241253	45	0.4275	307.800	
physics total	3.10	1.942	1398.009				0.6263	450.971	
dynamics total	2.00	2.646	1905.278				1.3231	952.639	
gpu to host 1 3dvar				8.14E-04	3871488	300	0.2442	0.244	
gpu to host 1 2dvar				2.10E-05	80656	350	0.0073	0.007	
total time			3611.087					1715.843	
xe6/xk6								2.105	