Accelerating the Community Atmosphere Model – Spectral Element Method: The Challenges, the Science, and the Future Direction

Rick Archibald

Predictive Methods Group
Computer Science and Mathematics Division
Climate Change Science Institute
Oak Ridge National Laboratory

Titan Users and Developers Workshop (East Coast) and Users Meeting
Knoxville, TN
February 19th, 2013

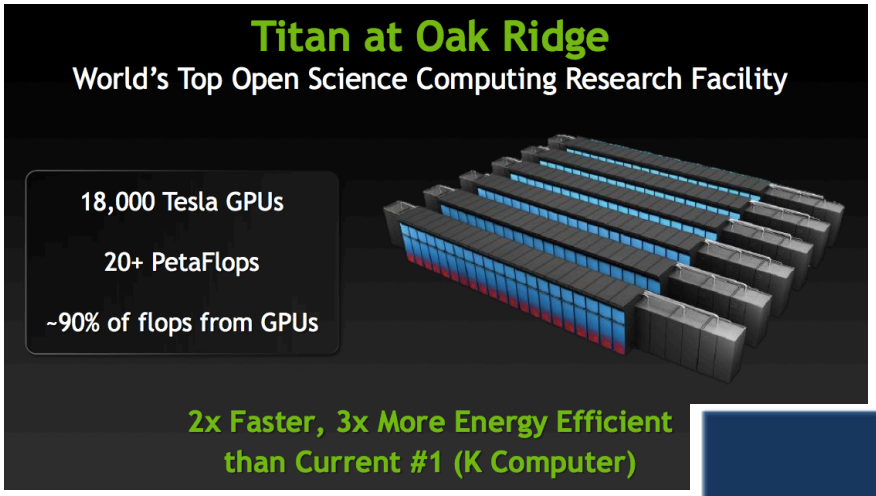
Teams and Projects

Valentine Anantharaj[1,3]	ORNL: OLCF
Rick Archibald[1,2,3]	ORNL: PMG/CCSI
Chris Baker[,2]	ORNL: CEES/CASL
Ilene Carpenter[1]	NREL
Katherine Evans*[23]	ORNL: CESG/CCSI
Jeffrey Larkin[1]	Cray → NVIDIA
Aaron Lott[2]	LLNL
Matthew Norman*[1]	ORNL: OLCF
Paulius Micikevicius[1]	NVIDIA
Mark Taylor[23]	SANDIA
Carrol Woodward[2]	LLNL

***Team/Project Leader**

1. **Center for Accelerated Application Readiness CAM-SE**
2. **Multiscale SCIDAC Process Integration**
3. **Ultra High Resolution Global Climate Simulation**

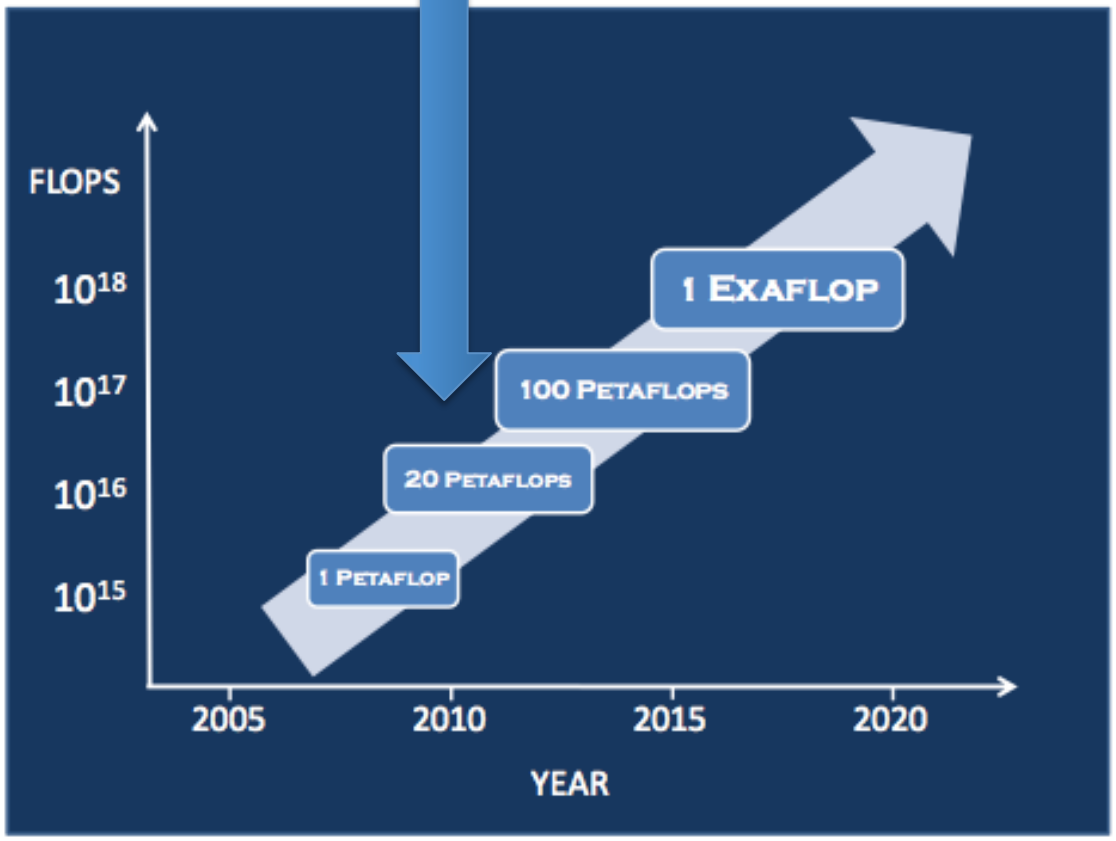
Motivation



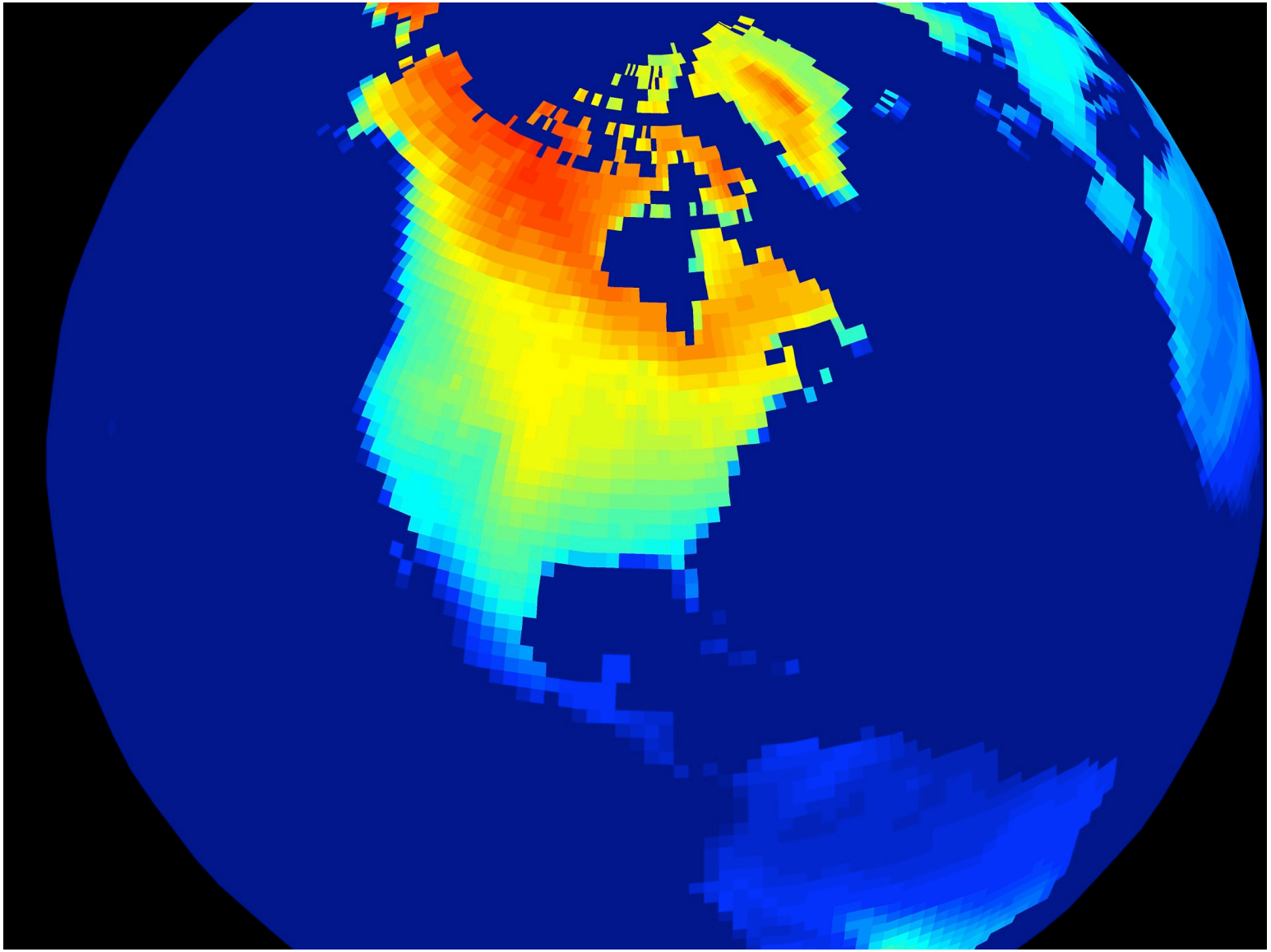
TOP 10 - 06/2012

1	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom
2	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect
3	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom
4	SuperMUC - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR
5	Tianhe-1A - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050
6	Jaguar - Cray XK6, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect,

NVIDIA Update, Jen-Hsun Huang

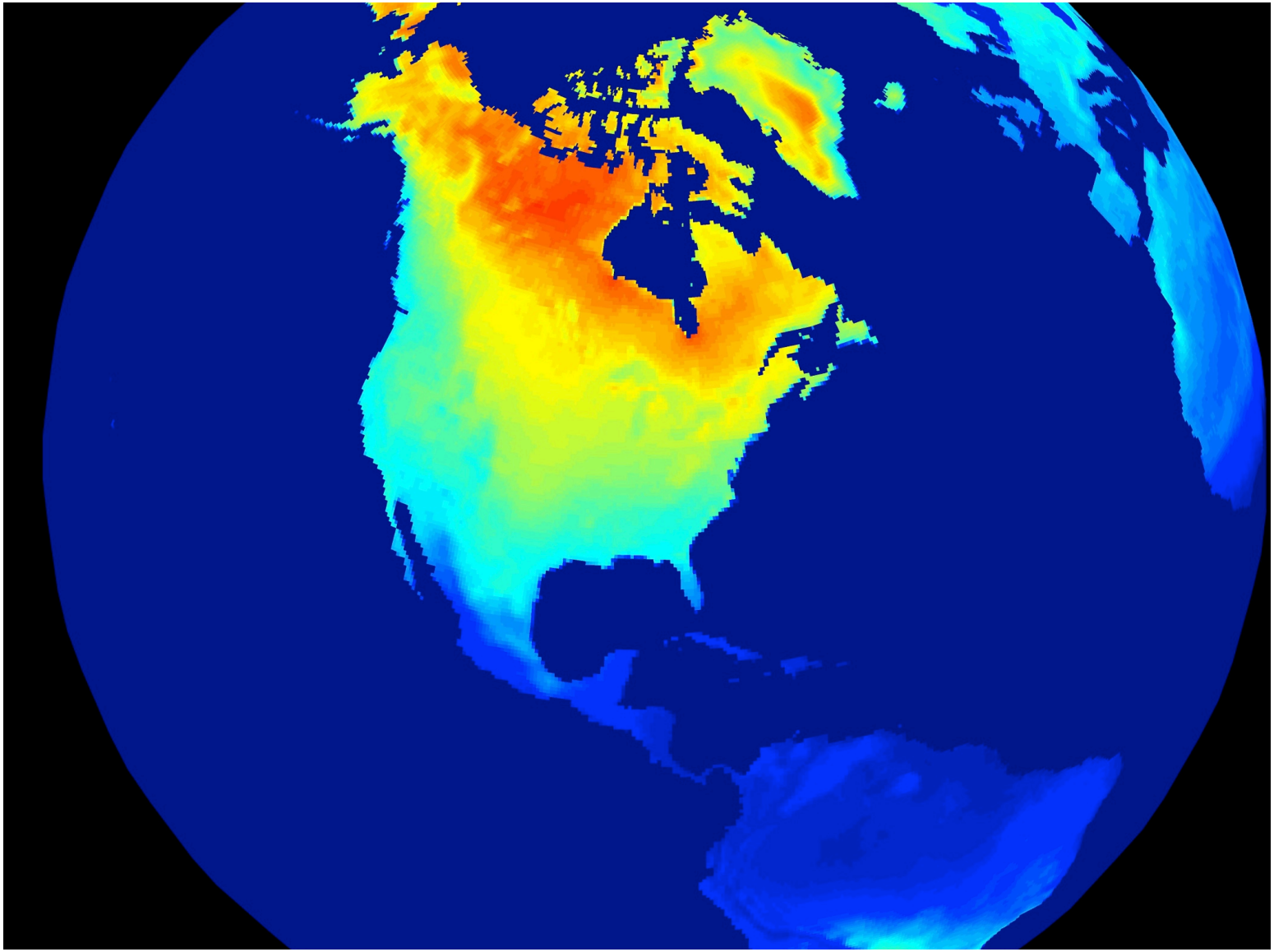


Motivation: Scientific



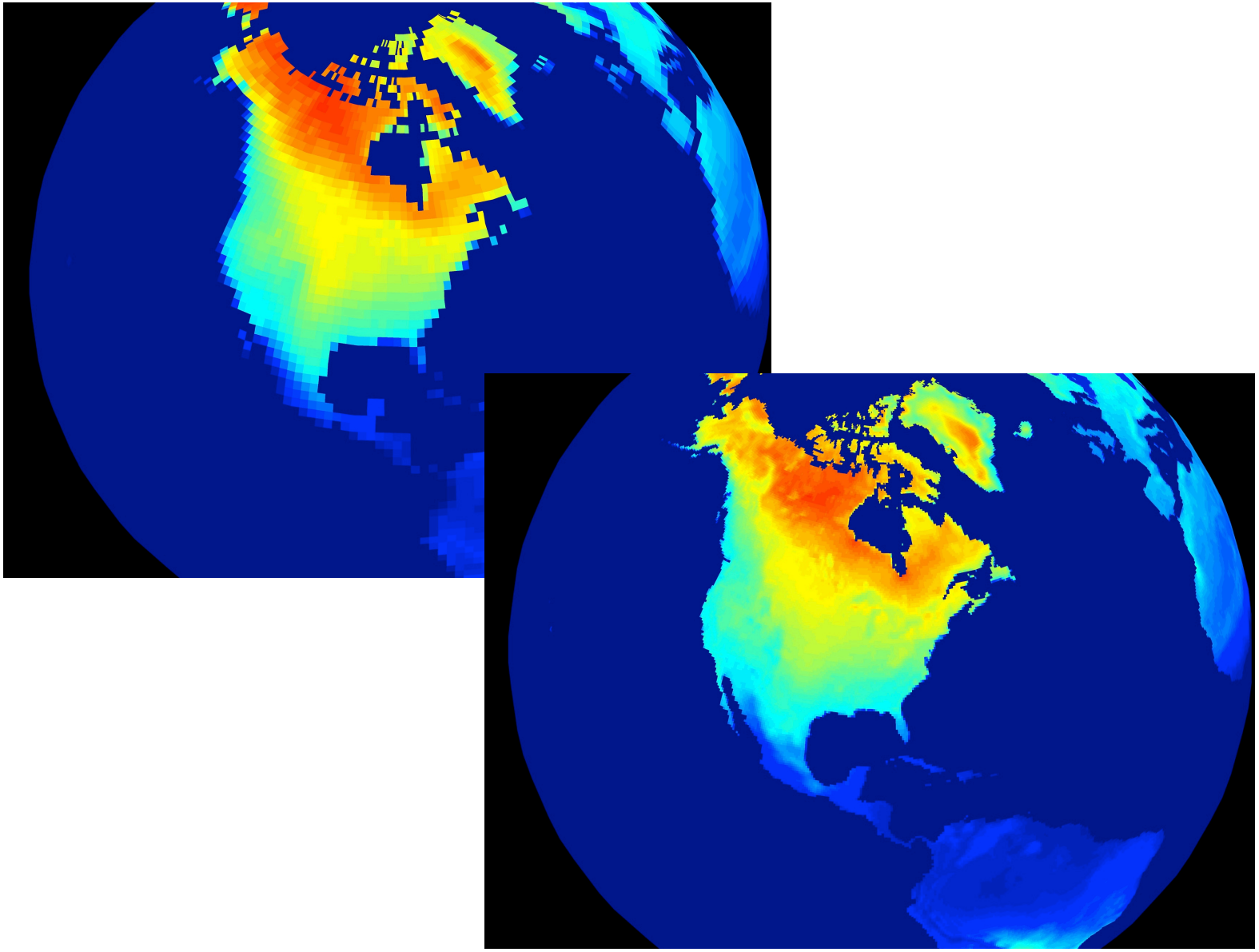
Variability of T85 Ensemble over 30 years of Simulation
Global Mean Variation 3 Degrees

Motivation: Scientific



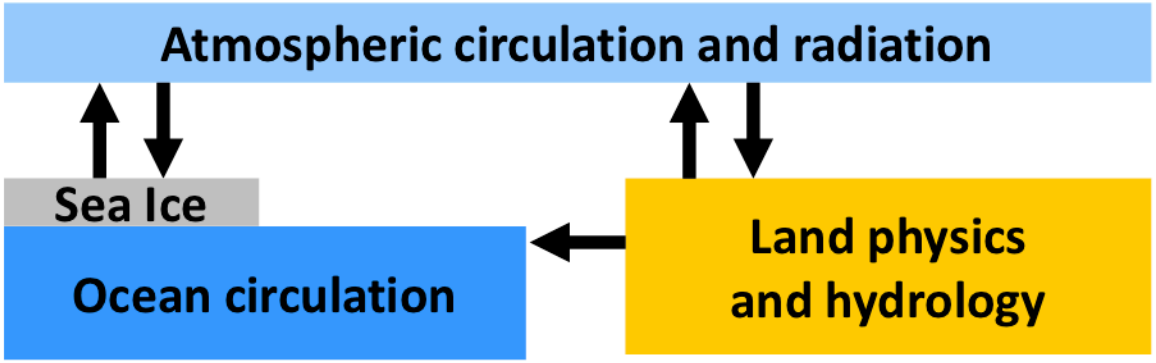
Variability of T341 Ensemble over 30 years of Simulation
Global Mean Variation 3.5 Degrees

Comparison

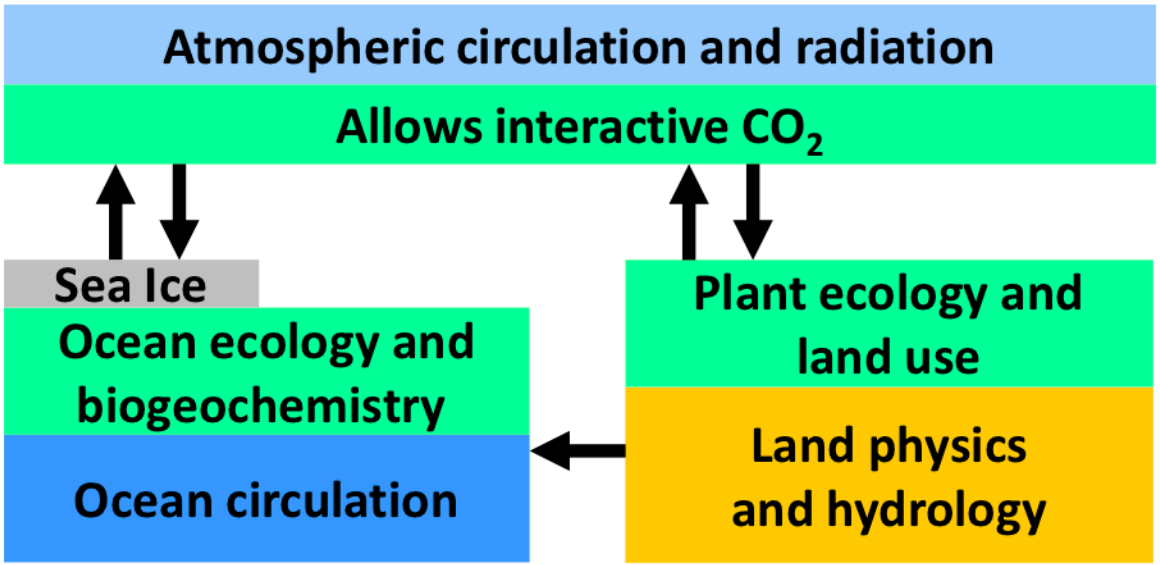


Earth System Model Description

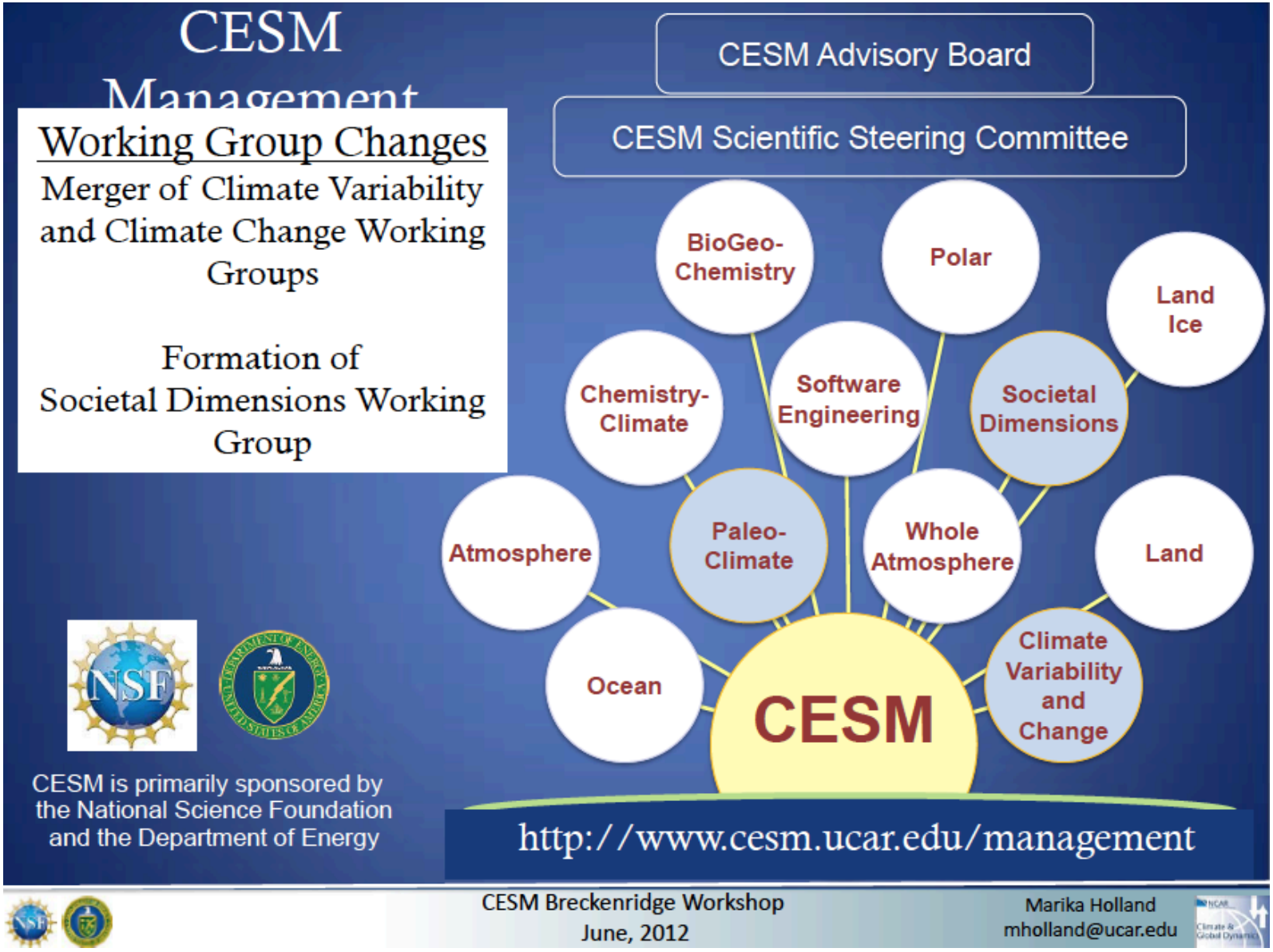
A Climate Model closes the radiative and hydrologic cycles



An Earth System Model closes additional cycles as well



State of the CESM

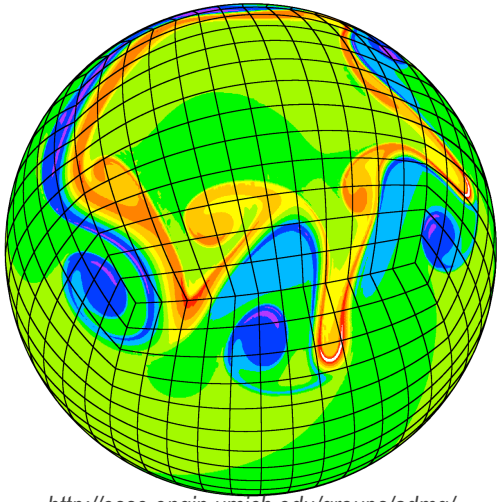


What is CAM-SE?

- Climate-scale atmospheric simulation for capability computing
- Comprised of (1) a dynamical core and (2) physics packages

What is CAM-SE?

- Climate-scale atmospheric simulation for capability computing
- Comprised of (1) a dynamical core and (2) physics packages



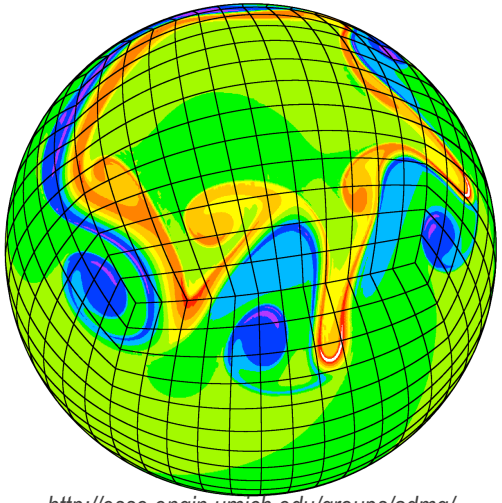
http://esse.engin.umich.edu/groups/admg/dcmip/jablonowski_cubed_sphere_vorticity.png

Dynamical Core

1. "Dynamics": wind, energy, & mass
2. "Tracer" Transport: (H_2O, CO_2, O_3, …)
   Transport quantities not advanced by the dynamics

What is CAM-SE?

- Climate-scale atmospheric simulation for capability computing
- Comprised of (1) a dynamical core and (2) physics packages



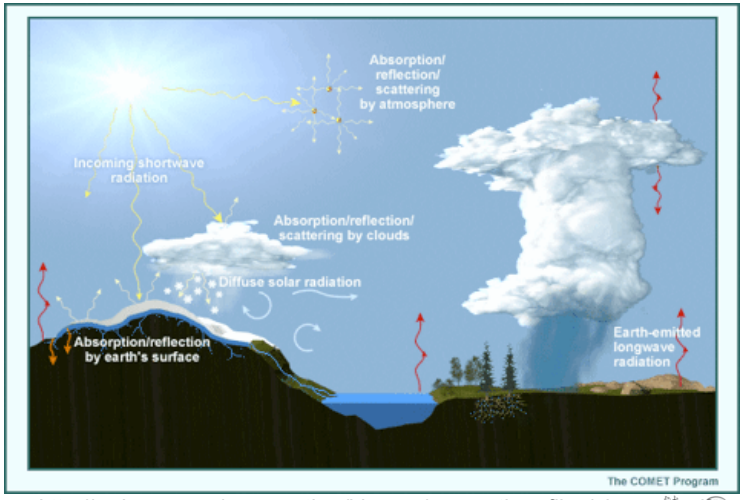
http://esse.engin.umich.edu/groups/admg/dcmip/jablonowski_cubed_sphere_vorticity.png

Dynamical Core

1. "Dynamics": wind, energy, & mass
2. "Tracer" Transport: (H_2O, CO_2, O_3, …)
   Transport quantities not advanced by the dynamics

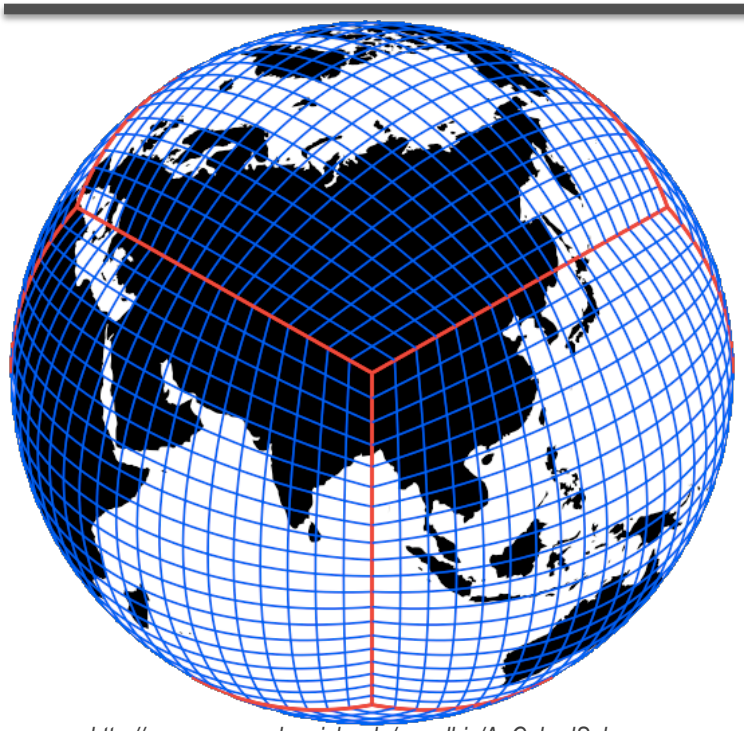
Physics Packages

Resolve anything interesting not included in dynamical core (moist convection, radiation, chemistry, etc)



http://web.me.com/macweather/blogger/maweather_files/physprc2.gif

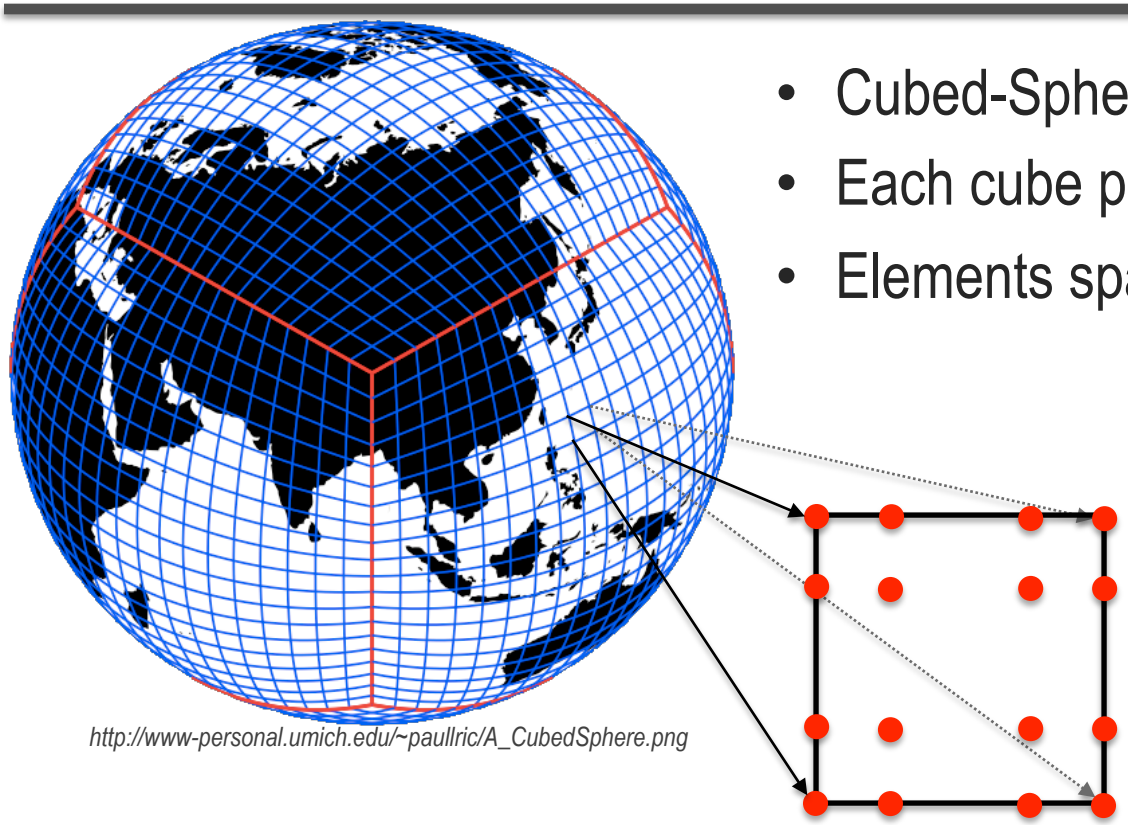
Gridding, Numerics, & Target Run



http://www-personal.umich.edu/~paullric/A_CubedSphere.png

- Cubed-Sphere + Spectral Element
- Each cube panel divided into elements

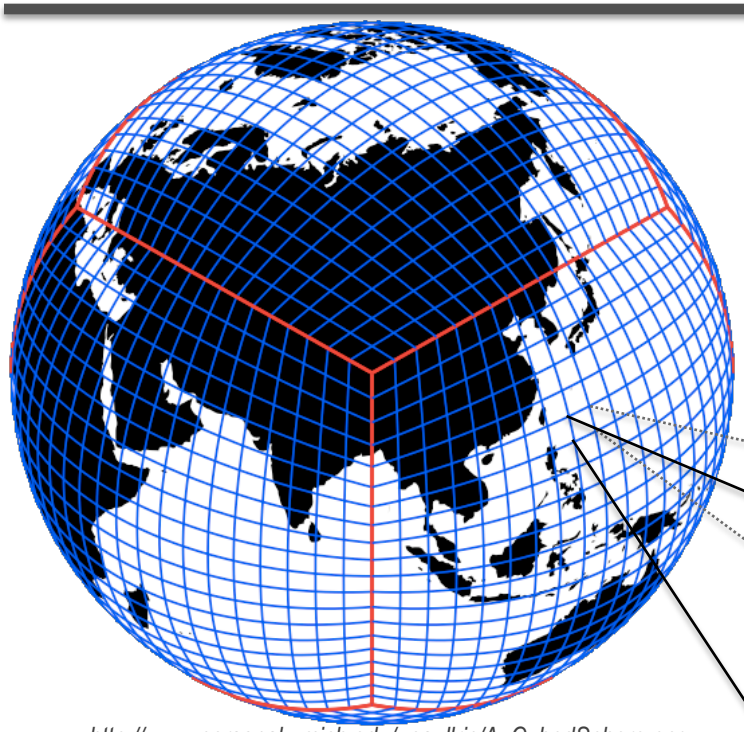
Gridding, Numerics, & Target Run



http://www-personal.umich.edu/~paullric/A_CubedSphere.png

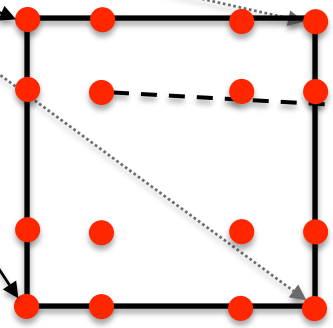
- Cubed-Sphere + Spectral Element
- Each cube panel divided into elements
- Elements spanned by basis functions

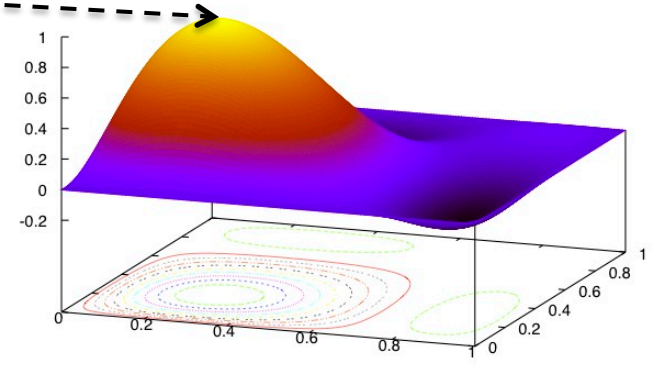
Gridding, Numerics, & Target Run



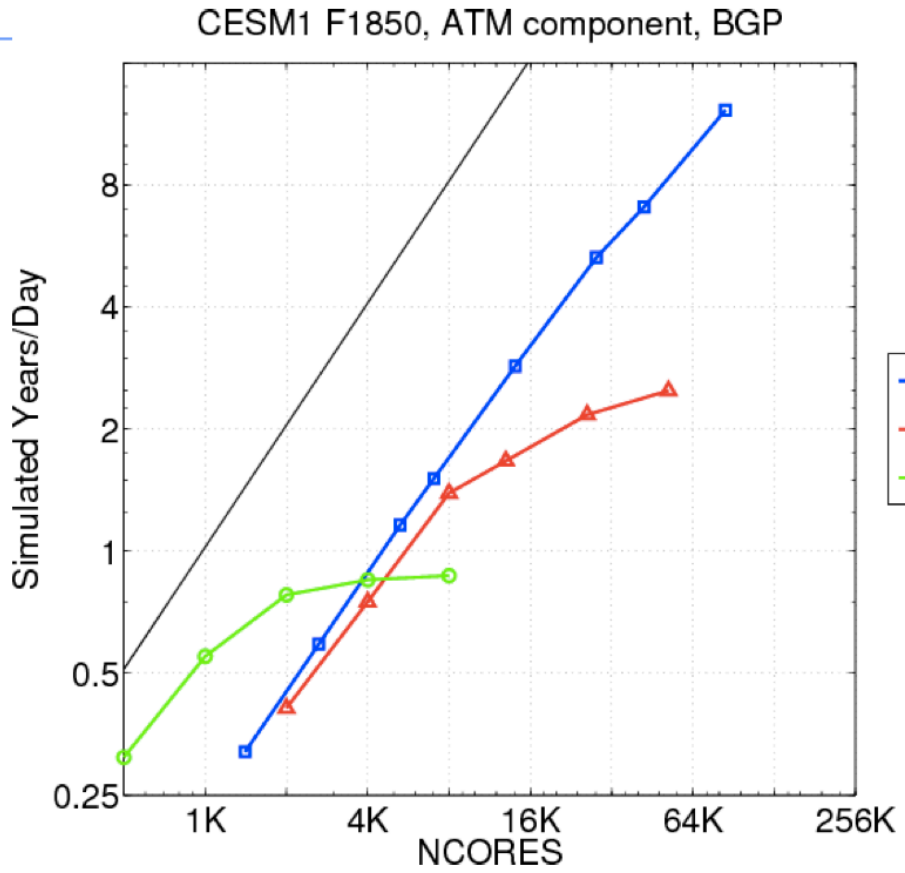
http://www-personal.umich.edu/~paullric/A_CubedSphere.png

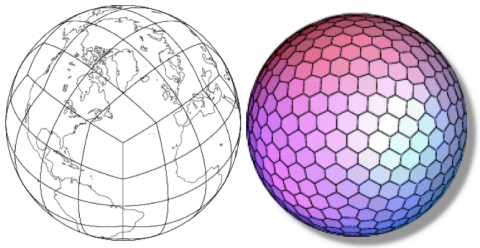
- Cubed-Sphere + Spectral Element
- Each cube panel divided into elements
- Elements spanned by basis functions
- Basis coefficients describe the fluid





Scalability: CAM4 ¼ degree (3 tracers)





- Faster performance at high resolution on parallel computers
- Not cheaper - cost in terms of core-hours is the same.
- CAM-Eul the most efficient (cheapest), but with lowest peak SYPD.

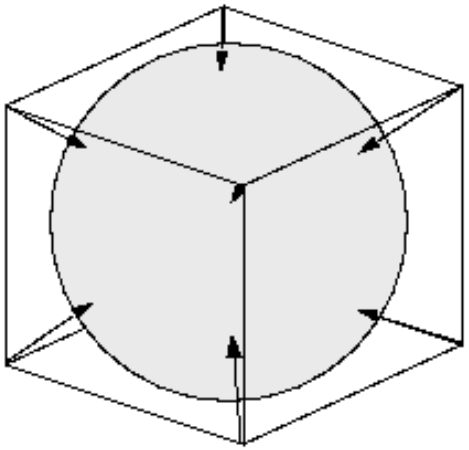
'New capabilities and new dynamical cores in CAM'. Mark Taylor, 16th Annual CESM Workshop, 2011.

Target 14km Simulations

- 16 billion degrees of freedom

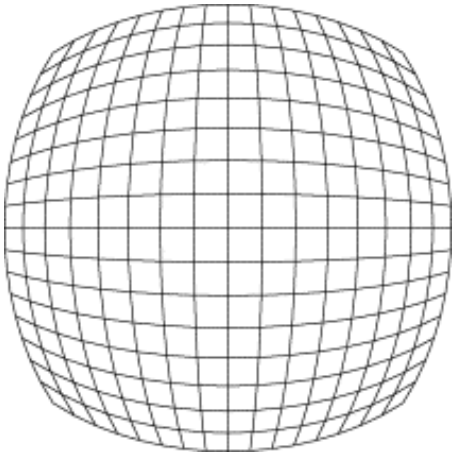
Target 14km Simulations

- 16 billion degrees of freedom
  - 6 cube panels



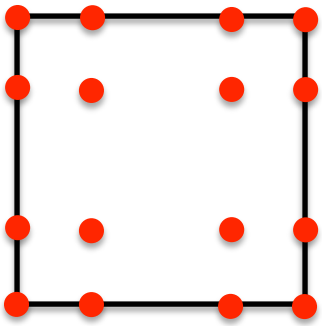
Target 14km Simulations

- 16 billion degrees of freedom
  - 6 cube panels
  - 240 x 240 columns of elements per panel



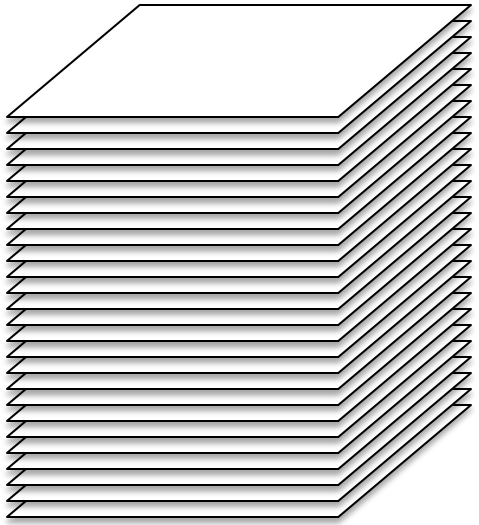
Target 14km Simulations

- 16 billion degrees of freedom
  - 6 cube panels
  - 240 x 240 columns of elements per panel
  - 4 x 4 basis functions per element



Target 14km Simulations

- 16 billion degrees of freedom
  - 6 cube panels
  - 240 x 240 columns of elements per panel
  - 4 x 4 basis functions per element
  - 26 vertical levels



Target 14km Simulations

- 16 billion degrees of freedom
  - 6 cube panels
  - 240 x 240 columns of elements per panel
  - 4 x 4 basis functions per element
  - 26 vertical levels
  - 110 prognostic variables

$$\rho, \rho u, \rho v, p$$

$$H_2O \ , \ CO_2 \ , \ O_3 \ , \ CH_4 \ , \ ...$$

Target 14km Simulations

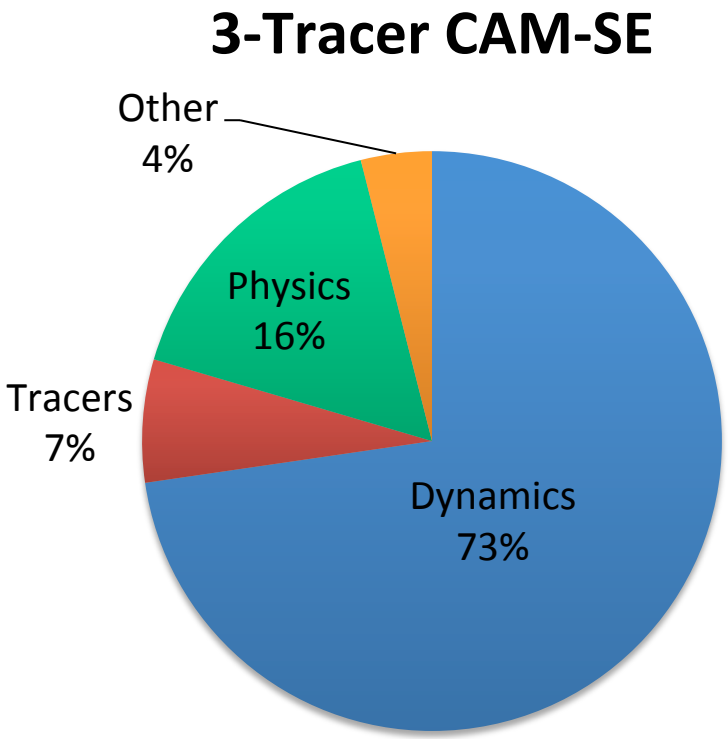
- 16 billion degrees of freedom
  - 6 cube panels
  - 240 x 240 columns of elements per panel
  - 4 x 4 basis functions per element
  - 26 vertical levels
  - 110 prognostic variables
- Scaled to 14,400 XT5 nodes with 60% parallel efficiency

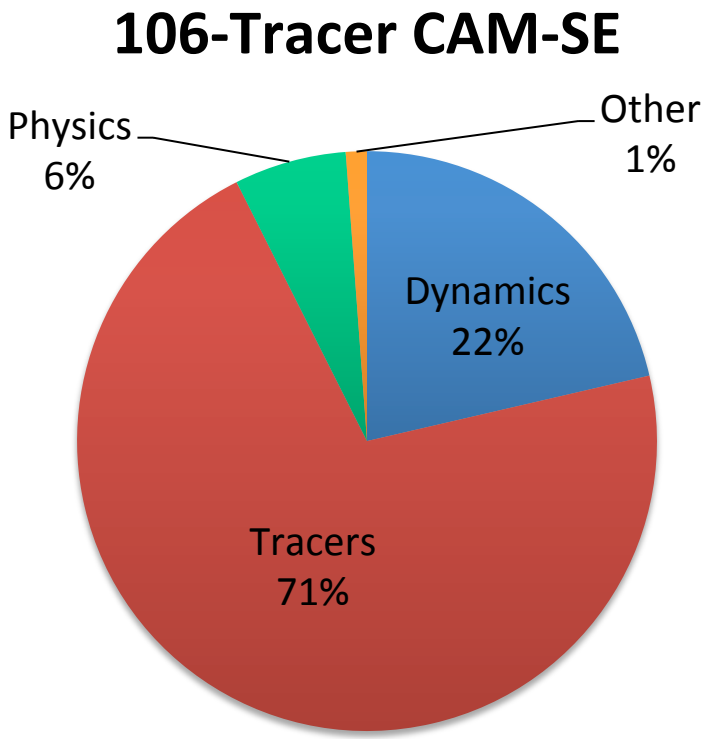
Target 14km Simulations

- 16 billion degrees of freedom
  - 6 cube panels
  - 240 x 240 columns of elements per panel
  - 4 x 4 basis functions per element
  - 26 vertical levels
  - 110 prognostic variables
- Scaled to 14,400 XT5 nodes with 60% parallel efficiency
- Must simulate 1-2 thousand times faster than real time
- With 10 second CAM-SE time step, need ≤ 10 ms per time step
  - 32-64 columns of elements per node, 5-10 thousand nodes

CAM-SE Profile (Cray XT5, 14K Nodes)

- Original CAM-SE used 3 tracers (20% difficult to port)
- Mozart chemistry provides 106 tracers (7% difficult to port)
  - Centralizes port to tracers with mostly data-parallel routines





The Routines of Tracer Transport

Three Main Routines

- **Euler_step (3x)**: Performs the actual transport
- **Advance_hypervis**: Stabilizes the solution with ∇^4 diffusion
- **Vertical_Remap**: Re-grids the data vertically to a reference grid

Three Helper Routines

- **Edge_pack**: Place all element edges into process-wide buffer
  Edge_unpack: Sum data at element edges
  - *Once* per euler_step *Twice* per advance_hypervis
- **Limiter2d_zero**: Remove negative coefficients & conserve mass
  - *Once* per euler_step *Once* per advance_hypervis

Call Tree Structure

```
prim_main
```

Main driver routines

```
prim_run_subcycle
```

Time stepping routines

```
prim_advec_tracers...
```

Tracer advection routines

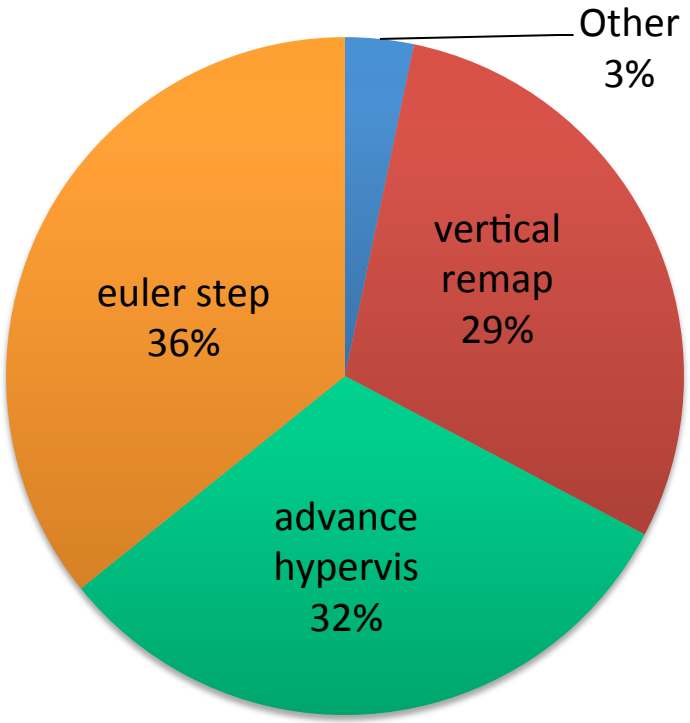
```
euler_step
  edgevpack
  bndry_exchangev   }=pack-unpack
  edgevunpack
    laplace_sphere_wk
      pack-unpack
remap_velocity
```

This routine called multiple times depending upon the type of timestep (ie, leapfrog, RK, etc).

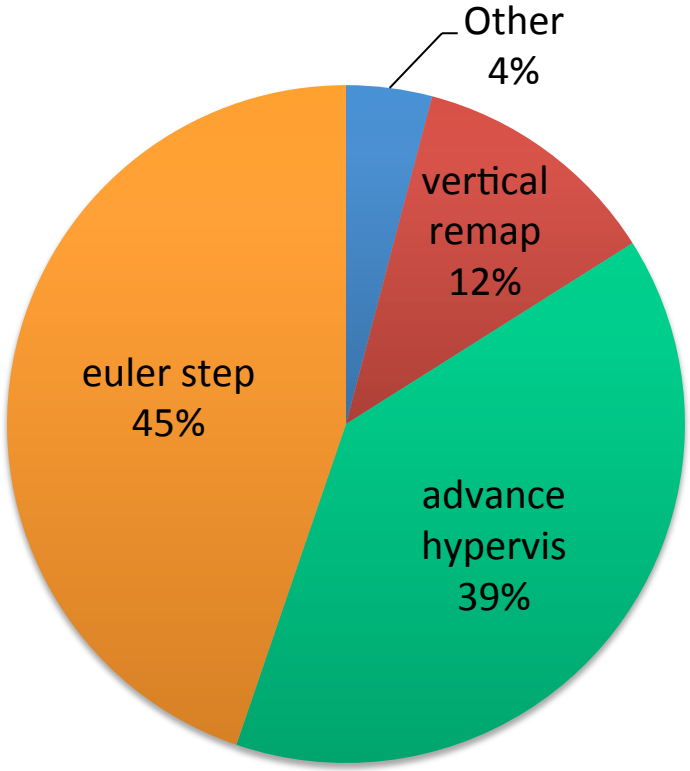
Profile of a Tracer Time Step

- We wrote a new remapping algorithm to reduce runtime
- Pack/exchange/unpack is ≈ half of euler_step & advance_hypervis

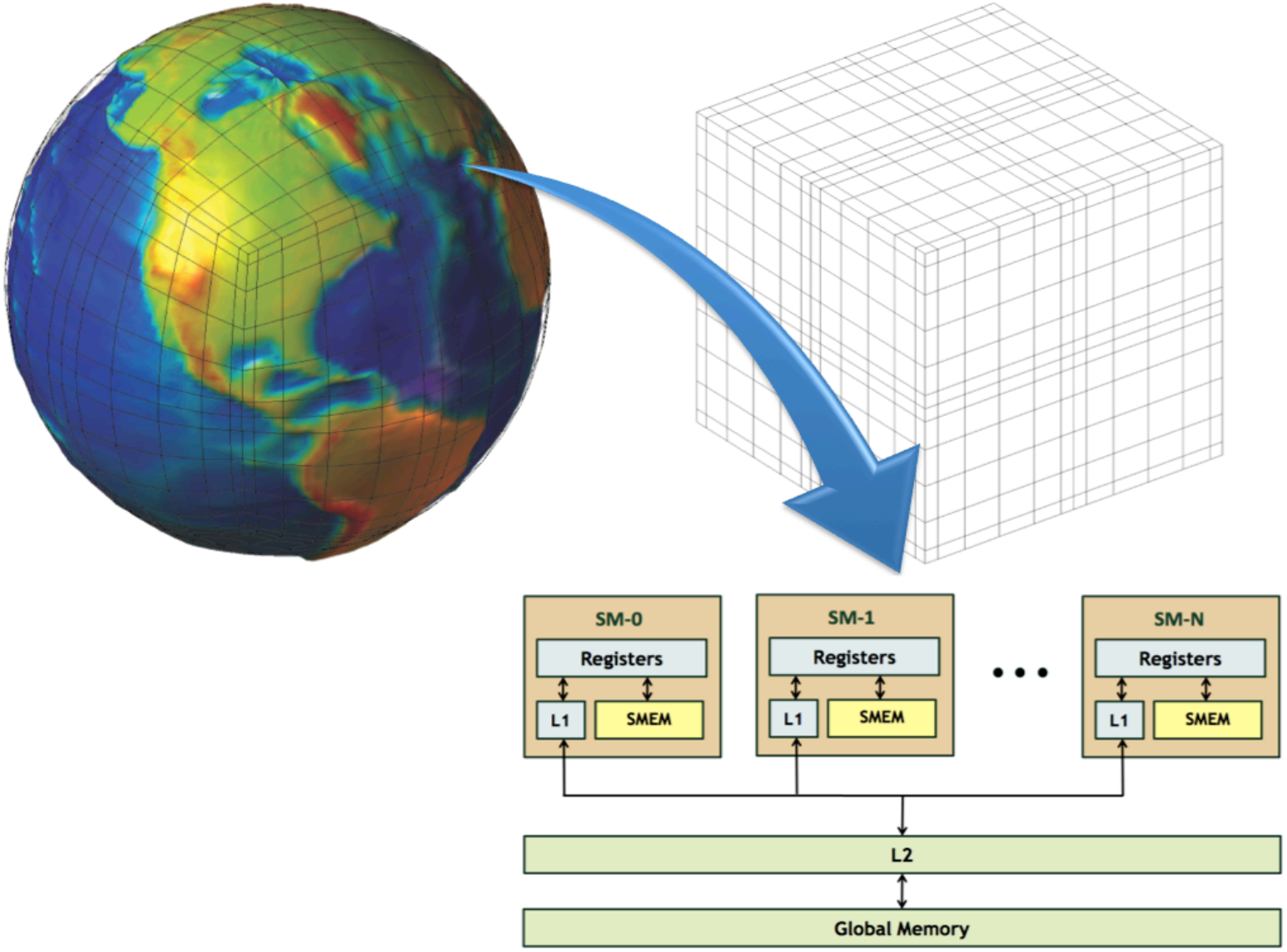
Original Vertical Remap



New Vertical Remap



GPU CAM-SE Goal



Node Design

http://www.thinkdigit.com/FCKeditor/uploads/26mar10470oin342t.jpg

http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg

Node Design

http://www.thinkdigit.com/FCKeditor/uploads/26mar10470oin342t.jpg

GPU Kernels

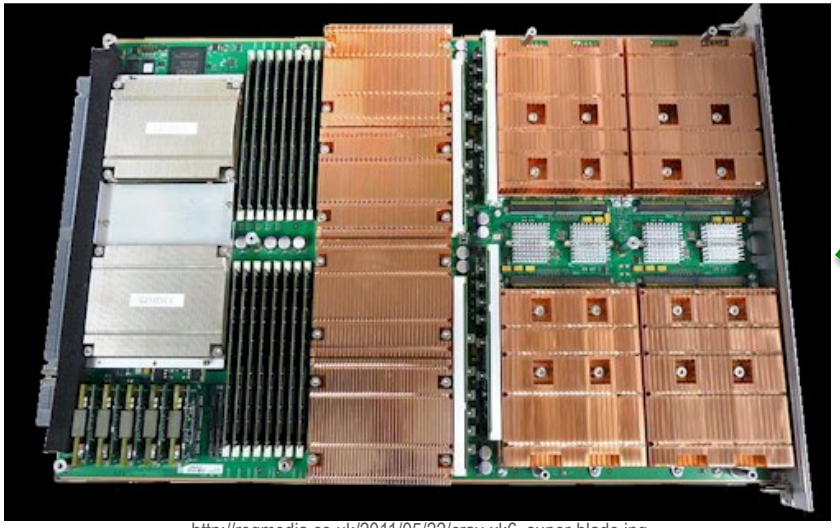

http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg

Node Design

http://www.thinkdigit.com/FCKeditor/uploads/26mar10470oin342t.jpg

GPU Kernels

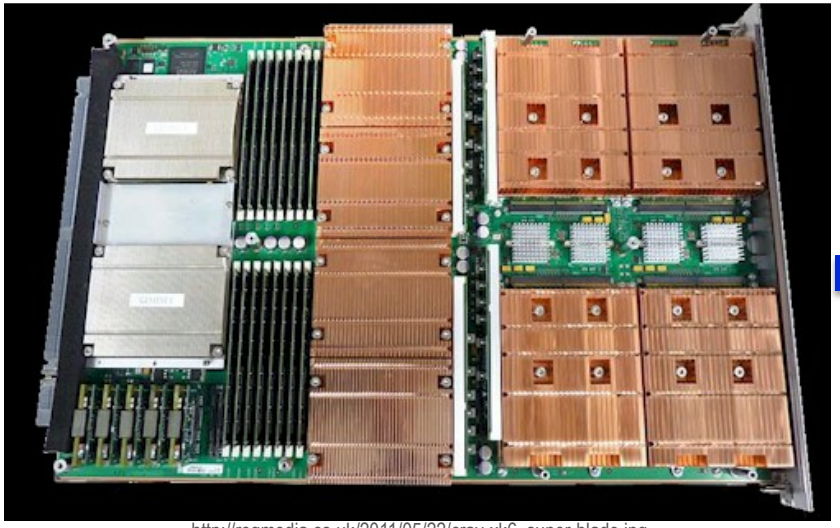
PCI-e D2H



http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg

Node Design

http://www.thinkdigit.com/FCKeditor/uploads/26mar10470oin342t.jpg



http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg

GPU Kernels

PCI-e D2H

PCI-e H2D

Node Design

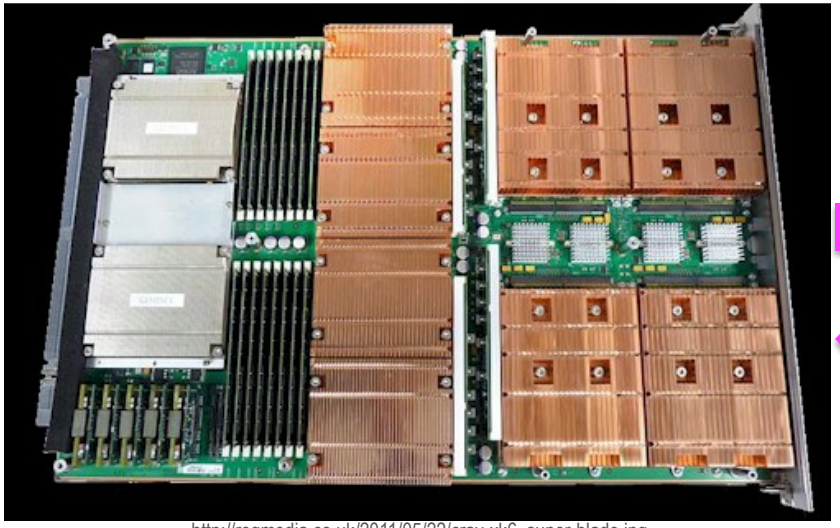
http://www.thinkdigit.com/FCKeditor/uploads/26mar10470oin342t.jpg

GPU Kernels

PCI-e D2H

PCI-e H2D

MPI


http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg

Node Design

http://www.thinkdigit.com/FCKeditor/uploads/26mar10470oin342t.jpg

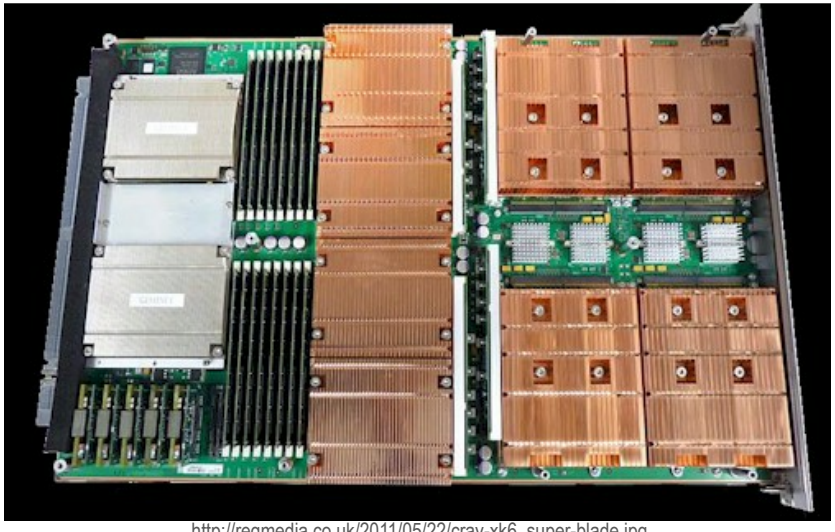
GPU Kernels

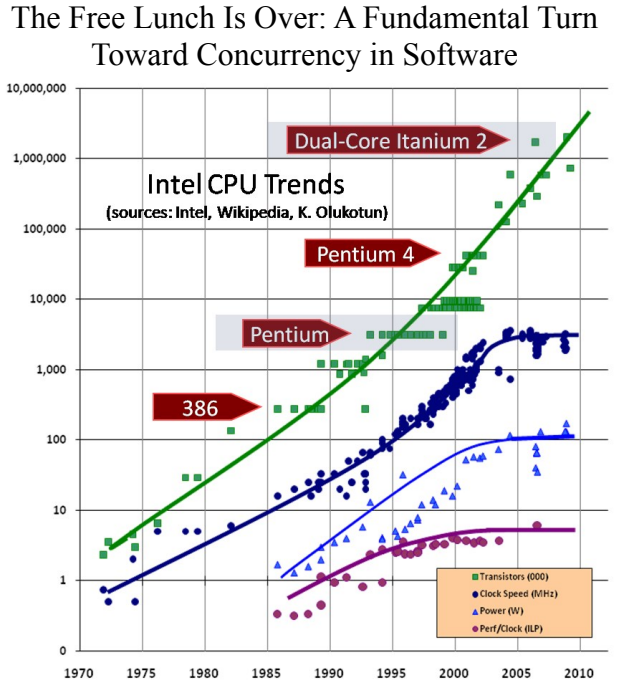
PCI-e D2H

PCI-e H2D

MPI

Host Computation


http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg



Node Design

http://www.thinkdigit.com/FCKeditor/uploads/26mar10470oin342t.jpg

If you were plowing a field, which would you rather use: 2 strong oxen or 1024 chickens?

– Seymour Cray

http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg

Node Design

http://www.thinkdigit.com/FCKeditor/uploads/26mar10470oin342t.jpg

If you were plowing a field, which would you rather use: 12,160 bipolar oxen[1] or 786,432 chickens[2]?

– Not Seymour Cray

http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg

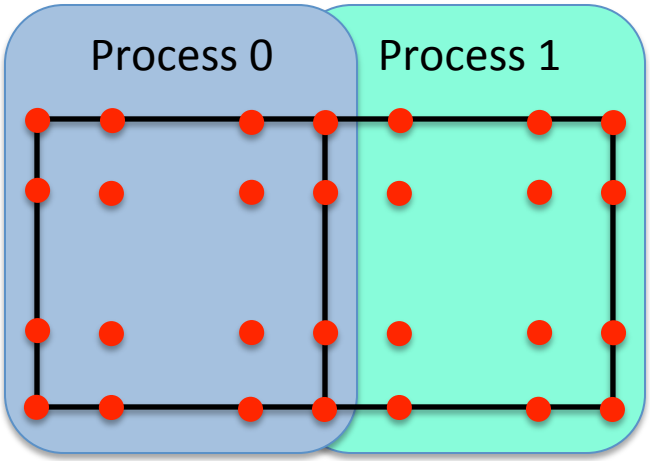
[1] A bipolar oxen is one AMD Bulldozer and one NVIDIA Fermi.
[2] A chicken is a Blue Gene/Q core.

Herding millions of chickens: Programming models for Blue Gene/Q and beyond

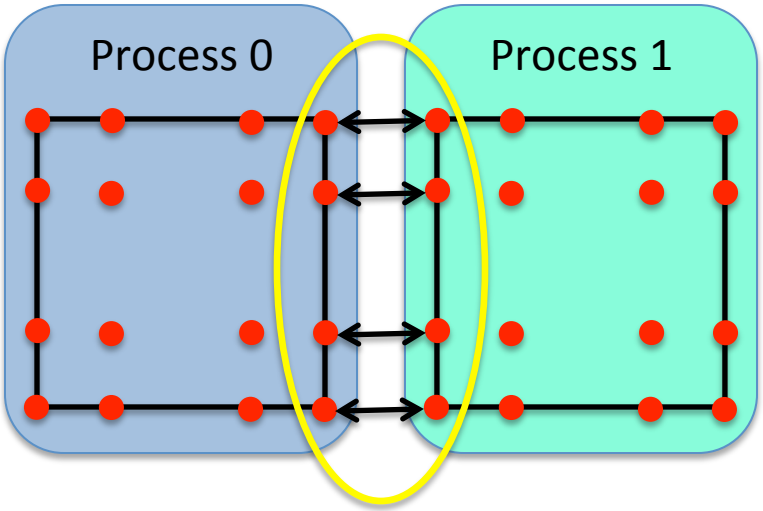
Jeff Hammond

Leadership Computing Facility
Argonne National Laboratory

Communication Between Elements



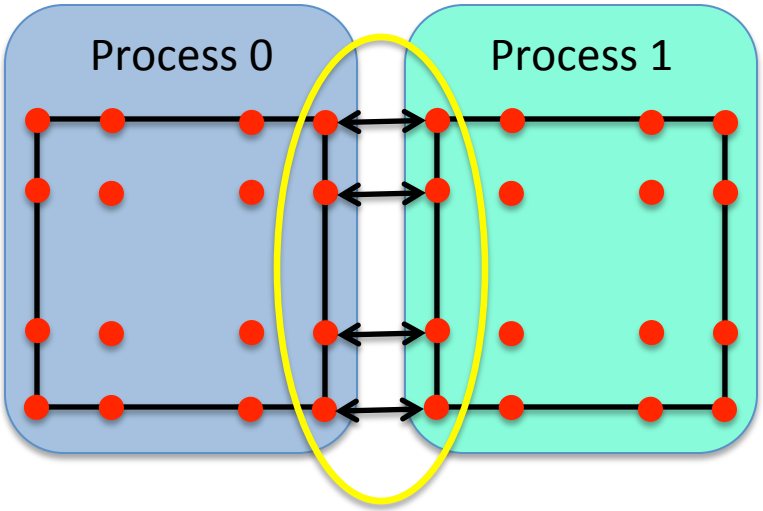
Communication Between Elements



Physically occupy the same location, Spectral Element requires them to be equal

Edges are averaged, and the average replaces both edges

Communication Between Elements



Physically occupy the same location, Spectral Element requires them to be equal

Edges are averaged, and the average replaces both edges

Implementation

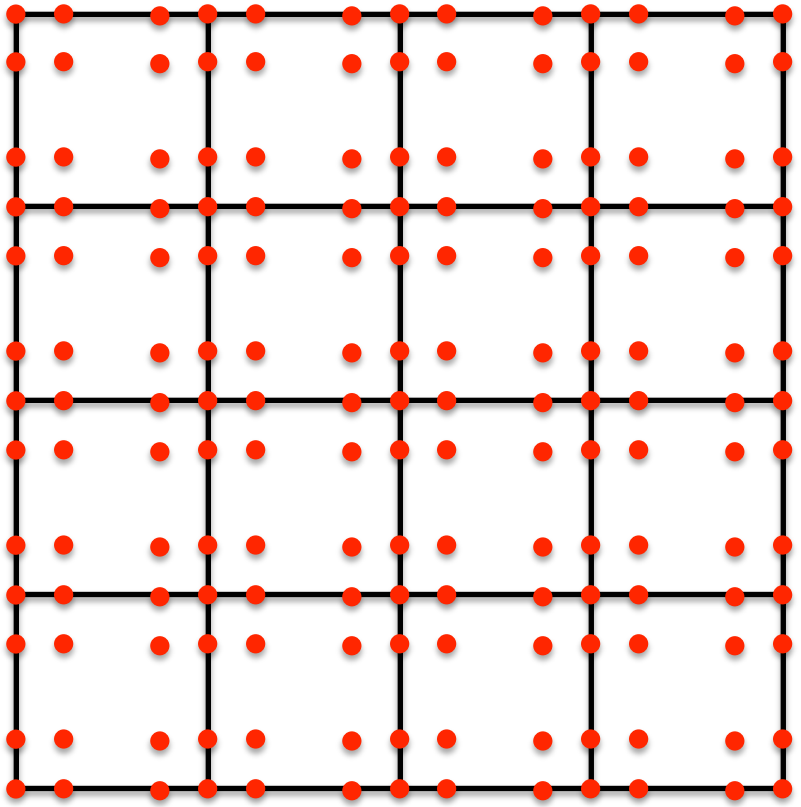
Edge_pack: pack all element edges into process-wide buffer. Data sent over MPI are contiguous in buffer.

Bndry_exchange: Send & receive data at domain decomposition boundaries

Edge_unpack: Perform a weighted sum for data at all element edges.

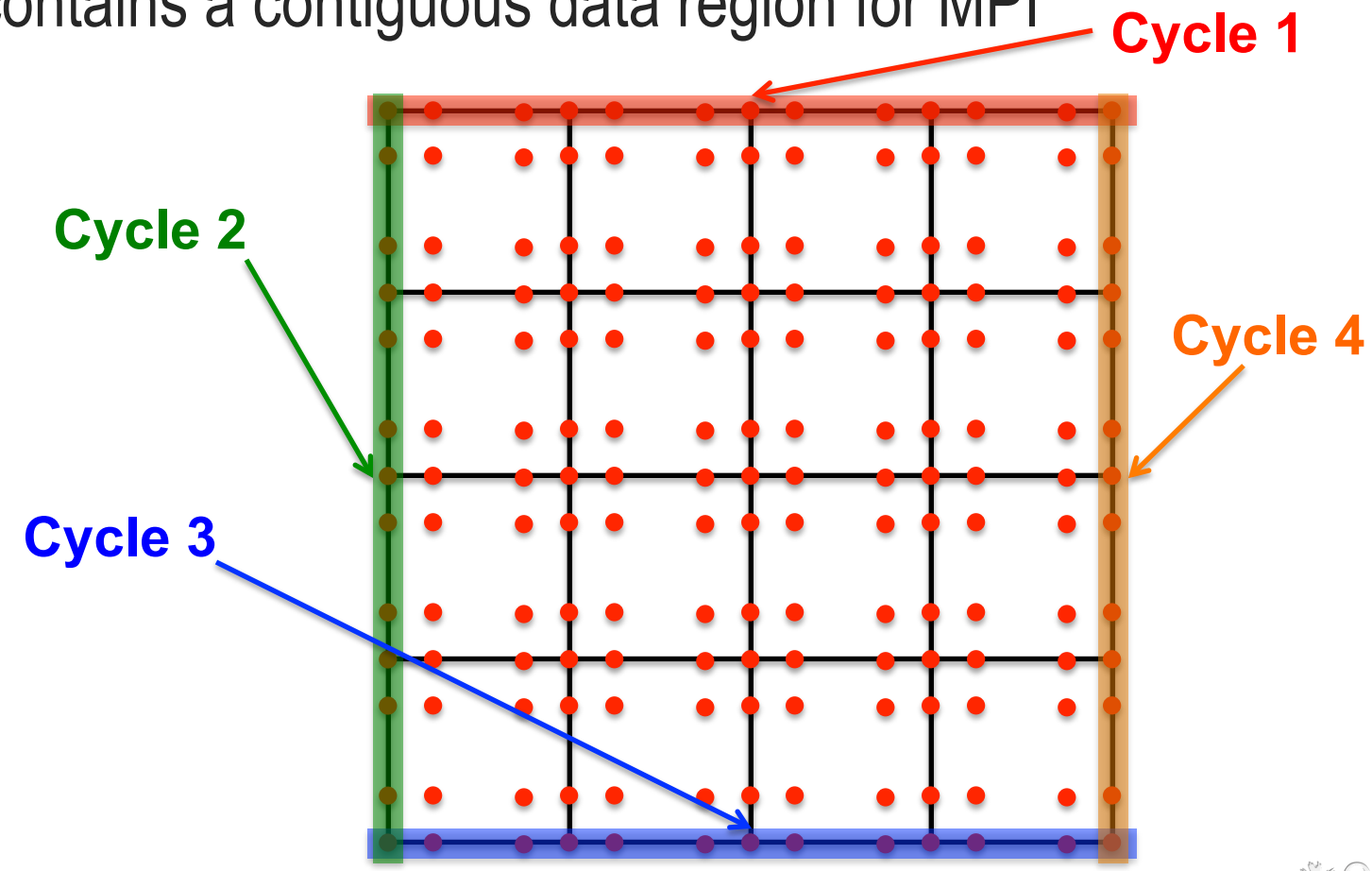
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”



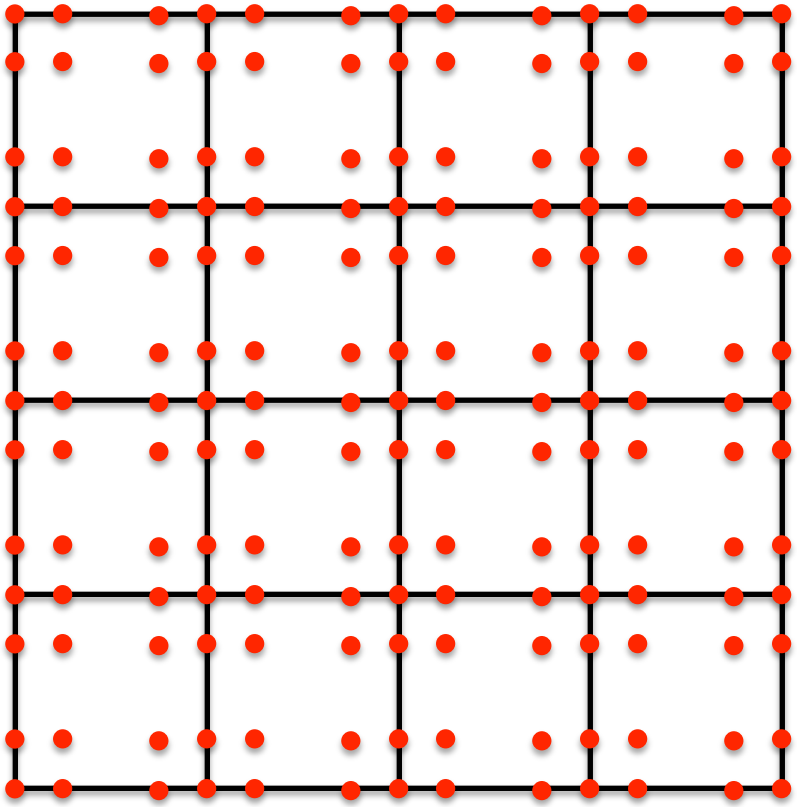
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI



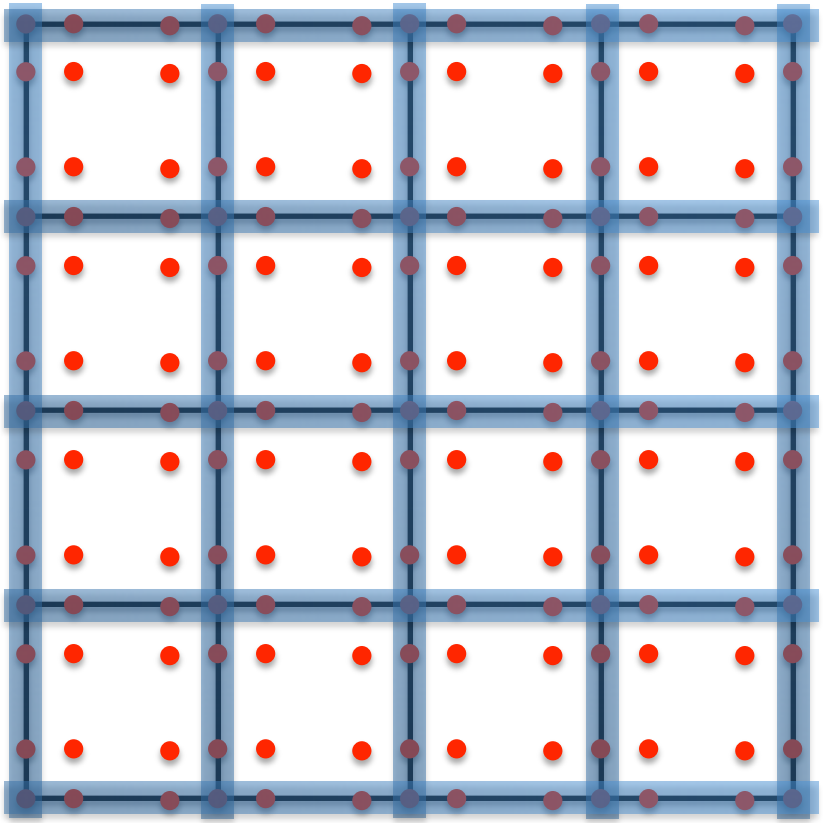
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack



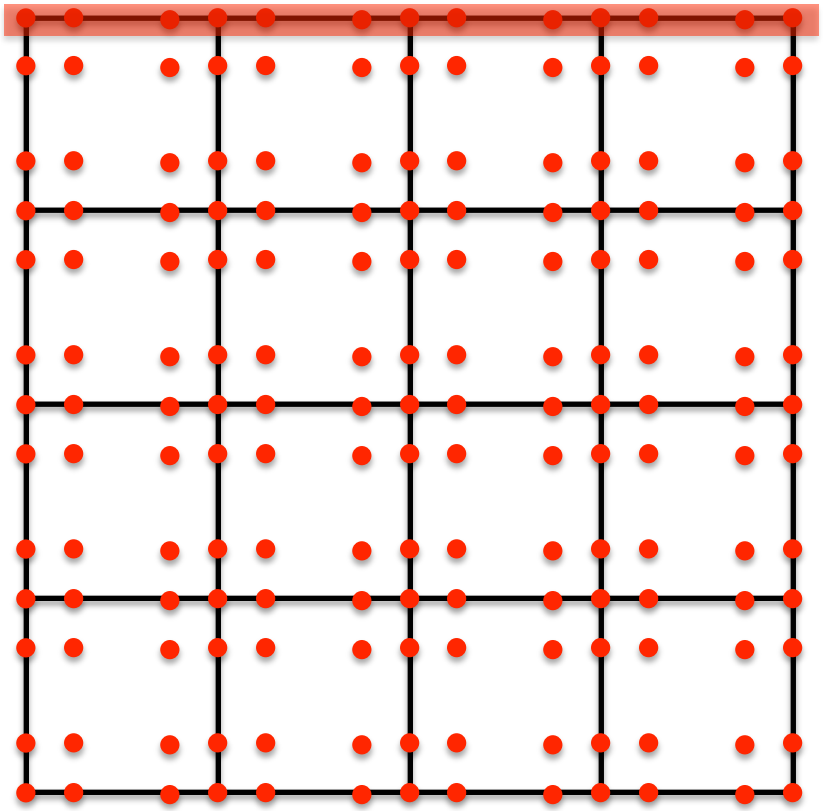
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel



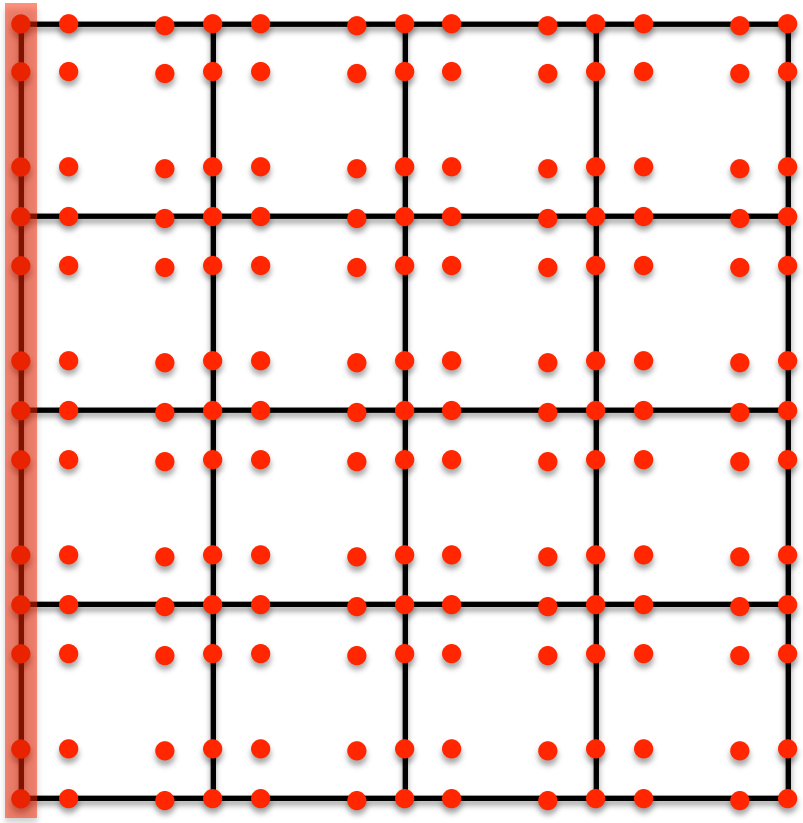
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel
  - For each “send cycle”
    - Send cycle over PCI-e (D2H)
    - MPI_Isend the cycle



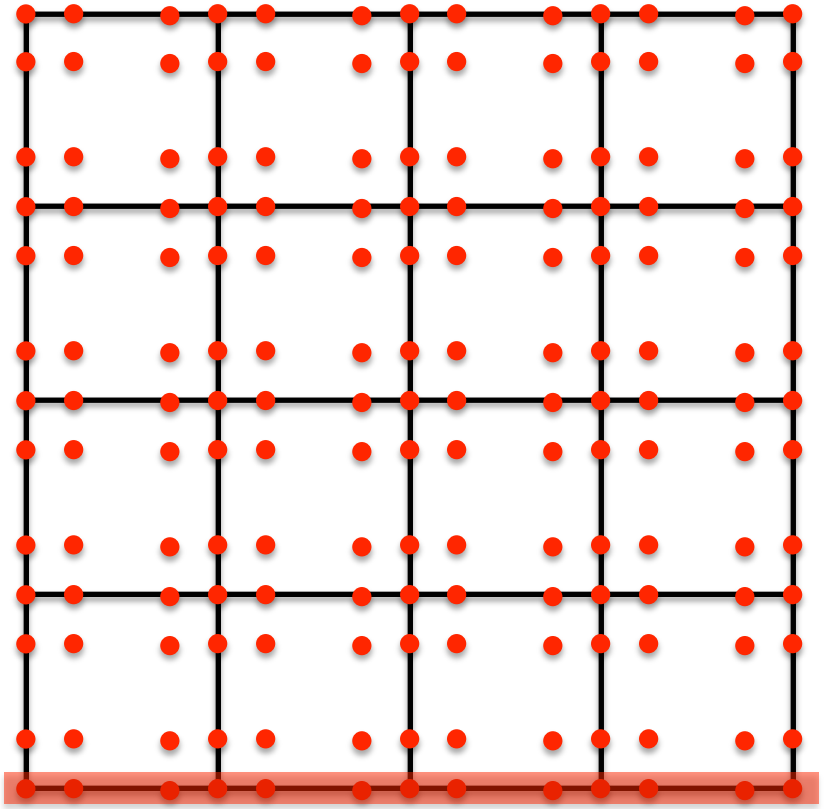
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel
  - For each “send cycle”
    - Send cycle over PCI-e (D2H)
    - MPI_Isend the cycle



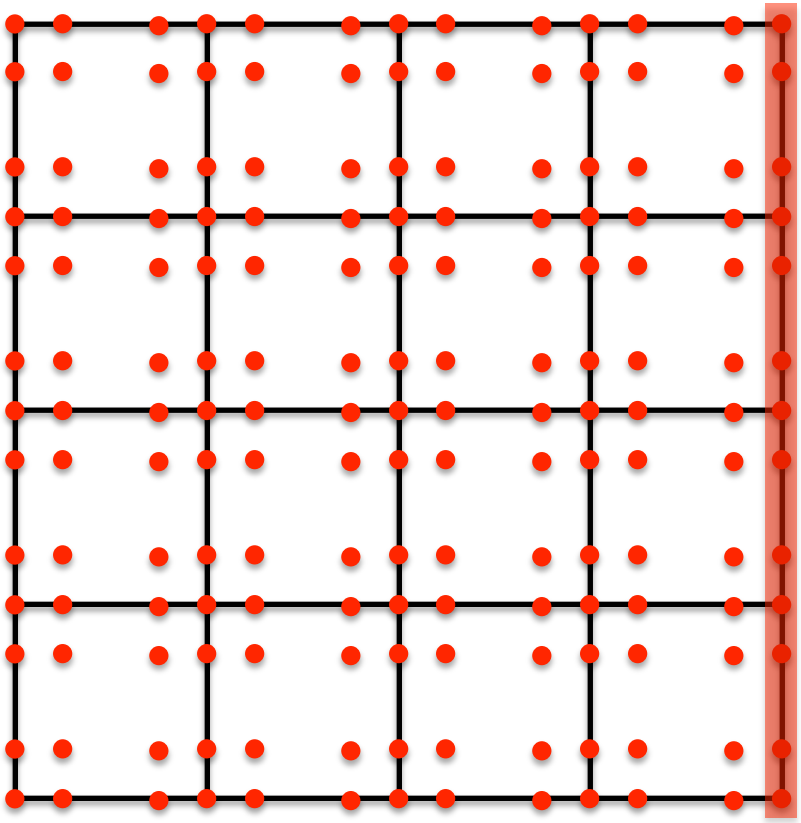
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel
  - For each “send cycle”
    - Send cycle over PCI-e (D2H)
    - MPI_Isend the cycle



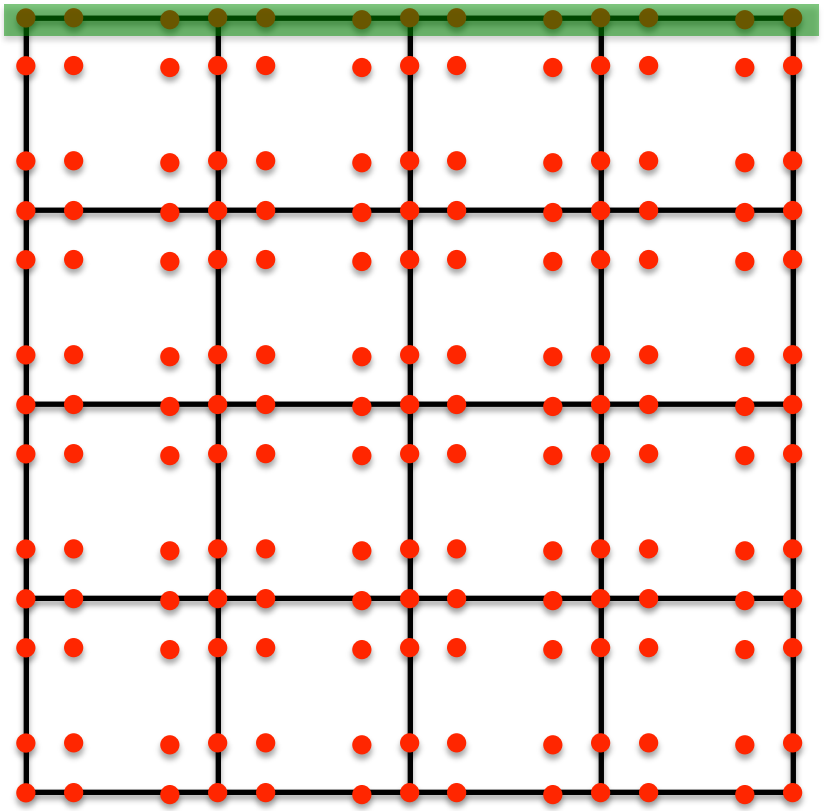
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel
  - For each “send cycle”
    - Send cycle over PCI-e (D2H)
    - MPI_Isend the cycle



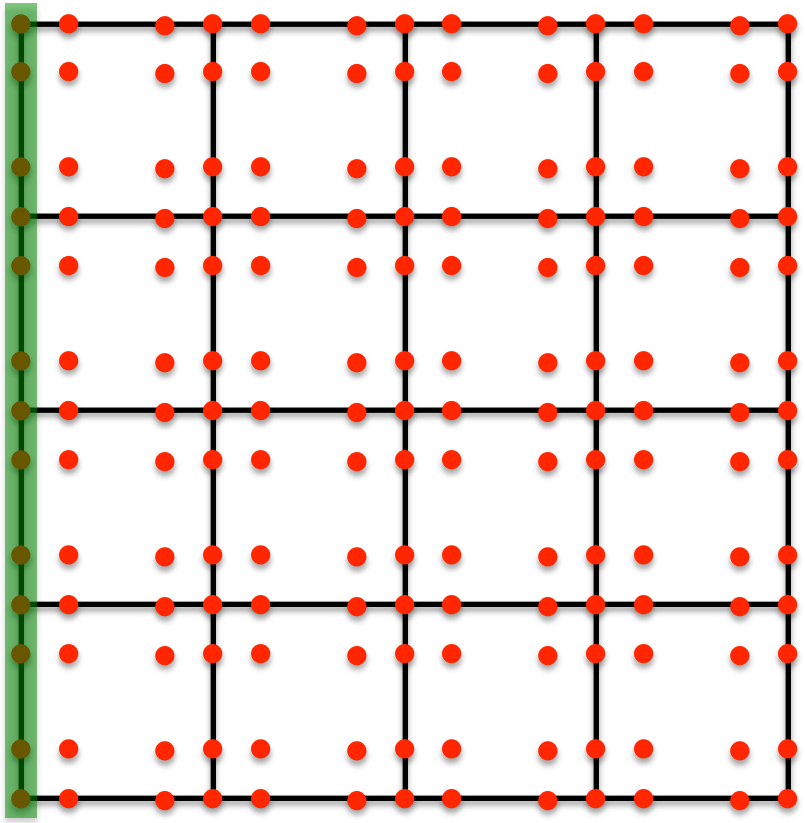
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel
  - For each “send cycle”
    - Send cycle over PCI-e (D2H)
    - MPI_Isend the cycle
  - For each “receive cycle”
    - MPI_Wait for the data
    - Send cycle over PCI-e (H2D)



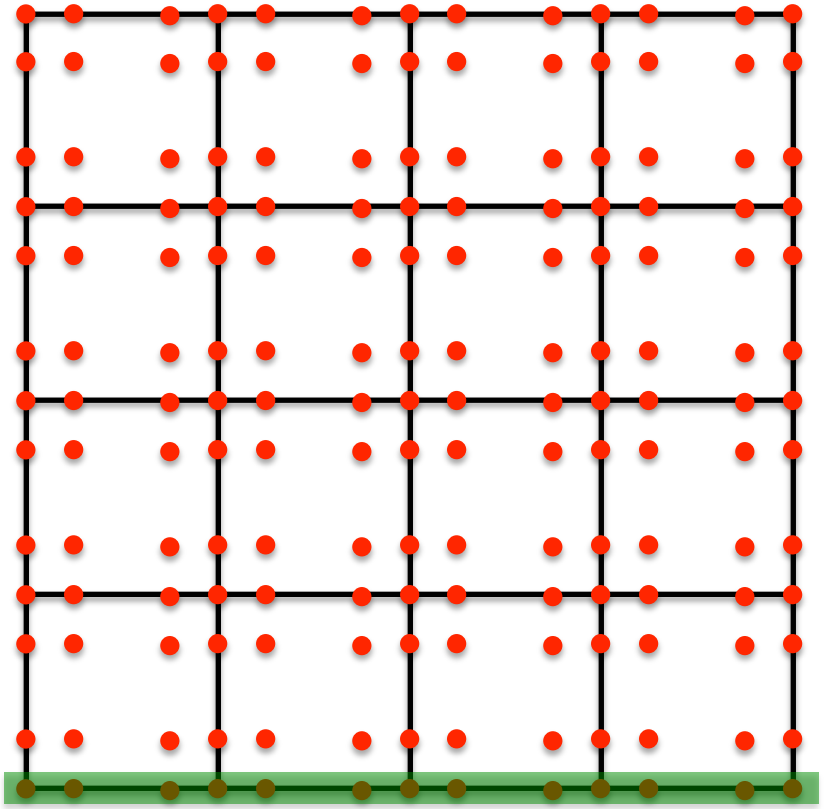
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel
  - For each “send cycle”
    - Send cycle over PCI-e (D2H)
    - MPI_Isend the cycle
  - For each “receive cycle”
    - MPI_Wait for the data
    - Send cycle over PCI-e (H2D)



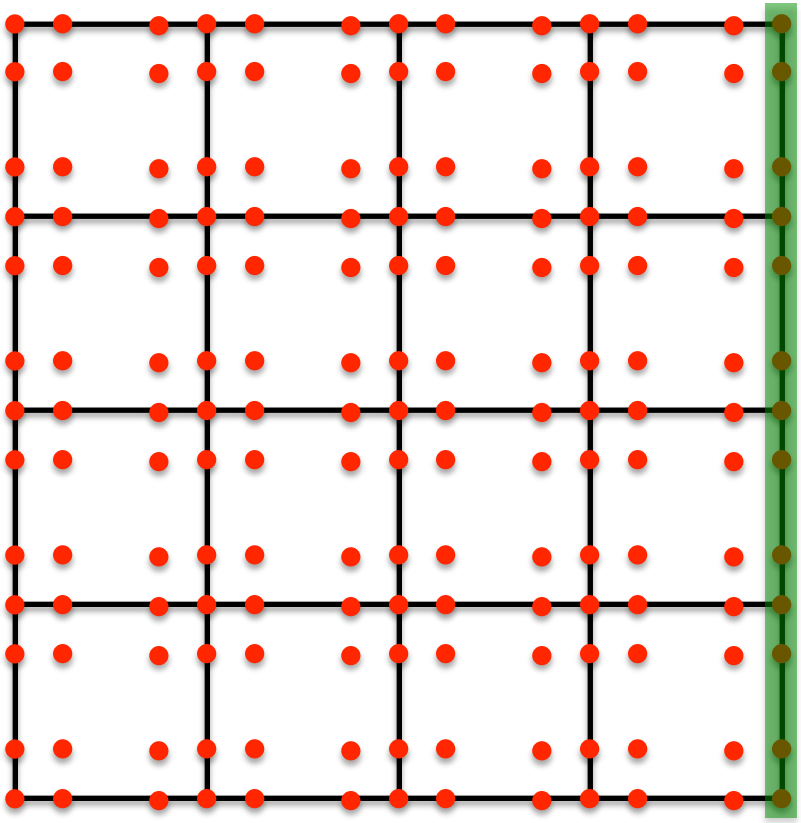
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel
  - For each “send cycle”
    - Send cycle over PCI-e (D2H)
    - MPI_Isend the cycle
  - For each “receive cycle”
    - MPI_Wait for the data
    - Send cycle over PCI-e (H2D)



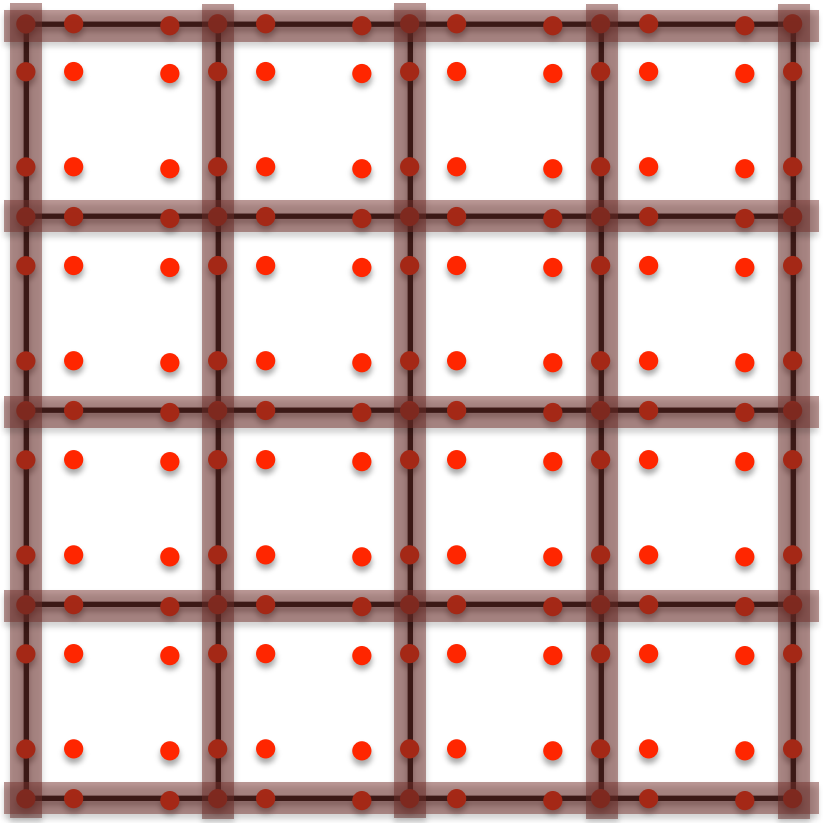
Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel
  - For each “send cycle”
    - Send cycle over PCI-e (D2H)
    - MPI_Isend the cycle
  - For each “receive cycle”
    - MPI_Wait for the data
    - Send cycle over PCI-e (H2D)



Original Pack/Exchange/Unpack

- Edge_pack ensures data for MPI is contiguous in buffer
- MPI communication occurs in “cycles”
- A cycle contains a contiguous data region for MPI
- Original pack/exchange/unpack
  - Pack all edges in a GPU Kernel
  - For each “send cycle”
    - Send cycle over PCI-e (D2H)
    - MPI_Isend the cycle
  - For each “receive cycle”
    - MPI_Wait for the data
    - Send cycle over PCI-e (H2D)
  - Unpack all edges in a GPU Kernel

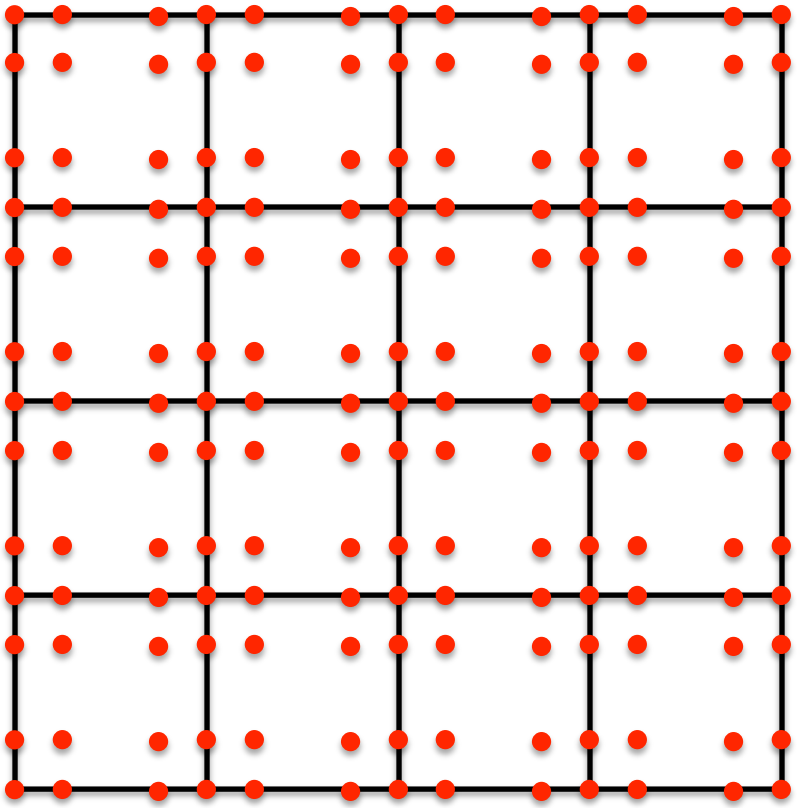


Optimizing Pack/Exchange/Unpack

- For a cycle, PCI-e D2H depends only on packing that cycle
  - Divide edge_pack into equal-sized cycles
    1. Find only the elements directly involved in each separate cycle
    2. Evenly divide remaining elements among the cycles
  - Associate each cycle with a unique CUDA stream
  - Launch each pack in its stream
  - After a cycle is packed, call async. PCI-e D2H in its Stream
- Edge_unpack at MPI boundaries requires all MPI to be finished
- However, internal unpacks can be done directly after packing

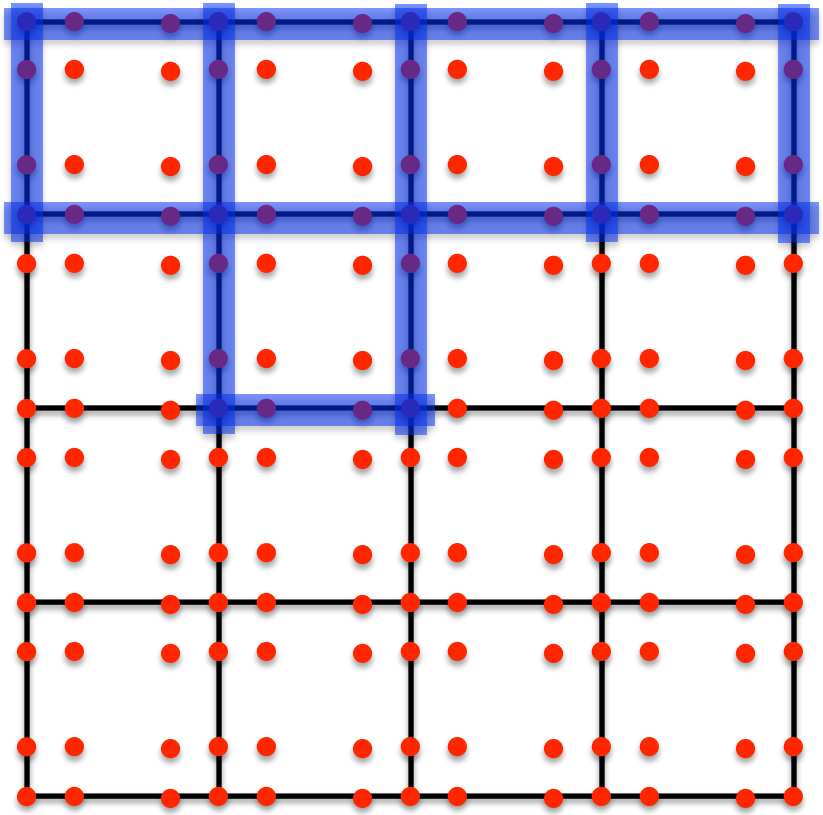
Porting Strategy: Pack/Exchange/Unpack

- For each cycle
  - Launch edge_pack kernel for the cycle in a unique stream
  - Call a cudaEventRecord for the stream's packing event



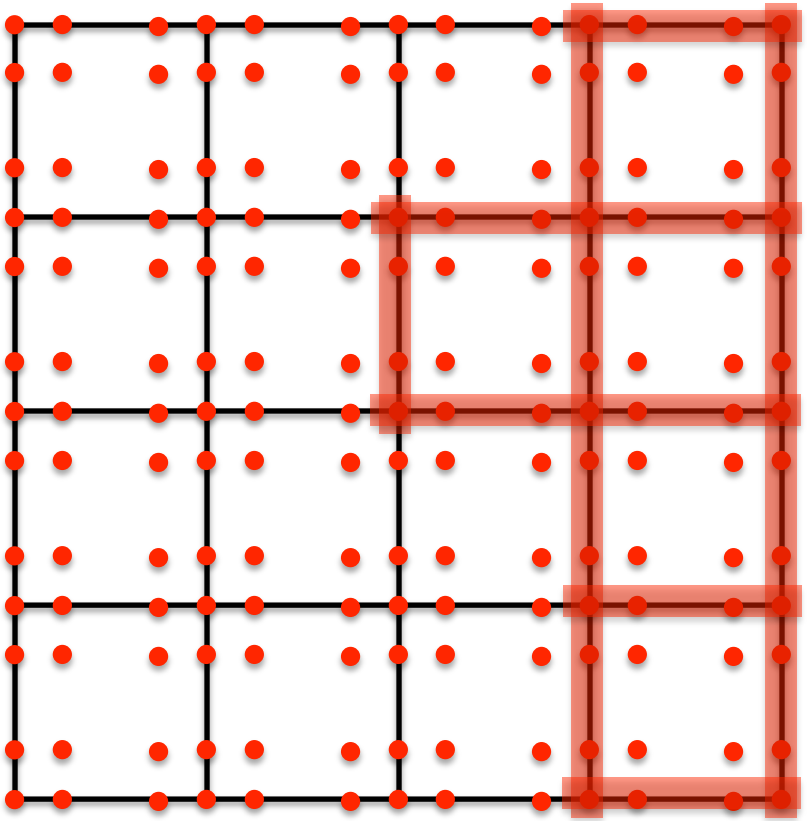
Porting Strategy: Pack/Exchange/Unpack

- For each cycle
  - Launch edge_pack kernel for the cycle in a unique stream
  - Call a cudaEventRecord for the stream's packing event



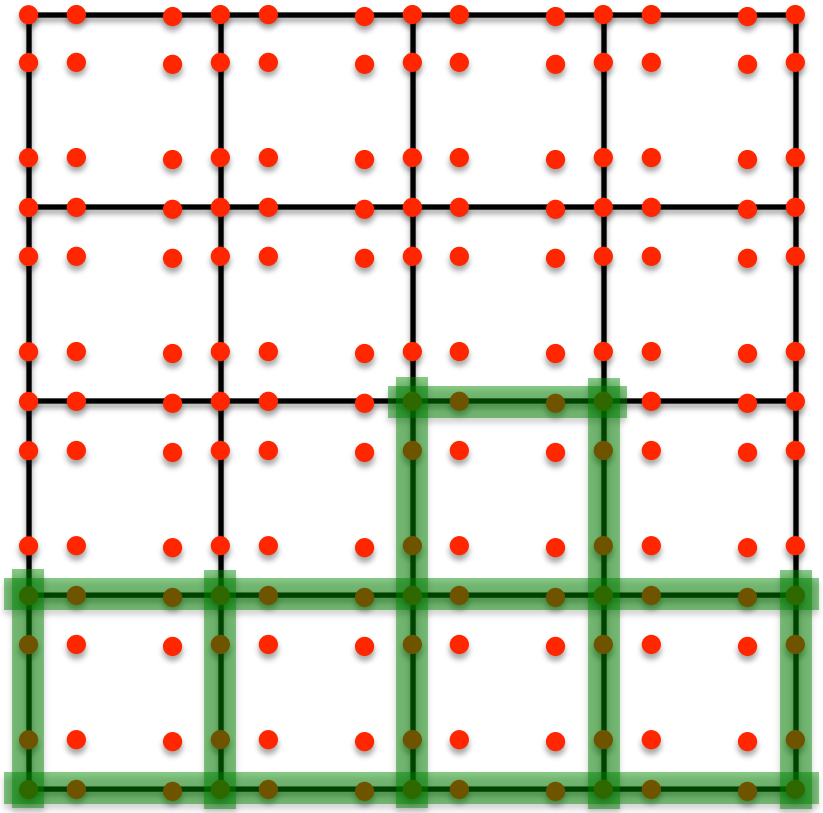
Porting Strategy: Pack/Exchange/Unpack

- For each cycle
  - Launch edge_pack kernel for the cycle in a unique stream
  - Call a cudaEventRecord for the stream's packing event



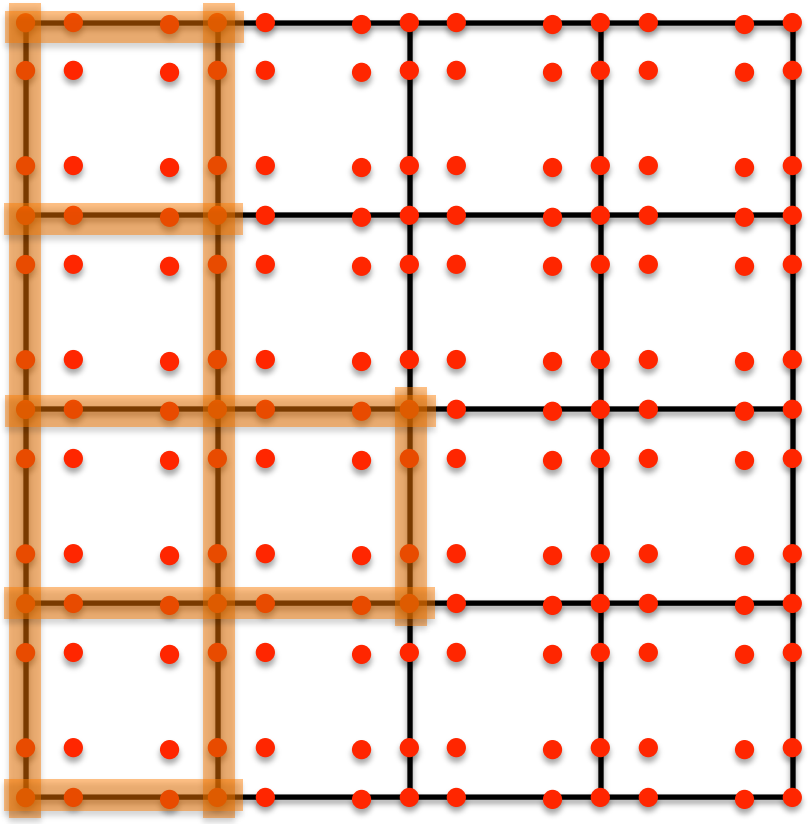
Porting Strategy: Pack/Exchange/Unpack

- For each cycle
  - Launch edge_pack kernel for the cycle in a unique stream
  - Call a cudaEventRecord for the stream's packing event



Porting Strategy: Pack/Exchange/Unpack

- For each cycle
  - Launch edge_pack kernel for the cycle in a unique stream
  - Call a cudaEventRecord for the stream's packing event



Other Important Porting Considerations

- Memory coalescing in kernels
  - Know how threads are accessing GPU DRAM

Think Differently About Threading

CPU Code

```
do ie=1,nelemd
 do q=1,qsize
  do k=1,nlev
   do j=1,np
    do i=1,np
     coefs(1,i,j,k,q,ie) = ...
     coefs(2,i,j,k,q,ie) = ...
     coefs(3,i,j,k,q,ie) = ...
```

GPU Code

```
ie = blockidx%y
q  = blockidx%x
k  = threadidx%z
j  = threadidx%y
i  = threadidx%x
coefs(1,i,j,k,q,ie) = ...
coefs(2,i,j,k,q,ie) = ...
coefs(3,i,j,k,q,ie) = ...
```

Think Differently About Threading

CPU Code

```
do ie=1,nelemd
 do q=1,qsize
  do k=1,nlev
   do j=1,np
    do i=1,np
     coefs(1,i,j,k,q,ie) = ...
     coefs(2,i,j,k,q,ie) = ...
     coefs(3,i,j,k,q,ie) = ...
```

Coded to respect cache locality

GPU Code

```
ie = blockidx%y
q  = blockidx%x
k  = threadidx%z
j  = threadidx%y
i  = threadidx%x
coefs(1,i,j,k,q,ie) = ...
coefs(2,i,j,k,q,ie) = ...
coefs(3,i,j,k,q,ie) = ...
```

Think Differently About Threading

CPU Code

```
do ie=1,nelemd
 do q=1,qsize
  do k=1,nlev
   do j=1,np
    do i=1,np
     coefs(1,i,j,k,q,ie) = ...
     coefs(2,i,j,k,q,ie) = ...
     coefs(3,i,j,k,q,ie) = ...
```

Coded to respect cache locality

GPU Code

```
ie = blockidx%y
q  = blockidx%x
k  = threadidx%z
j  = threadidx%y
i  = threadidx%x
coefs(1,i,j,k,q,ie) = ...
coefs(2,i,j,k,q,ie) = ...
coefs(3,i,j,k,q,ie) = ...
```

However, this will not coalesce to fill the DRAM bus

Think Differently About Threading

CPU Code

```
do ie=1,nelemd
 do q=1,qsize
  do k=1,nlev
   do j=1,np
    do i=1,np
     coefs(1,i,j,k,q,ie) = ...
     coefs(2,i,j,k,q,ie) = ...
     coefs(3,i,j,k,q,ie) = ...
```

GPU Code

```
ie = blockidx%y
q  = blockidx%x
k  = threadidx%z
j  = threadidx%y
i  = threadidx%x
coefs(1,i,j,k,q,ie) = ...
coefs(2,i,j,k,q,ie) = ...
coefs(3,i,j,k,q,ie) = ...
```

```
ie = blockidx%y
q  = blockidx%x
k  = threadidx%z
j  = threadidx%y
i  = threadidx%x
coefs(i,j,k,q,ie,1) = ...
coefs(i,j,k,q,ie,2) = ...
coefs(i,j,k,q,ie,3) = ...
```

Other Important Porting Considerations

- Memory coalescing in kernels
  - Know how threads are accessing GPU DRAM
- Use of shared memory
  - New vertical remap does nearly all computations on shared memory
  - Only accesses to DRAM are at start and end of a very large kernel
  - Hence, a 5.5x speed-up over CPU code for vertical remap
  - Watch out for banking conflicts

Other Important Porting Considerations

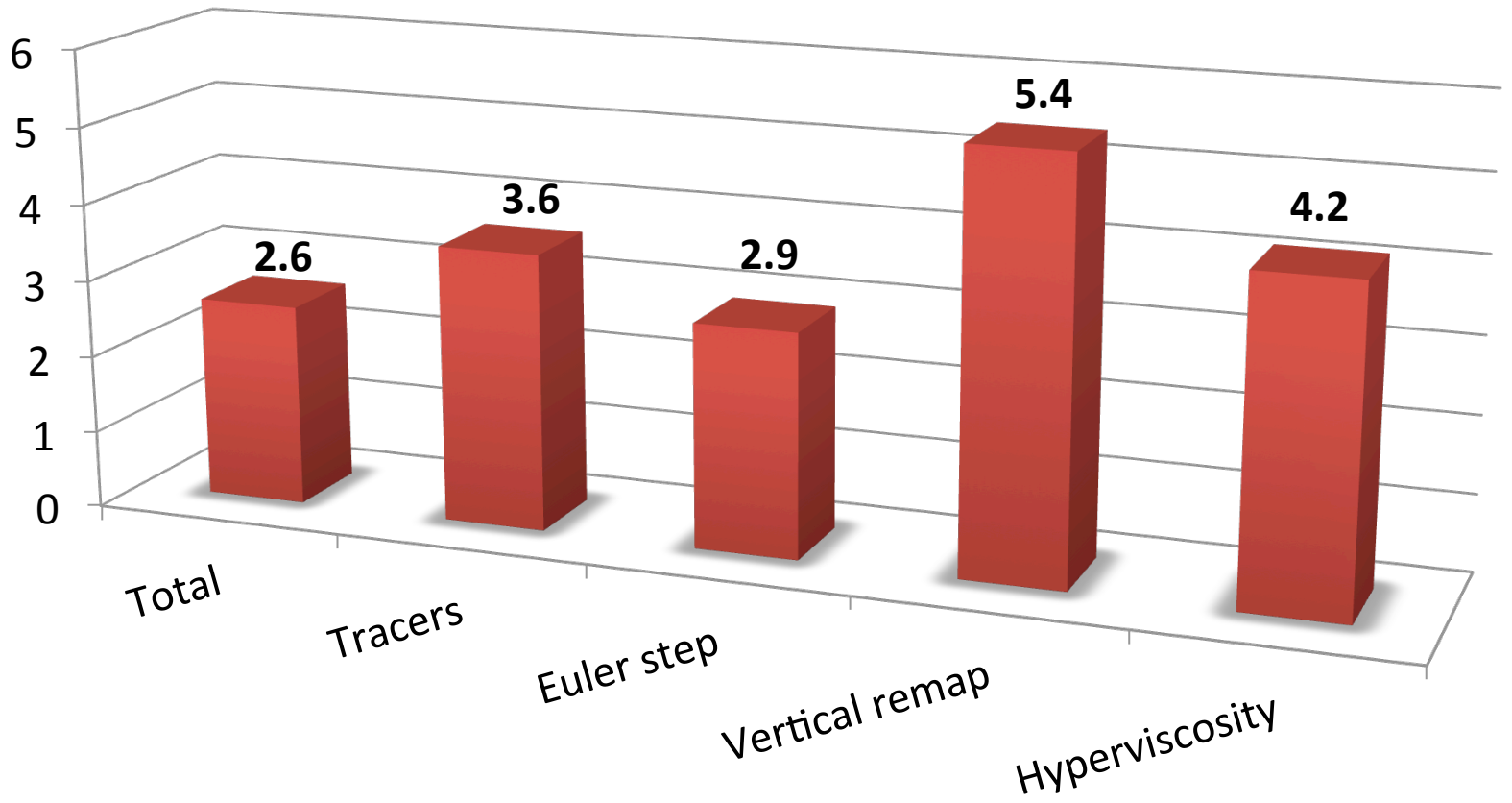
- Memory coalescing in kernels
  - Know how threads are accessing GPU DRAM
- Use of shared memory
  - New vertical remap does nearly all computations on shared memory
  - Only accesses to DRAM are at start and end of a very large kernel
  - Hence, a 5.5x speed-up over CPU code for vertical remap
  - Watch out for banking conflicts
- Overlapping kernels with CPU code execution
  - Any non-dependent sections of code overlapped with kernels
  - Computed & PCI-e copied data for future kernels during prior kernels

Other Important Porting Considerations

- Memory coalescing in kernels
  - Know how threads are accessing GPU DRAM
- Use of shared memory
  - New vertical remap does nearly all computations on shared memory
  - Only accesses to DRAM are at start and end of a very large kernel
  - Hence, a 5.5x speed-up over CPU code for vertical remap
  - Watch out for banking conflicts
- Overlapping kernels with CPU code execution
  - Any non-dependent sections of code overlapped with kernels
  - Computed & PCI-e copied data for future kernels during prior kernels
- PCI-e copies: consolidate if small, break up & pipeline if large

Speed-Up: Fermi GPU vs 1 Interlagos / Node

- Benchmarks performed on XK6 using end-to-end wall timers
- All PCI-e and MPI communication included



Computational Time-step barrier

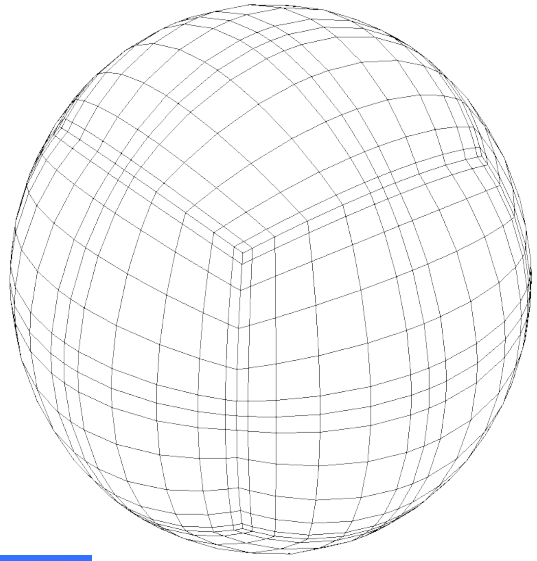
- Without it, climate simulation will hit the time-step barrier
  - Little benefit from higher scales of computing
  - Must choose between higher resolution or tolerable throughput
- Dramatically improve accuracy of highly-coupled models
- Dramatically accelerate spin-up of new simulations
- Potentially revolutionize climate simulation and other application areas with long time integration
- ***Crucial for climate simulation, promising for other fields***

Implicit versus Explicit

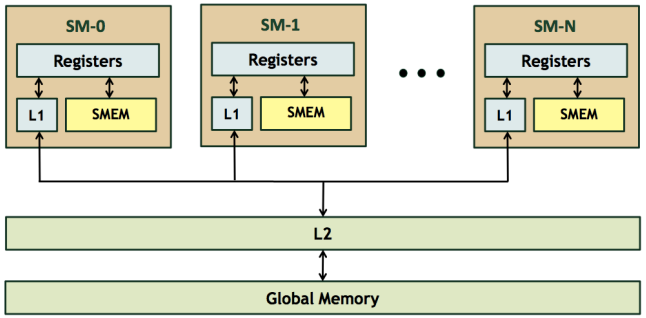
- *Explicit* time integration
  - Directly compute future state using derivatives at current or past times
  - Time step limited by numerical stability
- *Implicit* time integration
  - Define future state using derivates at that time, resulting in a system of (nonlinear) equations
  - Use (nonlinear) solver to determine future state
  - Numerically stable for large time steps
  - Time step limited by accuracy
  - Performance limited by solver

Accelerating Implicit solutions

Linking CAM-SE with third party software to create direct path to community algorithmic development

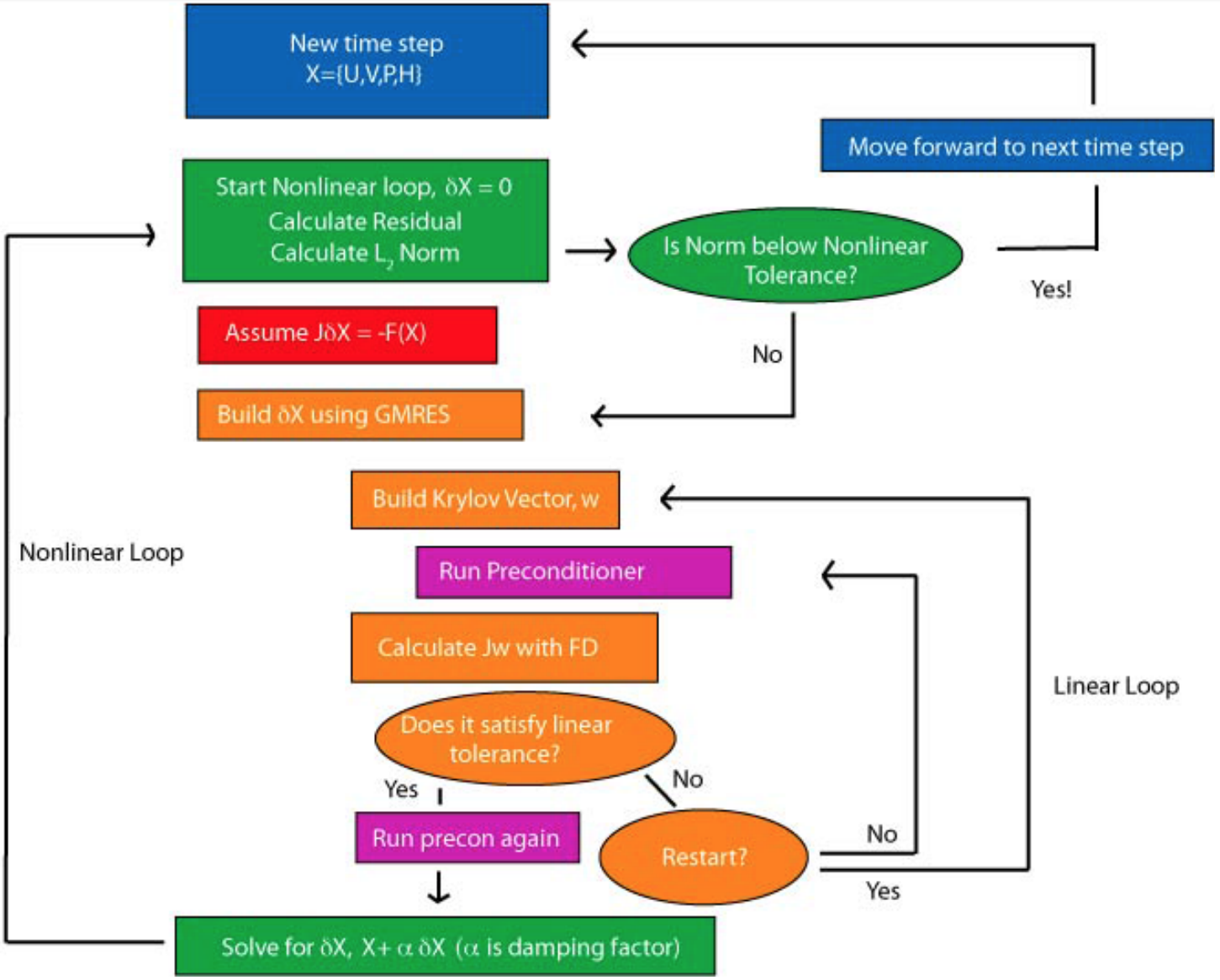


Linking CAM-SE with GPU community algorithmic development



Merging these separate efforts into the trunk of CAM-SE will allow accelerated time-stepping methods on hybrid architecture that keeps pass with community algorithmic development

Why Third Party Software?



JFNK Nested Algorithm

Modules available to CAM-SE

- **derived_type_mod**
  - type(derived_type) :: object (HOMME variables)
  - subroutine initialize(object, HOMME vars)
- **implicit_mod**
  - subroutine imp_solver(HOMME variables)
    - Set c_ptr -> f_ptr -> object
  - In c++, c_funptr points to calc_f in Fortran
  - subroutine calc_f(x(l), f(l), l, c_ptr) bind(C,name='user_sub_name')
- **noxlocainterface.cpp calls calc_f()**
- **noxlocainterface.hpp**

Next Steps – Implicit Framework

```
do n=1,nvar
 do ie=nets,nete
 do k=1,nlev
   do j=1,np
     do i=1,np
      lx = lx+1
      if (n==1) xstate(lx) = elem(ie)%state%v(i,j,1,k,np1)
      if (n==2) xstate(lx) = elem(ie)%state%v(i,j,2,k,np1)
      if (n==3) xstate(lx) = elem(ie)%state%p(i,j,k,np1)
     end do  !np
   end do  !np
 end do  !nlev
end do !ie
end do !nvar

call noxsolve(size(xstate), xstate, c_ptr_to_object, c_ptr_to_pre)

lx = 0
do n=1,nvar
 do ie=nets,nete
 do k=1,nlev
   do j=1,np
     do i=1,np
      lx = lx+1
      if (n==1) elem(ie)%state%v(i,j,1,k,np1)=xstate(lx)
      if (n==2) elem(ie)%state%v(i,j,2,k,np1)=xstate(lx)
      if (n==3) elem(ie)%state%p(i,j,k,np1)=xstate(lx)
     end do  !np
   end do  !np
 end do  !nlev
end do !ie
end do !nvar

subroutine residual(xstate, fx, nelemd, c_ptr_to_object) bind(C,name='calc_f')

  use ,intrinsic :: iso_c_binding
  use kinds, only : real_kind
  use dimensions_mod, only : np, nlev, nvar, nelem
```

Next Steps – GPU Framework

```
subroutine remap_velocityQ_launcher(n0,np1,dt,elem,hvcoord,nets,nete,compute_diagnostics,rkstage)
  use hybrid_mod, only: hybrid_t

  ! Setup Thread Partitioning
  blockdim = dim3(np,np,_EUL_BLK_)
  griddim = dim3(qsize_d,int(ceiling(dble(nete-nets+1)/dble(_EUL_BLK_))),1)

  if(compute_mean_flux==1 .and. prescribed_wind==0) use_mean_flux=.true.

  do ie=nets,nete
    do i=1,np
      do j=1,np
        do k=1,nlev
            dp_np1_h(i,j,k,ie) = (hvcoord%hyai(k+1) - hvcoord%hyai(k))*hvcoord%ps0 + &
                                 (hvcoord%hybi(k+1) - hvcoord%hybi(k))*elem(ie)%state%ps_v(i,j,np1)

          endif
        enddo
      enddo
    enddo
  enddo
  ierr = cudaMemcpy(dp_star_d,dp_star_h,size(dp_star_h),cudaMemcpyHostToDevice)
  ierr = cudaMemcpy(dp_np1_d ,dp_np1_h ,size(dp_np1_h ),cudaMemcpyHostToDevice)

  ierr = cudaDeviceSetCacheConfig(cudaFuncCachePreferL1)
  call remap_velocityQ_kernel_new<<<griddim,blockdim>>>(n0,np1,dt,device_arrays%Qdp,nets,nete,comput
  ierr = cudaThreadSynchronize()

  call t_stopf('remap_velocityQ')

end subroutine remap_velocityQ_launcher
```

Next Steps – Linked Framework

```
      do n=1,nvar
       do ie=nets,nete
       do k=1,nlev
         do j=1,np
           do i=1,np
            lx = lx+1
            if (n==1) xstate(lx) = elem(ie)%state%v(i,j,1,k,np1)
            if (n==2) xstate(lx) = elem(ie)%state%v(i,j,2,k,np1)
            if (n==3) xstate(lx) = elem(ie)%state%p(i,j,k,np1)
           end do  !np
         end do  !np
       end do  !nlev
      end do !ie
      end do !nvar

   call noxsolve(size(xstate), xstate, c_ptr_to_object, c_ptr_to_pre)

      lx = 0
      do n=1,nvar
       do ie=nets,nete
       do k=1,nlev
         do j=1,np
           do i=1,np
            lx = lx+1
            if (n==1) elem(ie)%state%v(i,j,1,k,np1)=xstate(lx)
            if (n==2) elem(ie)%state%v(i,j,2,k,np1)=xstate(lx)
            if (n==3) elem(ie)%state%p(i,j,k,np1)=xstate(lx)
           end do  !np
         end do  !np
       end do  !nlev
      end do !ie
      end do !nvar
  subroutine residual(xstate, fx, nelemd, c_ptr_to_object) bind(C,name='calc_f')

    use ,intrinsic :: iso_c_binding
    use kinds, only : real_kind
    use dimensions_mod, only : np, nlev, nvar, nelem
```

```
  subroutine remap_velocityQ_launcher(n0,np1,dt,elem,hvcoord,nets,nete,compute_diagnostics,rkstage)
    use hybrid_mod, only: hybrid_t

    ! Setup Thread Partitioning
    blockdim = dim3(np,np,_EUL_BLK_)
    griddim = dim3(qsize_d,int(ceiling(dble(nete-nets+1)/dble(_EUL_BLK_))),1)

    if(compute_mean_flux==1 .and. prescribed_wind==0) use_mean_flux=.true.

    do ie=nets,nete
      do i=1,np
        do j=1,np
          do k=1,nlev
              dp_np1_h(i,j,k,ie) = (hvcoord%hyai(k+1) - hvcoord%hyai(k))*hvcoord%ps0 + &
                                   (hvcoord%hybi(k+1) - hvcoord%hybi(k))*elem(ie)%state%ps_v(i,j,np1)

            endif
          enddo
        enddo
      enddo
    enddo
    ierr = cudaMemcpy(dp_star_d,dp_star_h,size(dp_star_h),cudaMemcpyHostToDevice)
    ierr = cudaMemcpy(dp_np1_d ,dp_np1_h ,size(dp_np1_h ),cudaMemcpyHostToDevice)

    ierr = cudaDeviceSetCacheConfig(cudaFuncCachePreferL1)
    call remap_velocityQ_kernel_new<<<griddim,blockdim>>>(n0,np1,dt,device_arrays%Qdp,nets,nete,comput
    ierr = cudaThreadSynchronize()

    call t_stopf('remap_velocityQ')

  end subroutine remap_velocityQ_launcher
```

Merge framework so that array state is manipulated once and ‘calc_f’ executes on the GPU

Future Efforts