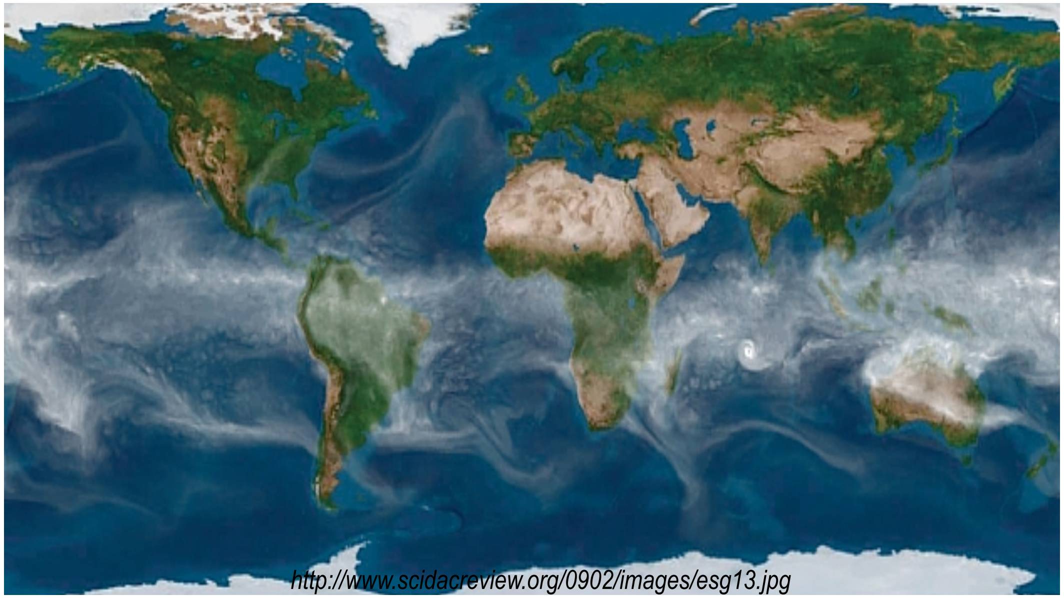
Porting The Spectral Element Community Atmosphere Model (CAM-SE) To Hybrid GPU Platforms



Matthew Norman : ORNL
Jeffrey Larkin : Cray
Richard Archibald : ORNL
Valentine Anantharaj : ORNL
Ilene Carpenter : NREL
Paulius Micikevicius : Nvidia
Katherine Evans : ORNL

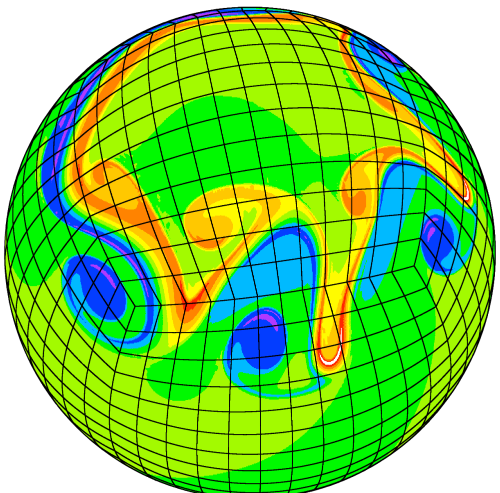
Titan Workshop

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



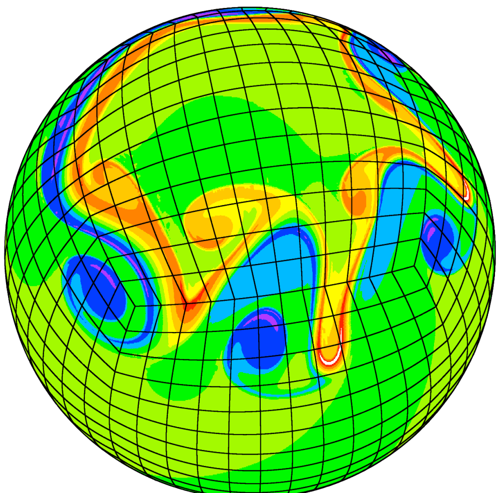
http://esse.engin.umich.edu/groups/admg/dcmip/jablonowski_cubed_sphere_vorticity.png

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



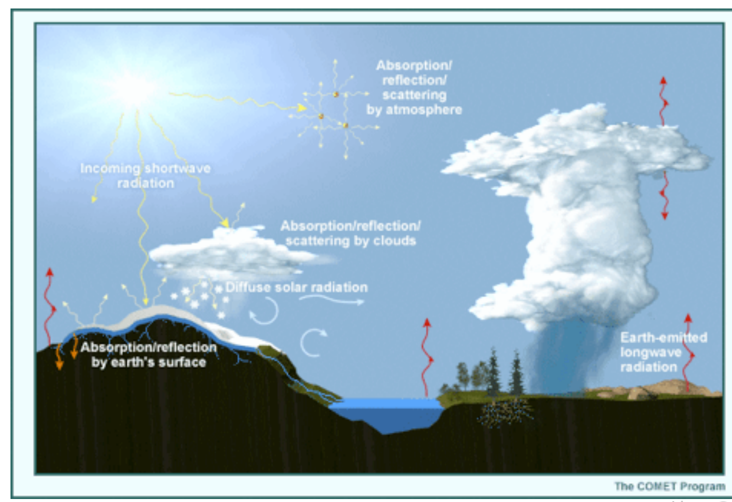
http://esse.engin.umich.edu/groups/admg/dcmip/jablonowski_cubed_sphere_vorticity.png

Dynamical Core

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

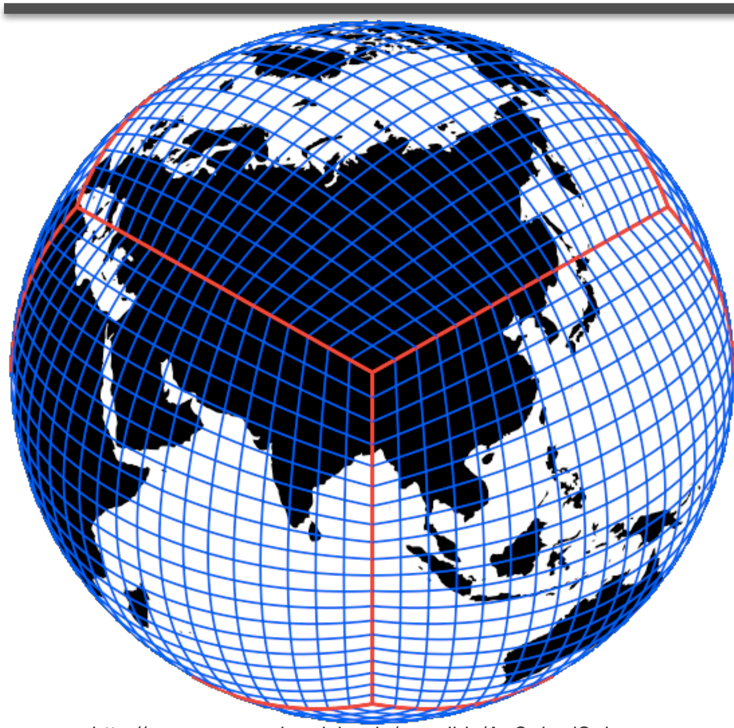
Physics Packages

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



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

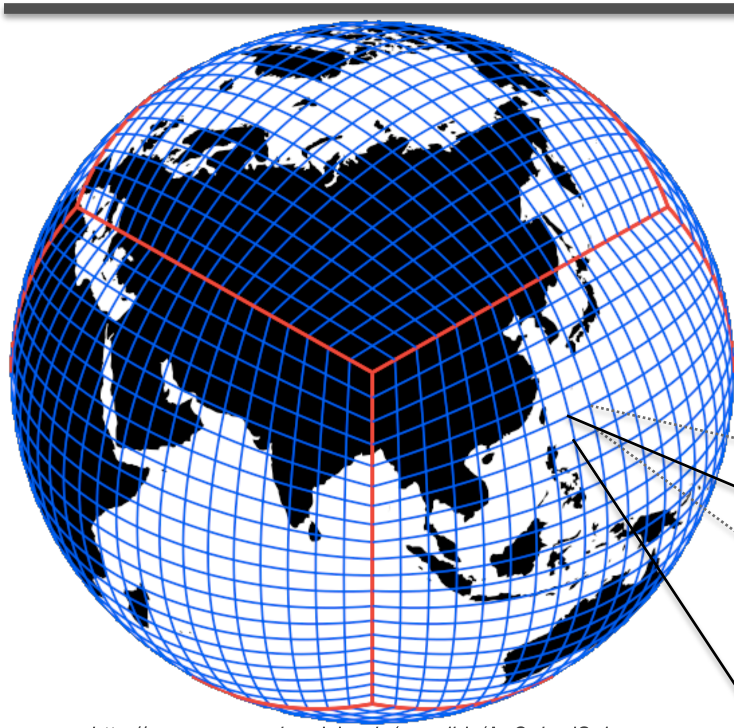
Gridding, Numerics, & Target Run



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

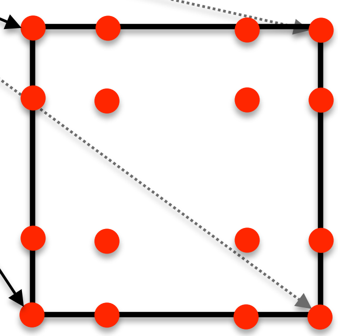
- Cubed-Sphere + Spectral Element
- Each cube panel divided into elements

Gridding, Numerics, & Target Run

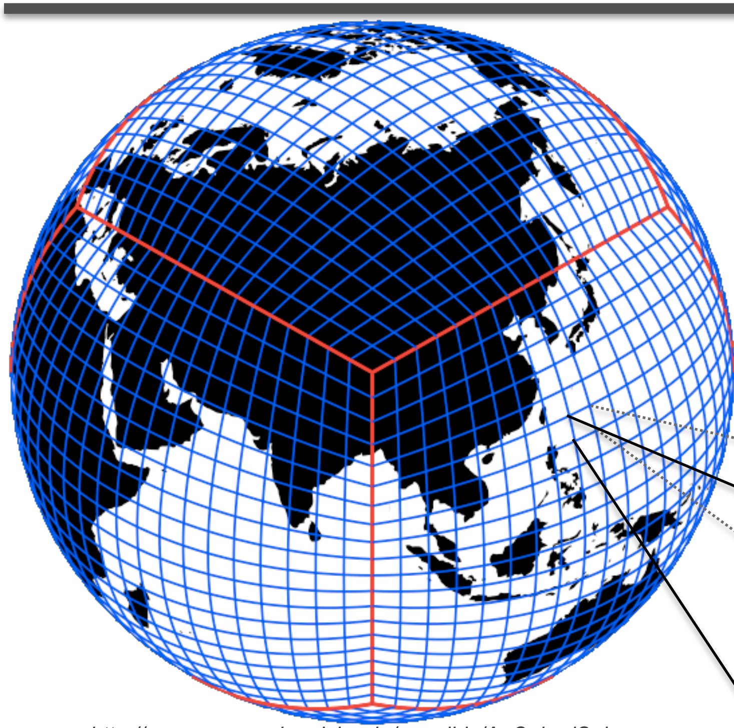


http://www-personal.umich.edu/~paullric/A_CubedSphere.png

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

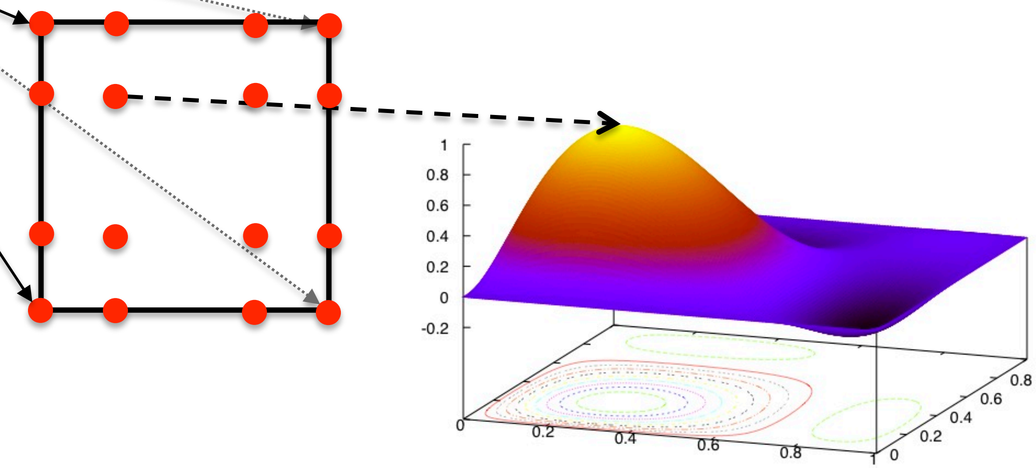


Gridding, Numerics, & Target Run

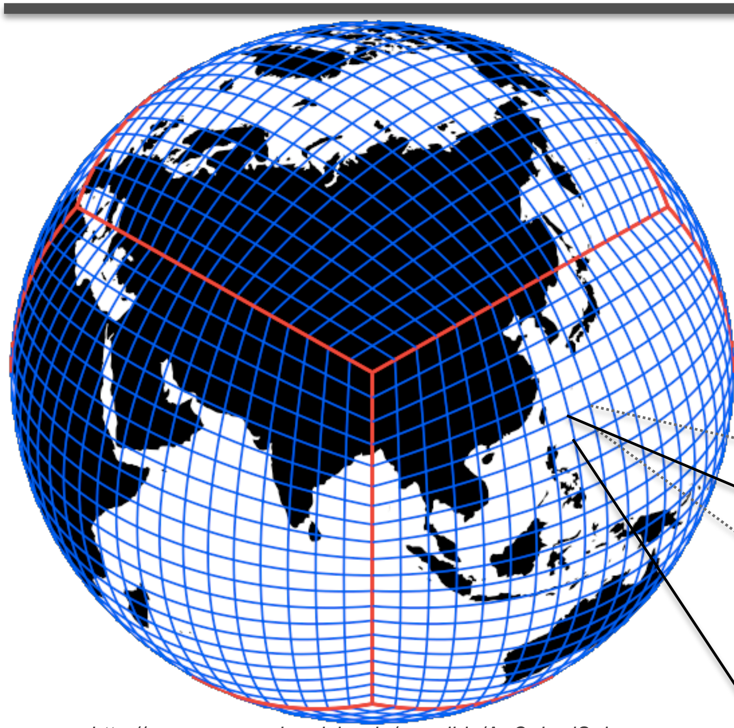


http://www-personal.umich.edu/~paullric/A_CubedSphere.png

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

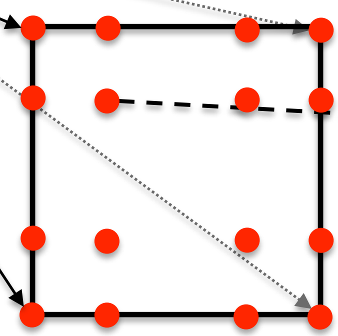


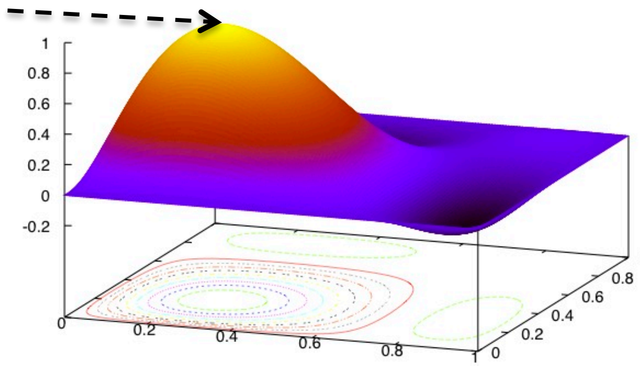
Gridding, Numerics, & Target Run



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





Used CUDA FORTRAN from PGI

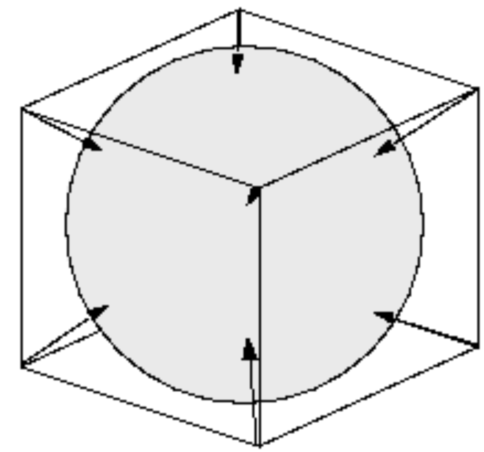
OACC Directives: Better software engineering option moving forward

Target 14km Simulations

- 16 billion degrees of freedom

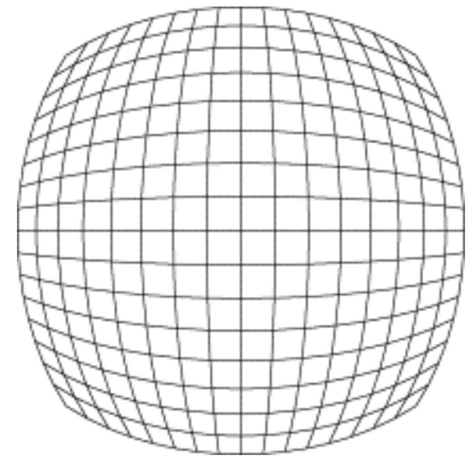
Target 14km Simulations

- 16 billion degrees of freedom
  - 6 cube panels



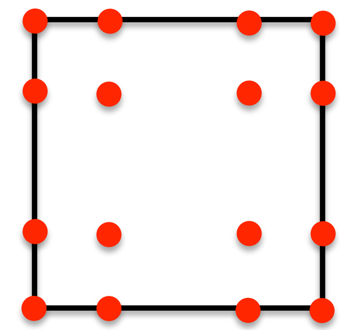
Target 14km Simulations

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



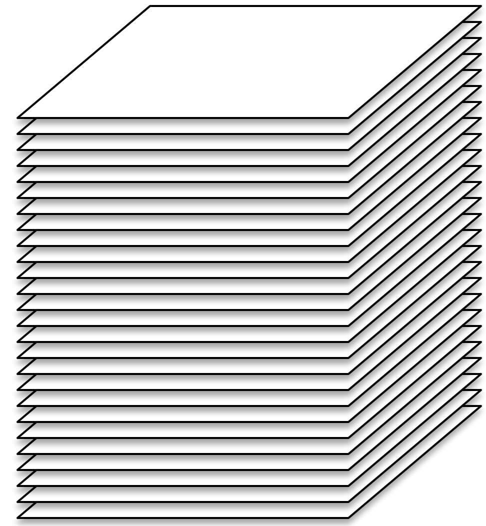
Target 14km Simulations

- 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

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

$$H_2O\ ,\ CO_2\ ,\ O_3\ ,\ CH_4\ ,\ \ldots$$

Target 14km Simulations

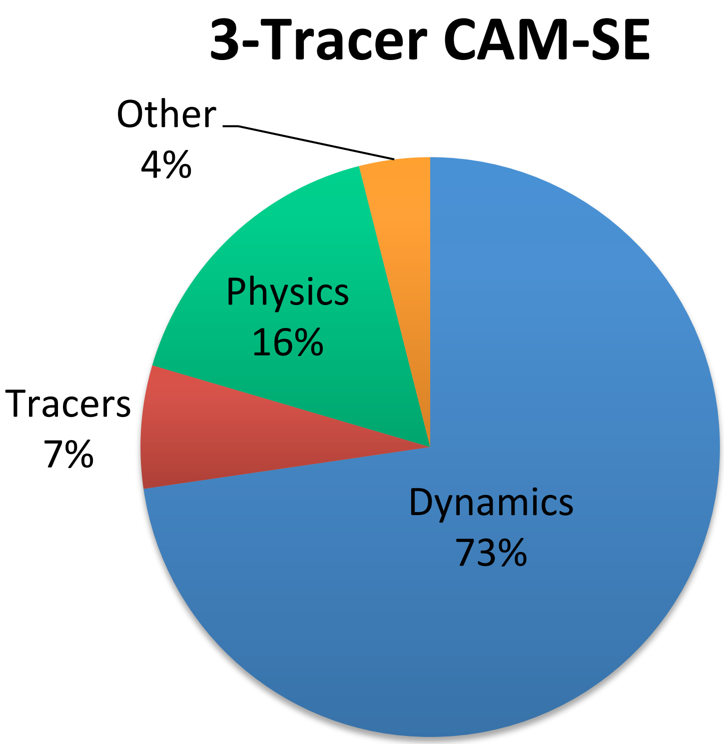
- 16 billion degrees of freedom
  - 6 cube panels
  - 240 x 240 columns of elements per panel
  - 4 x 4 basis functions per element
  - 26 vertical levels
  - 110 prognostic variables
- Scaled to 14,400 XT5 nodes with 60% parallel efficiency

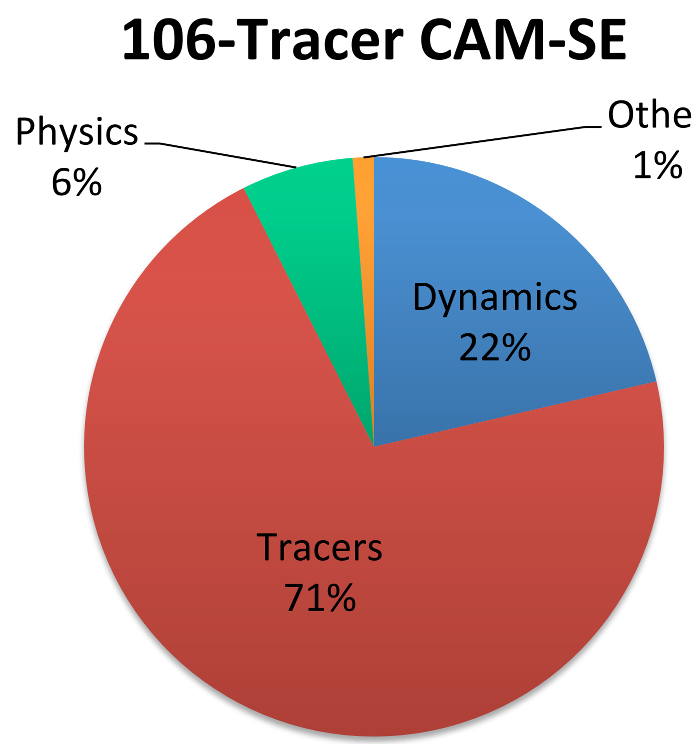
Target 14km Simulations

- 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 (XT5, 14km, 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

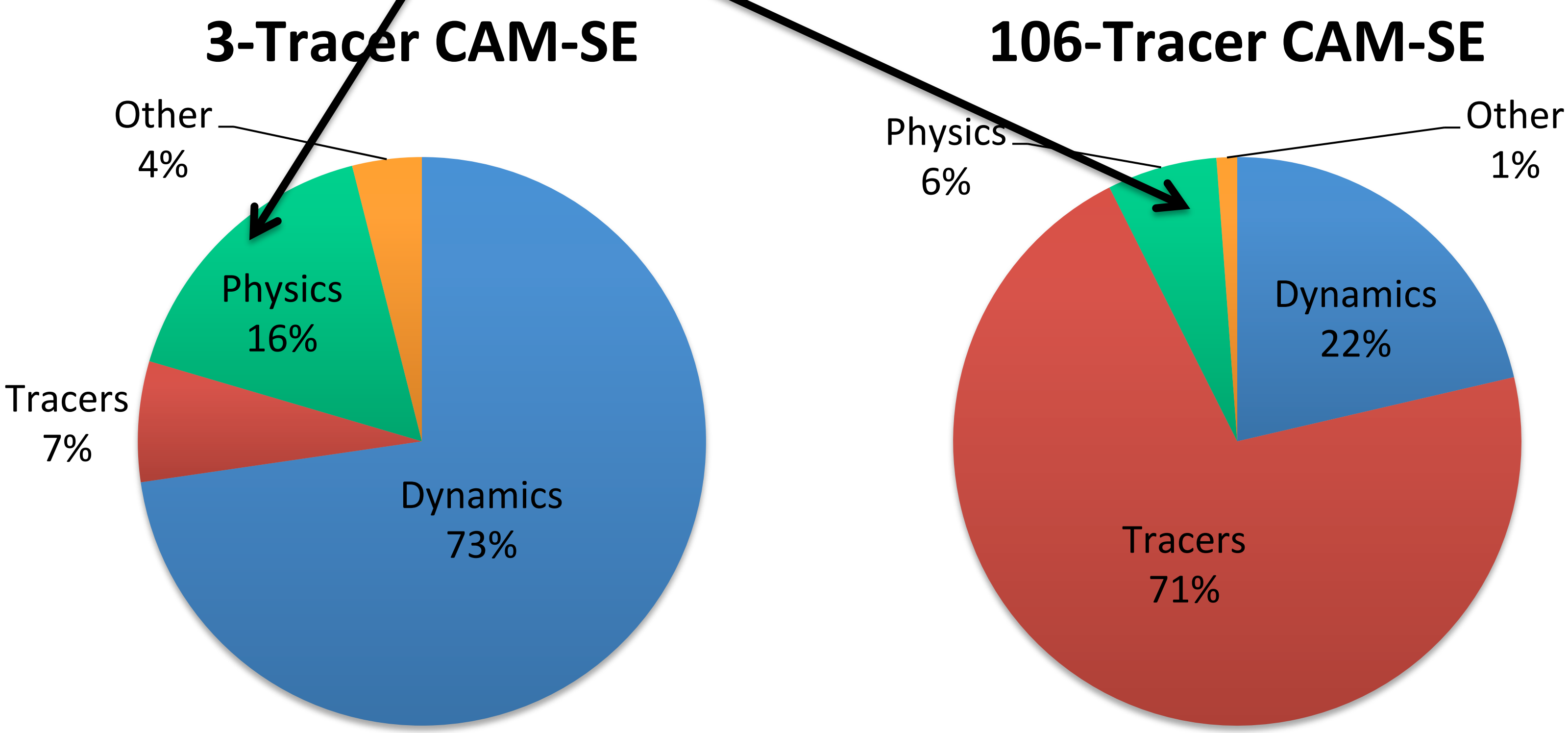




CAM-SE Profile (XT5, 14km, 14K Nodes)

- Original CAM-SE used 3 tracers (20% difficult to port)
- Mozart ... 06 tracers (7% difficult to port)
  - Cent ... h mostly data-parallel routines

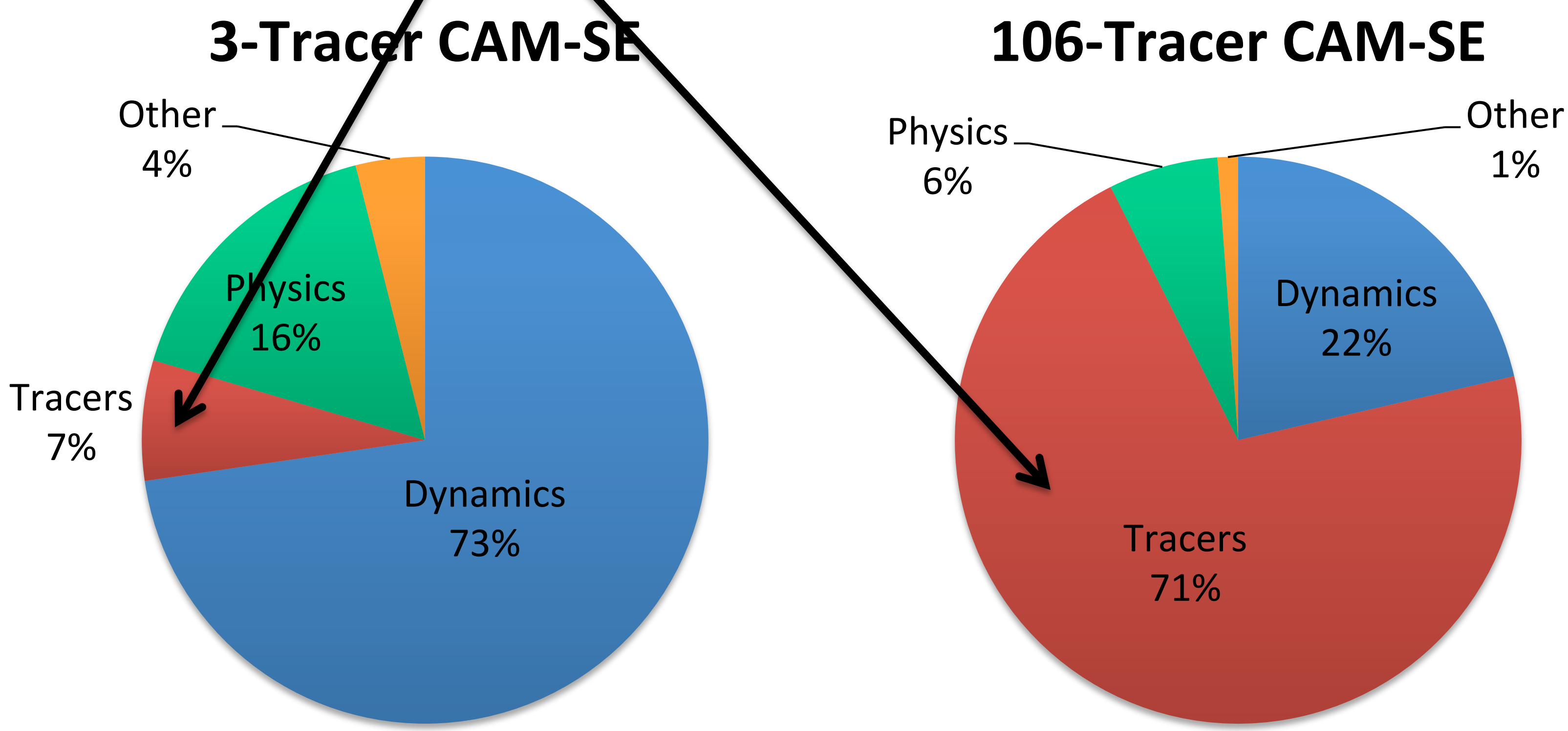
Not Portable



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

- Original CAM-SE used 3 tracers (20% difficult to port)
- Mozart ... 06 tracers (7% difficult to port)
  - Cent... mostly data-parallel routines

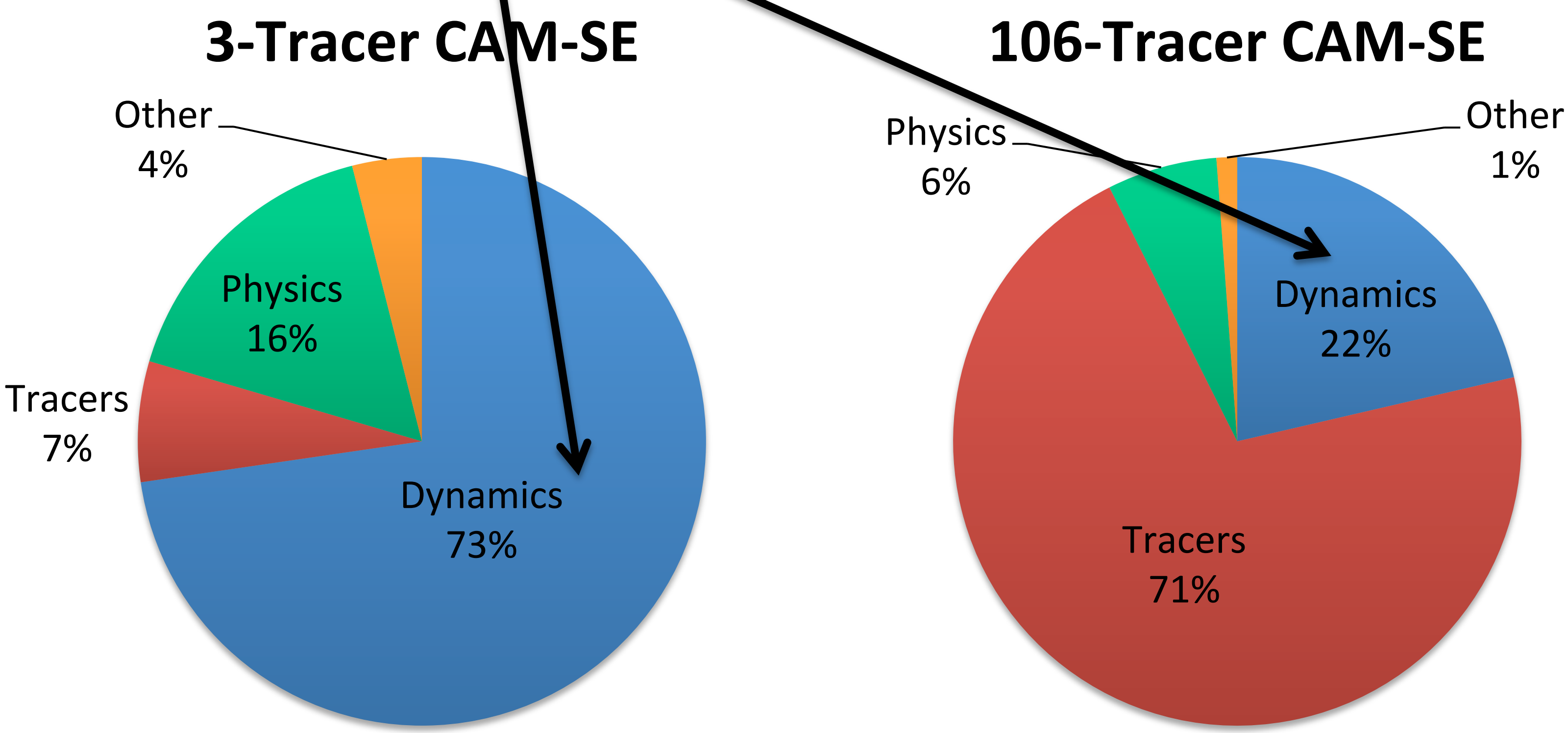
Very Easy To Port



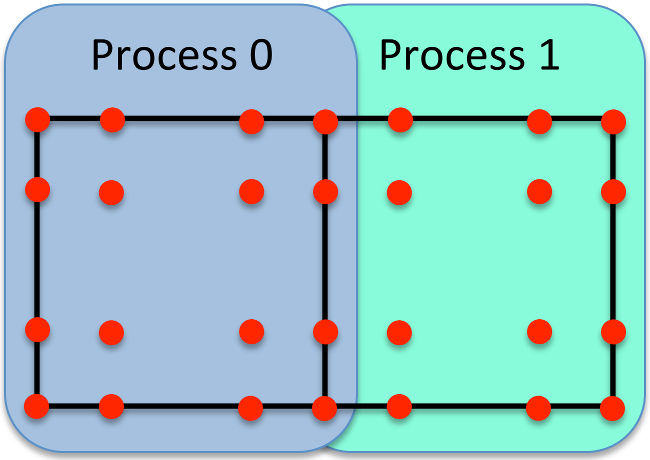
CAM-SE Profile (XT5, 14km, 14K Nodes)

- Original CAM-SE used 3 tracers (20% difficult to port)
- Mozart … 06 tracers (7% difficult to port)
  - Cent… mostly data-parallel routines

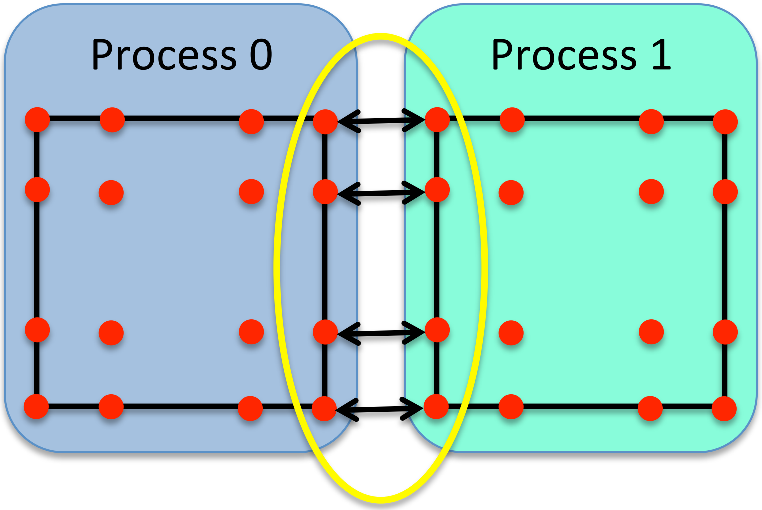
Fairly Easy To Port



Communication Between Elements



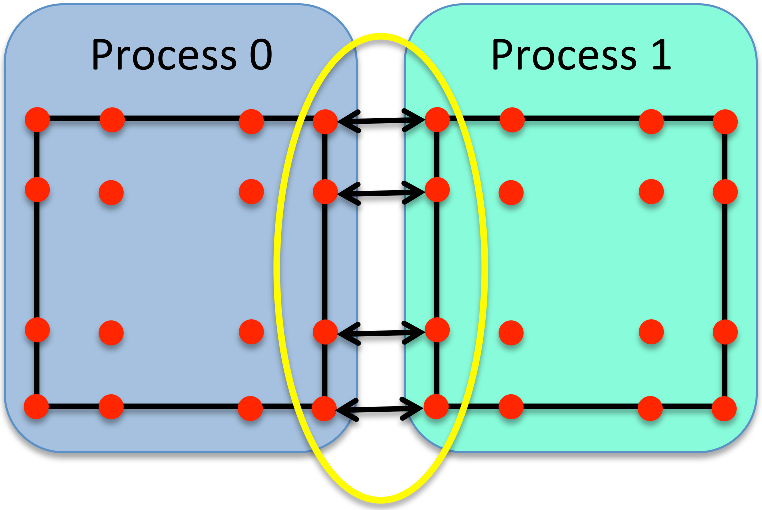
Communication Between Elements



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

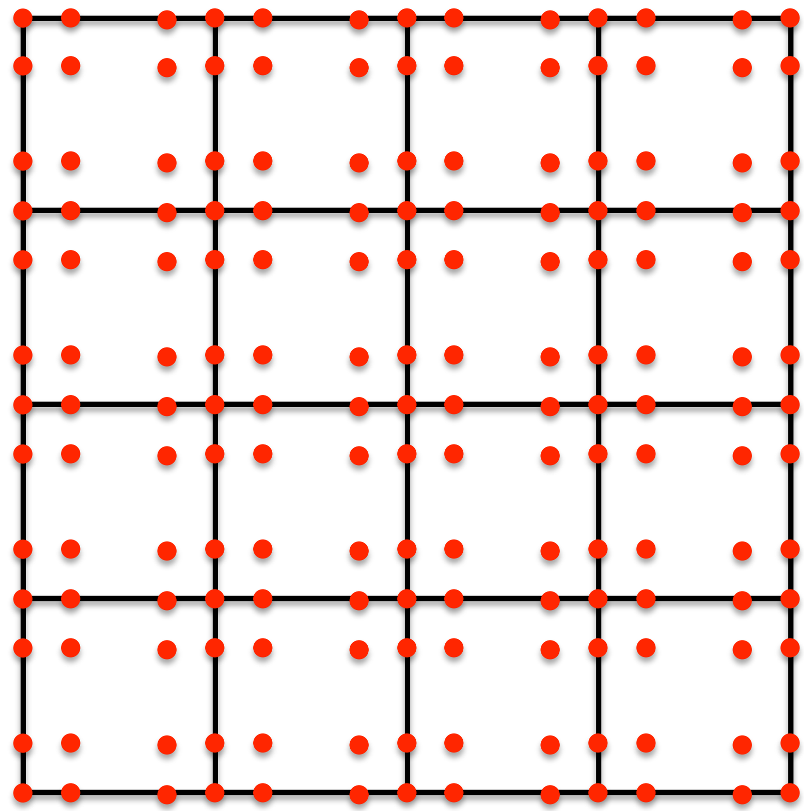
Edge_pack: pack <u>all</u> element edges into process-wide buffer. Data sent over MPI are contiguous in buffer.

Bndry_exchange: Send & receive data at domain decomposition boundaries

Edge_unpack: Perform a weighted sum for data at <u>all</u> element edges.

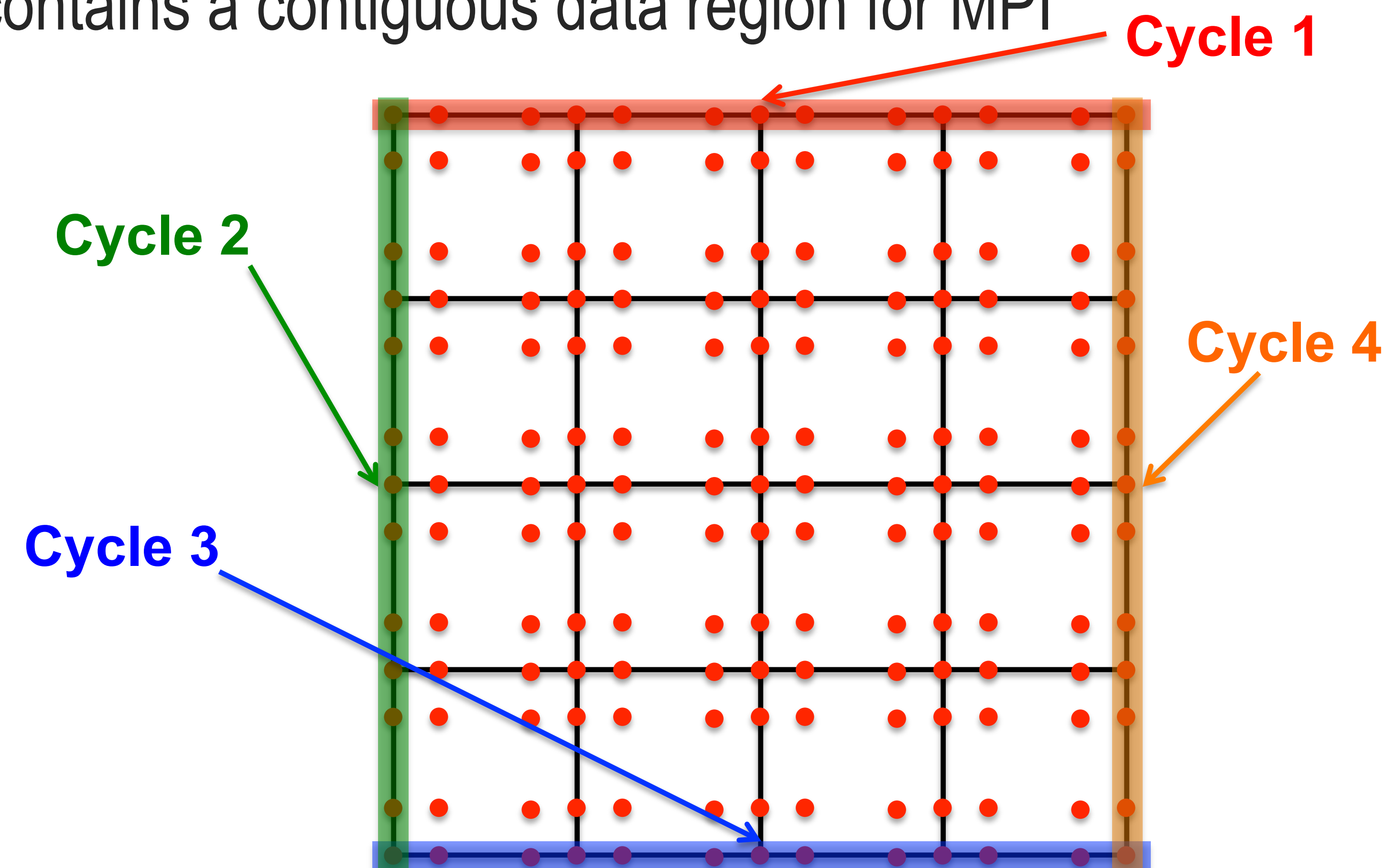
Original Pack/Exchange/Unpack

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



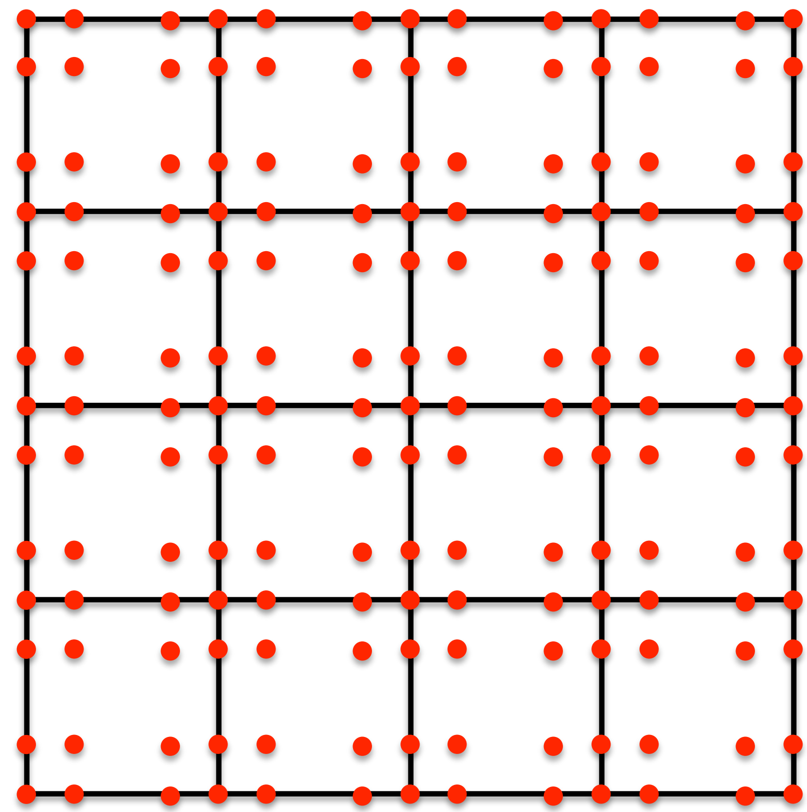
Original Pack/Exchange/Unpack

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



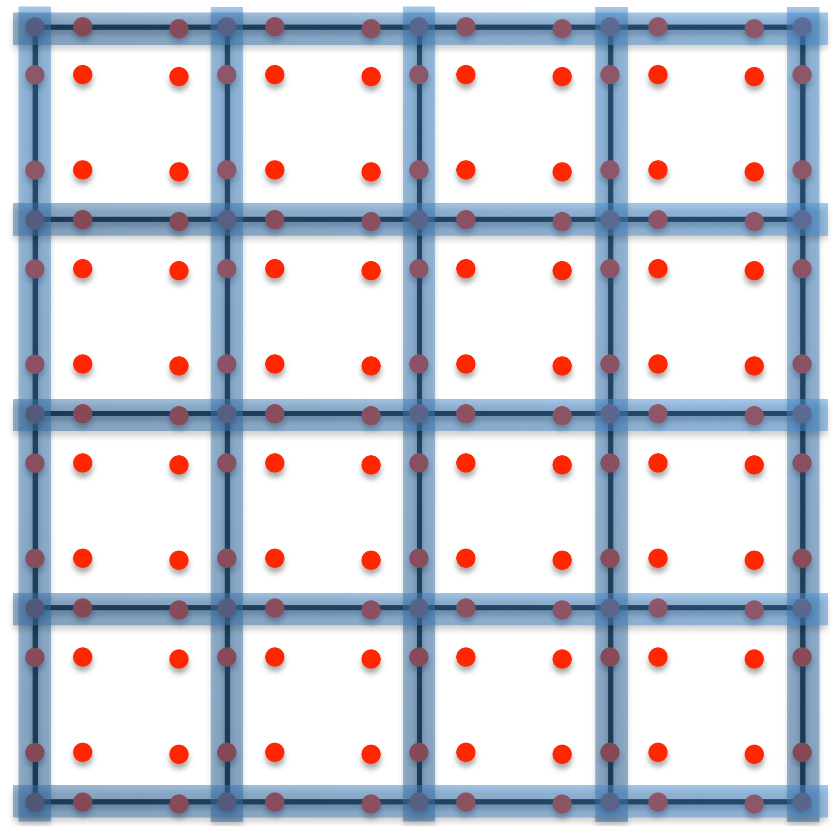
Original Pack/Exchange/Unpack

- 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



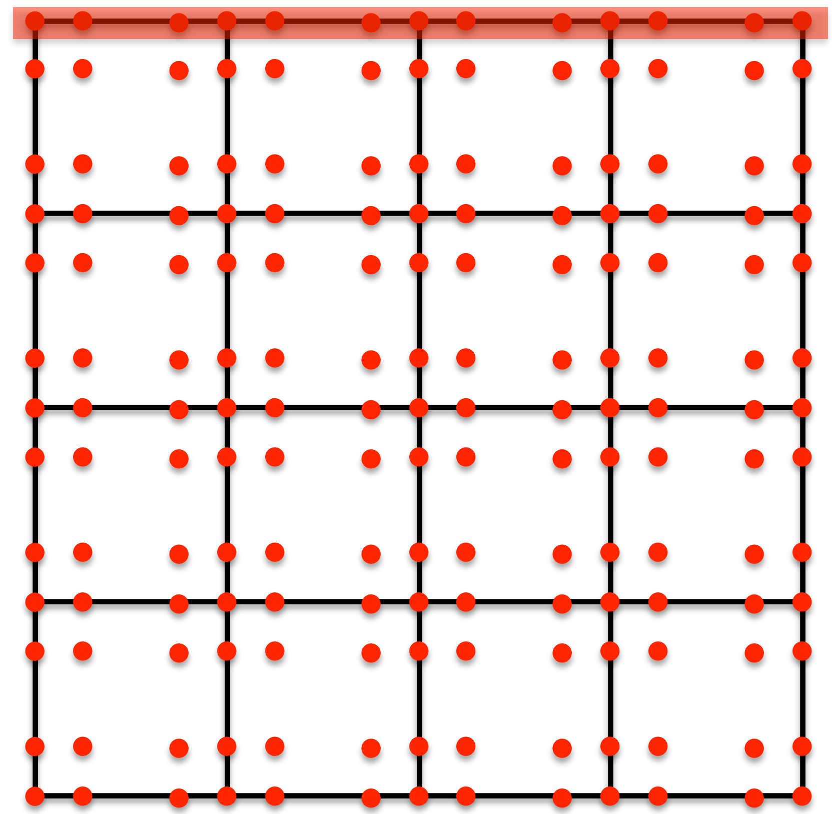
Original Pack/Exchange/Unpack

- 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



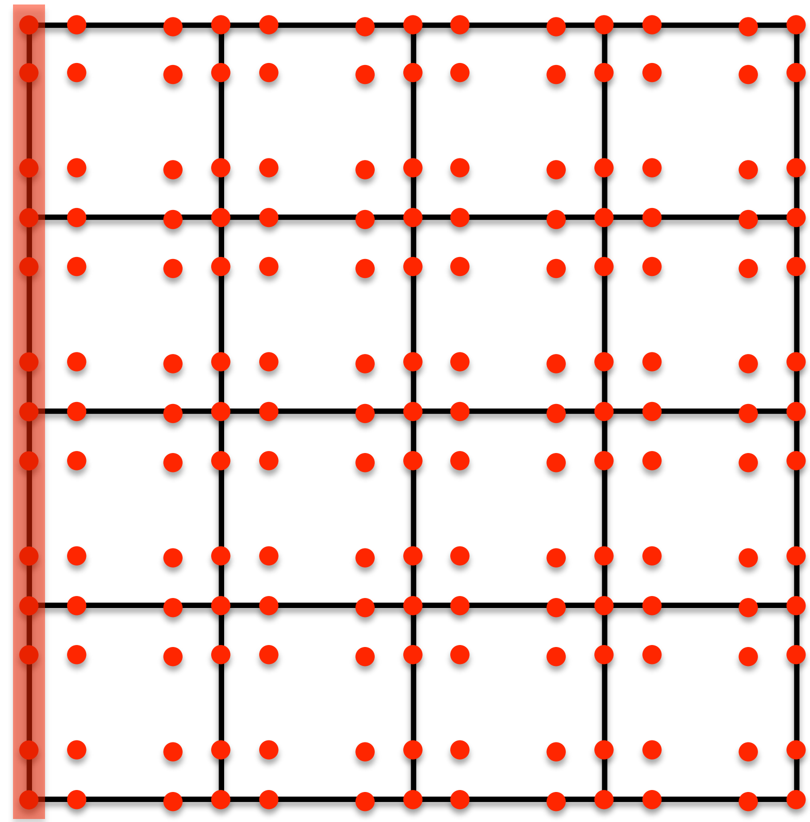
Original Pack/Exchange/Unpack

- 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



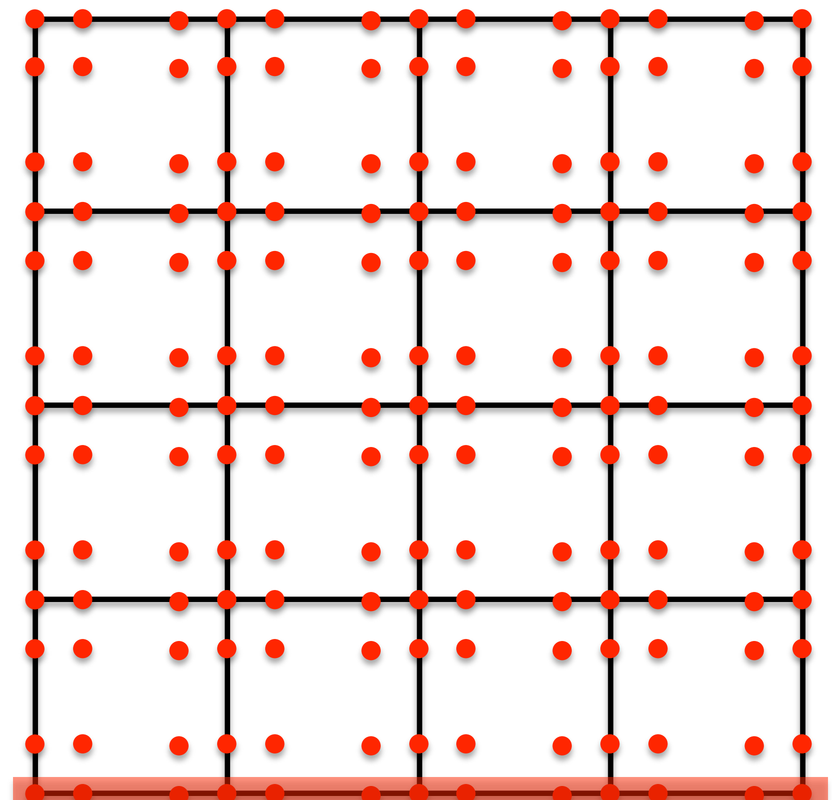
Original Pack/Exchange/Unpack

- 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



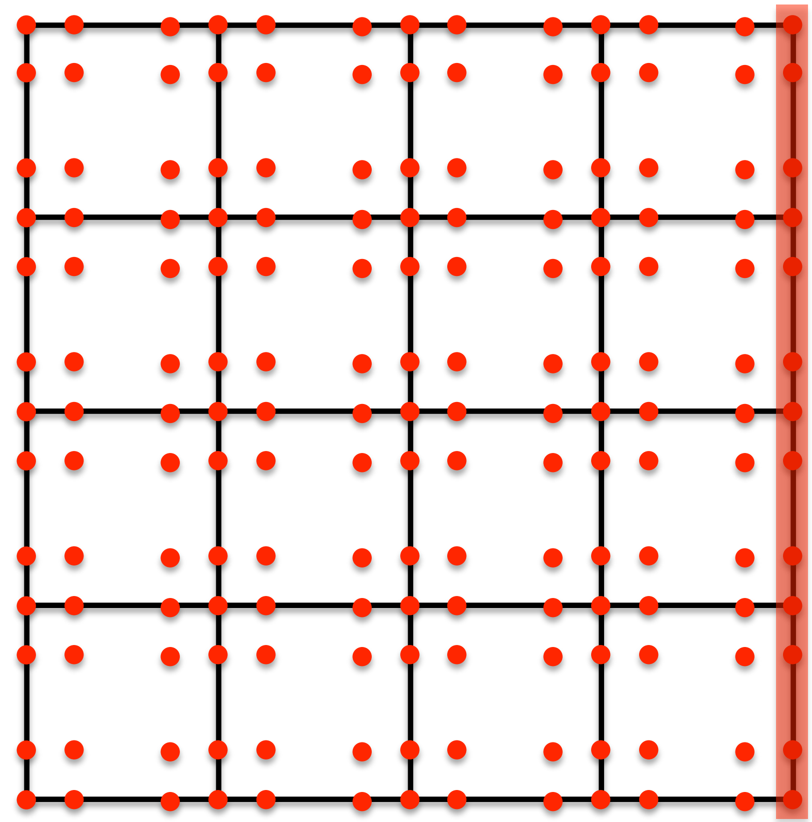
Original Pack/Exchange/Unpack

- 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



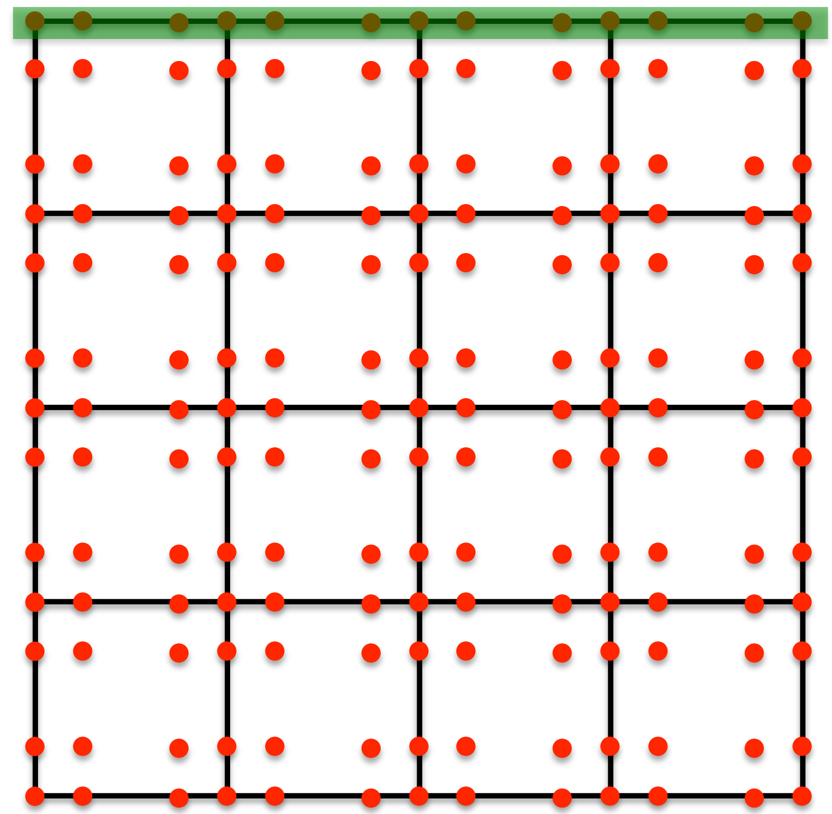
Original Pack/Exchange/Unpack

- 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



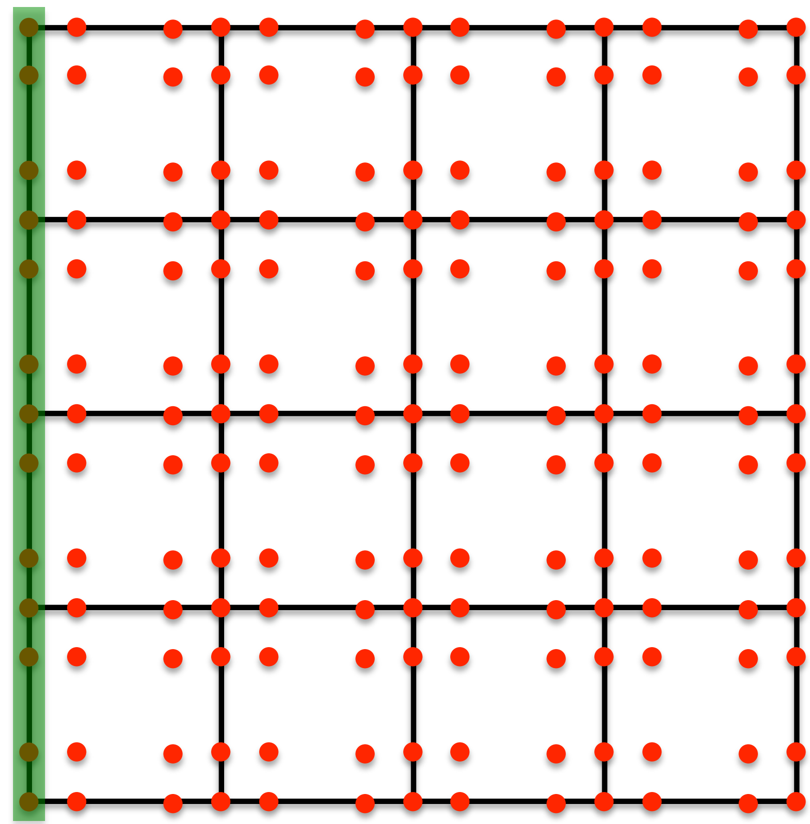
Original Pack/Exchange/Unpack

- 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)



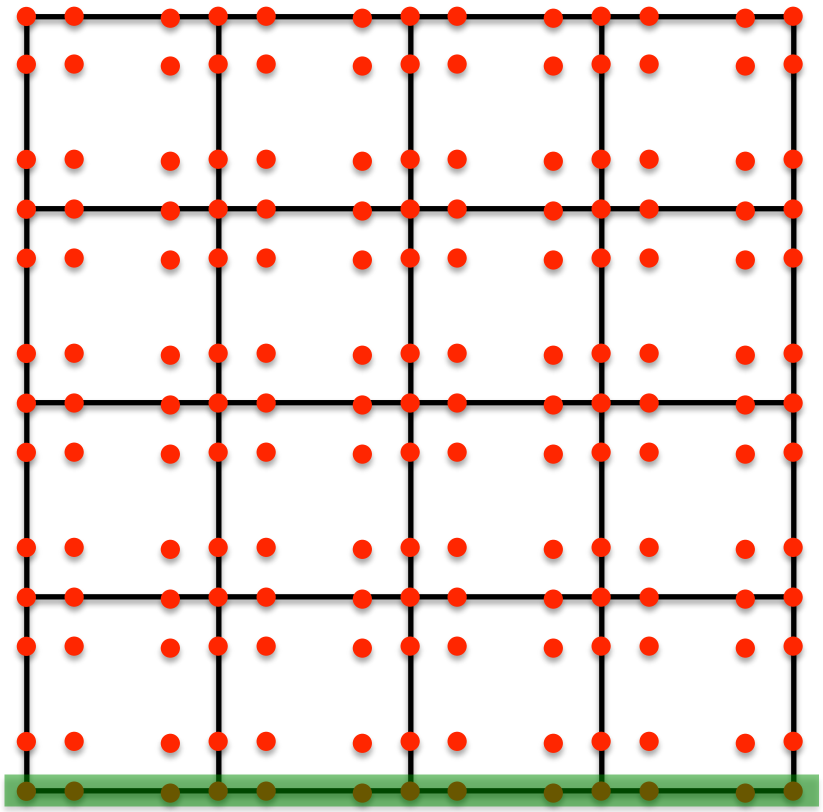
Original Pack/Exchange/Unpack

- 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)



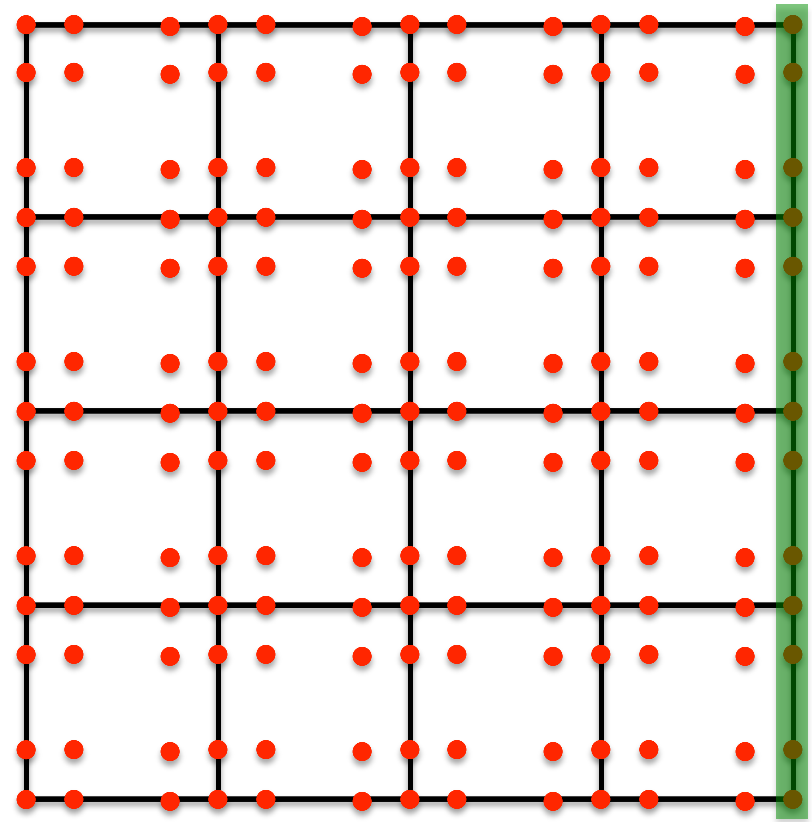
Original Pack/Exchange/Unpack

- 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)



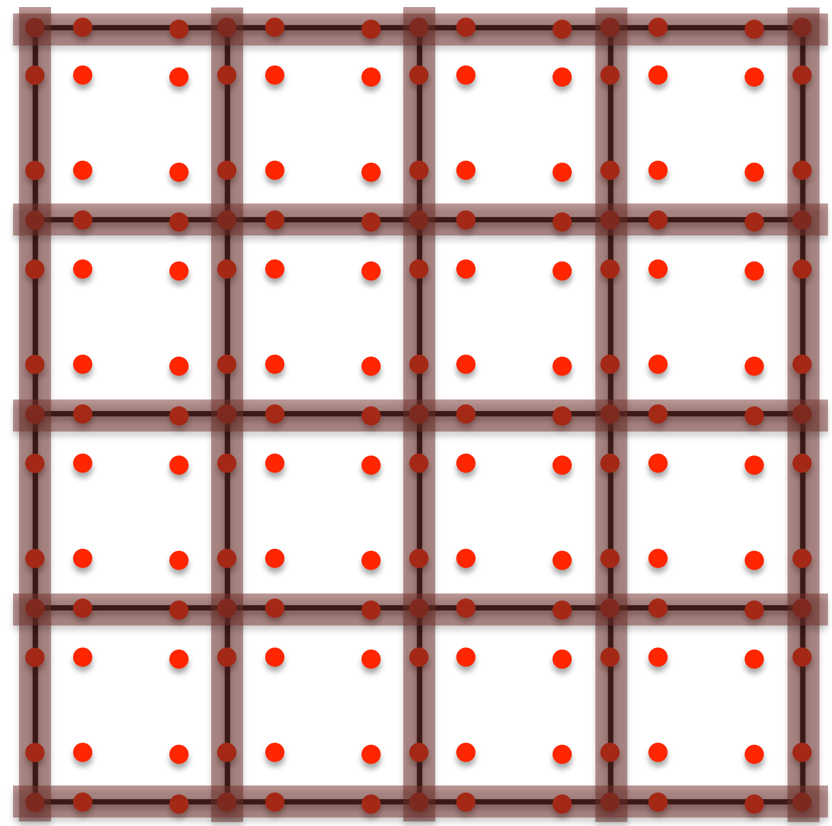
Original Pack/Exchange/Unpack

- 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

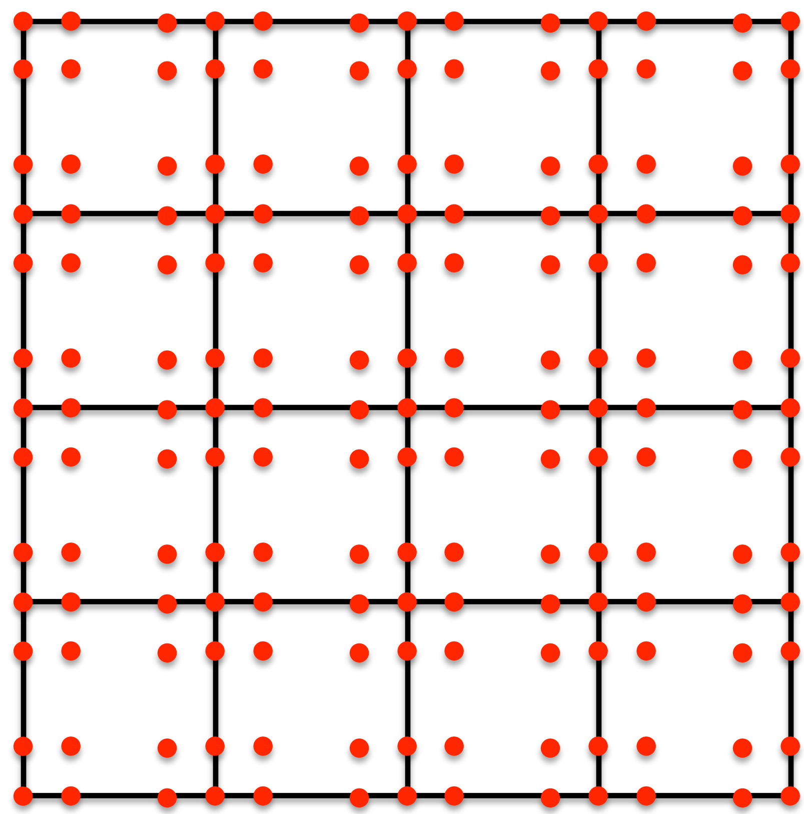


Optimizing Pack/Exchange/Unpack

- For a cycle, PCI-e D2H depends only on packing <u>that</u> cycle
  - <u>Divide</u> 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

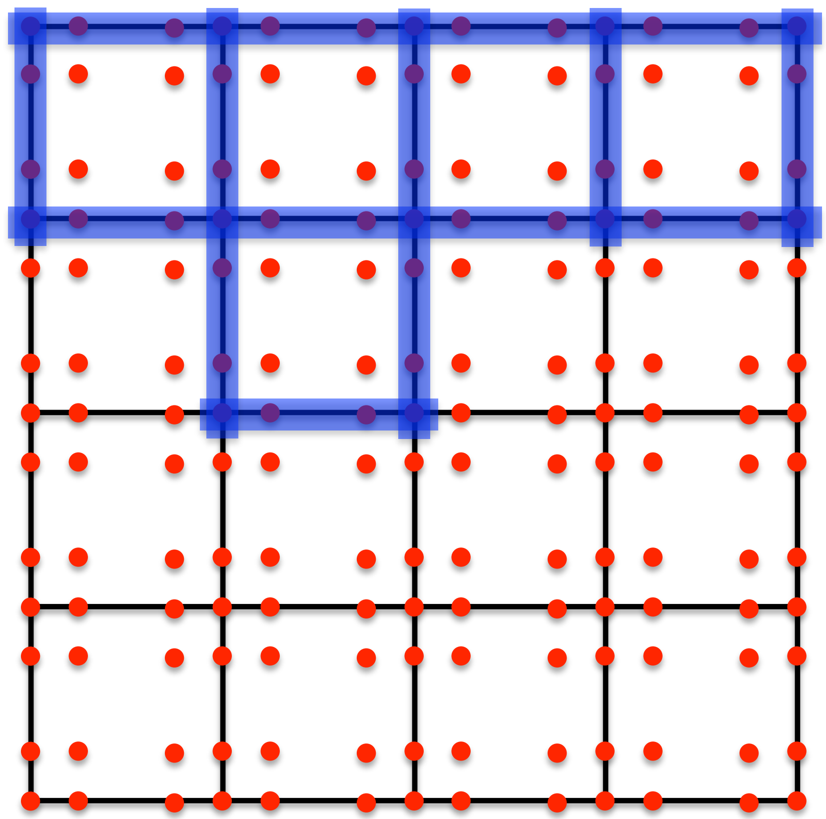
Porting Strategy: Pack/Exchange/Unpack

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



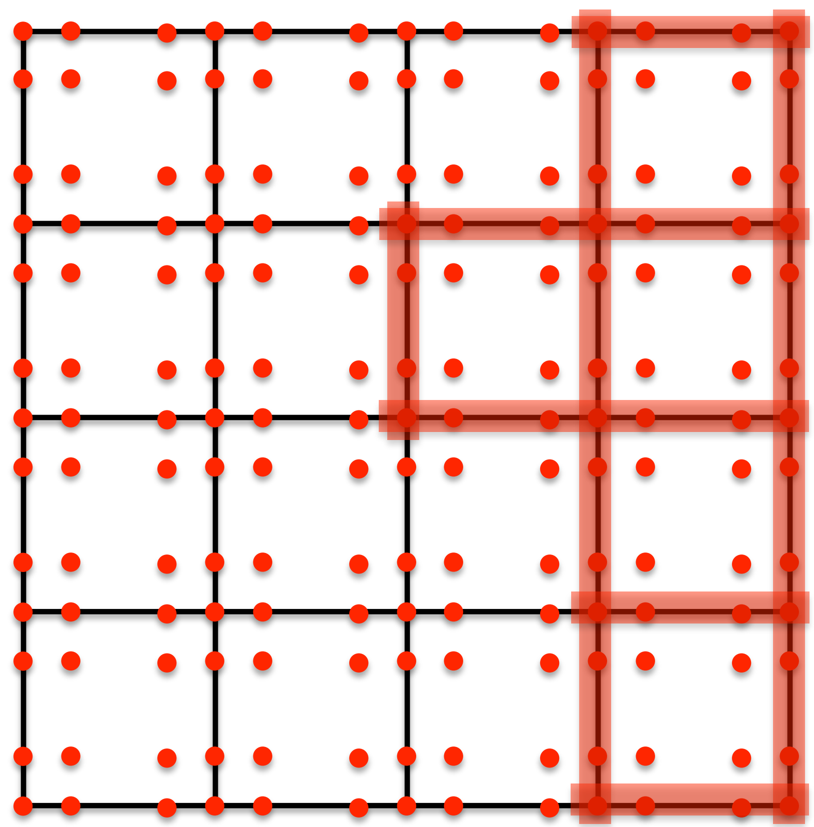
Porting Strategy: Pack/Exchange/Unpack

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



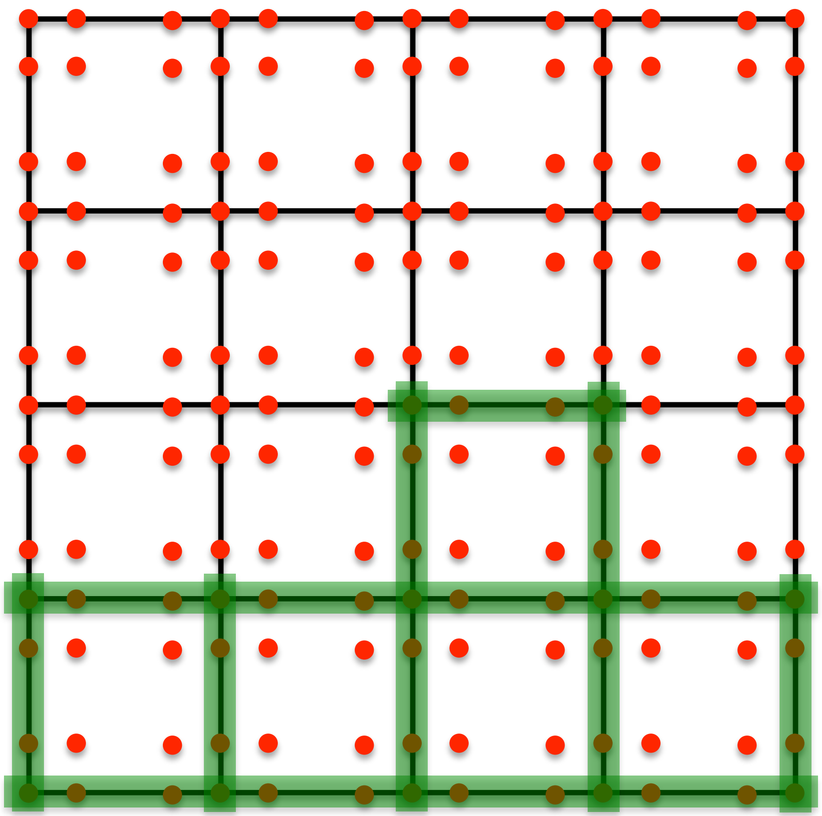
Porting Strategy: Pack/Exchange/Unpack

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



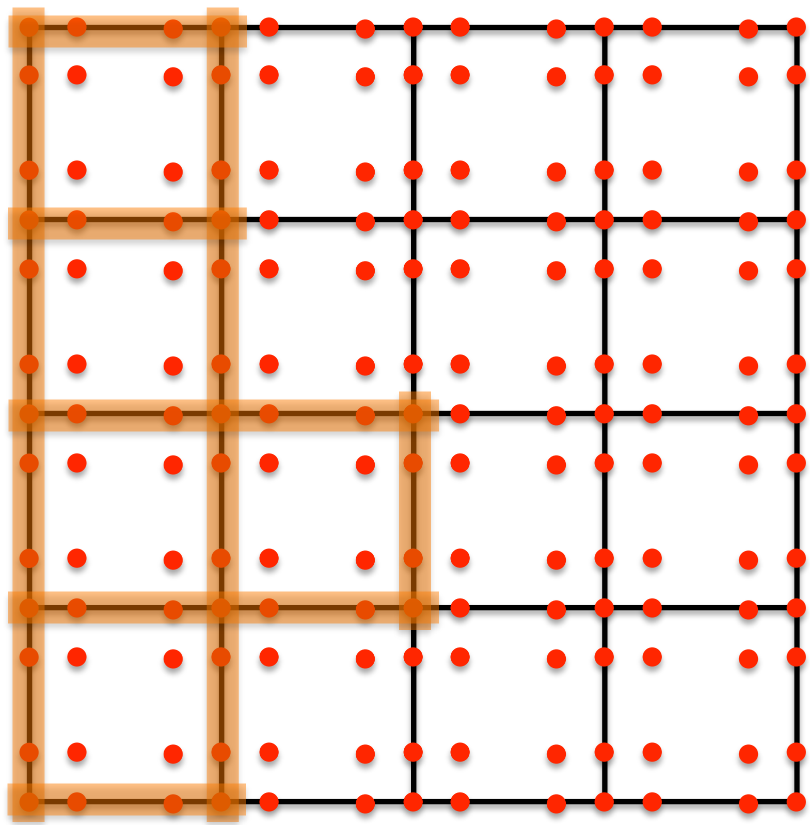
Porting Strategy: Pack/Exchange/Unpack

- 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



Porting Strategy: Pack/Exchange/Unpack

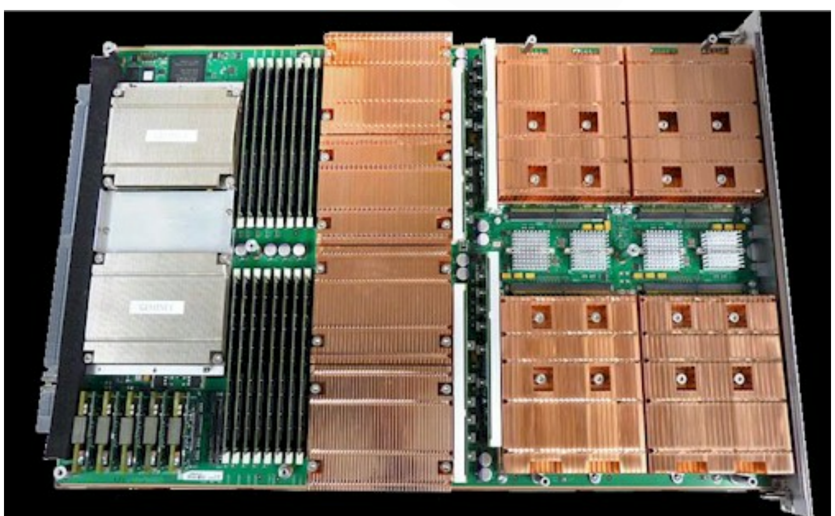
- For each cycle
  - Launch edge_pack kernel for the cycle in a unique stream
  - Call a cudaEventRecord for the stream's packing event

Porting Strategy: Pack/Exchange/Unpack

- Prepost each cycle's MPI_irecv
- While an MPI message remains pending
  - If <u>all</u> cycles finished packing (cudaEventQuery for all cycles' pack)
    - Launch edge_unpack kernel over elements not dealing with MPI
  - For each cycle
    - If cycle finished packing (cudaEventQuery for the cycle's pack)
      - Call async. PCI-e D2H copy for the cycle's MPI data
      - Call cudaEventRecord for a PCI-e D2H event
    - If cycle finished D2H PCI-e (cudaEventQuery for the cycle's D2H)
      - Call MPI_Isend for the cycle's MPI data
    - If MPI data has been received (MPI_Test for the cycle's transfer)
      - Call PCI-e H2D copy for the cycle's MPI data
- Call a device-wide barrier to ensure PCI-e H2D copies are done
- Unpack elements dealing with MPI

Resulting Concurrency

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

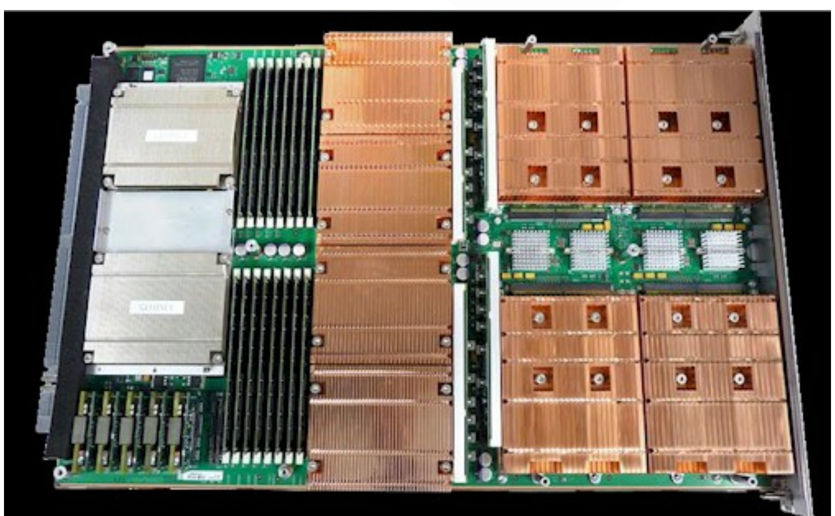


http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg

Resulting Concurrency

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

GPU Kernels



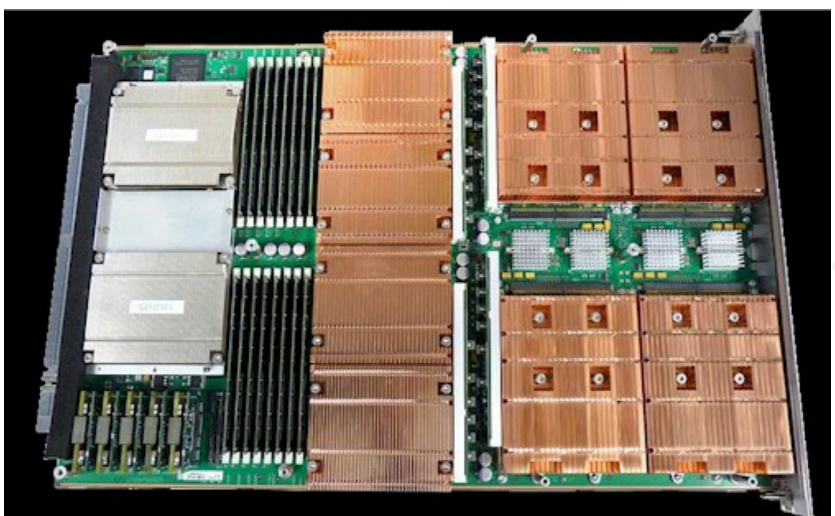
http://regmedia.co.uk/2011/05/22/cray-xk6_super-blade.jpg

Resulting Concurrency

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

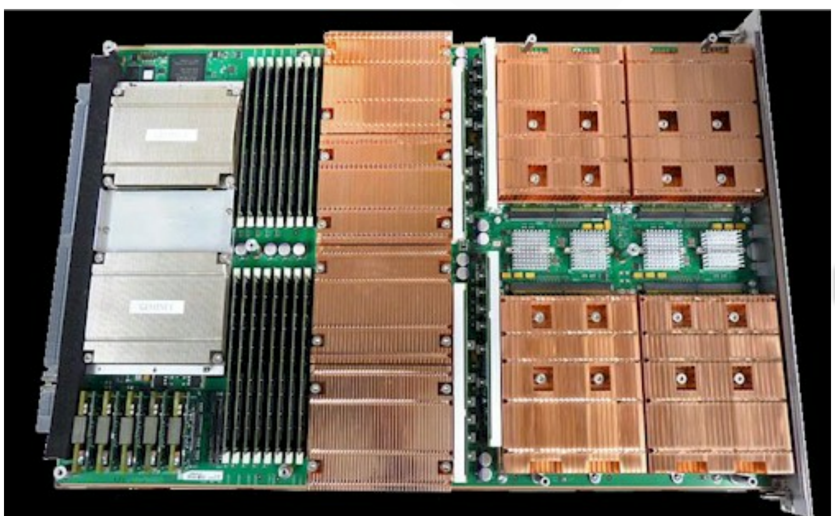
Resulting Concurrency

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

GPU Kernels

PCI-e D2H

PCI-e H2D


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

Resulting Concurrency

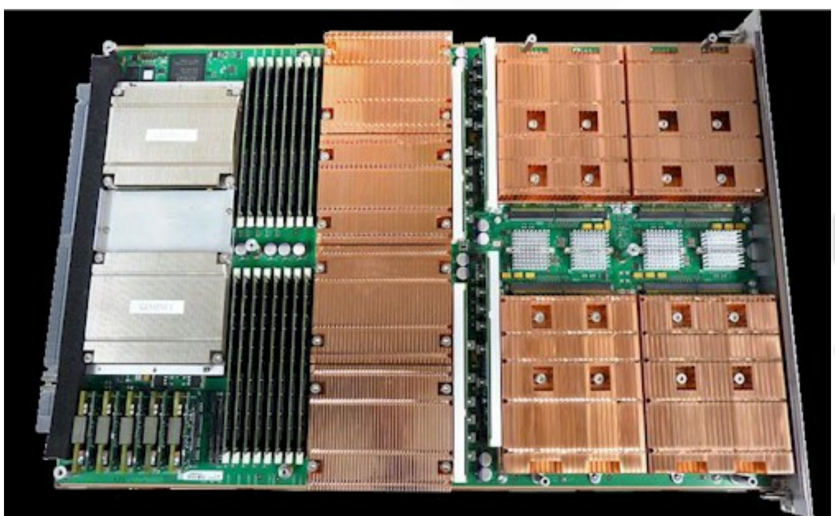
http://www.thinkdigit.com/FCKeditor/uploads/26mar10470oin342t.jpg

GPU Kernels

PCI-e D2H

PCI-e H2D

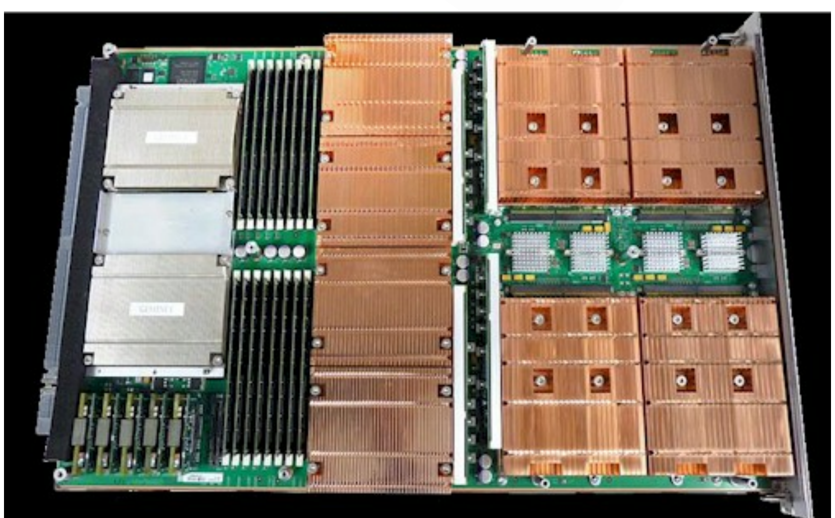
MPI



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

Resulting Concurrency

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

MPI

Host Computation

Other Important Porting Considerations

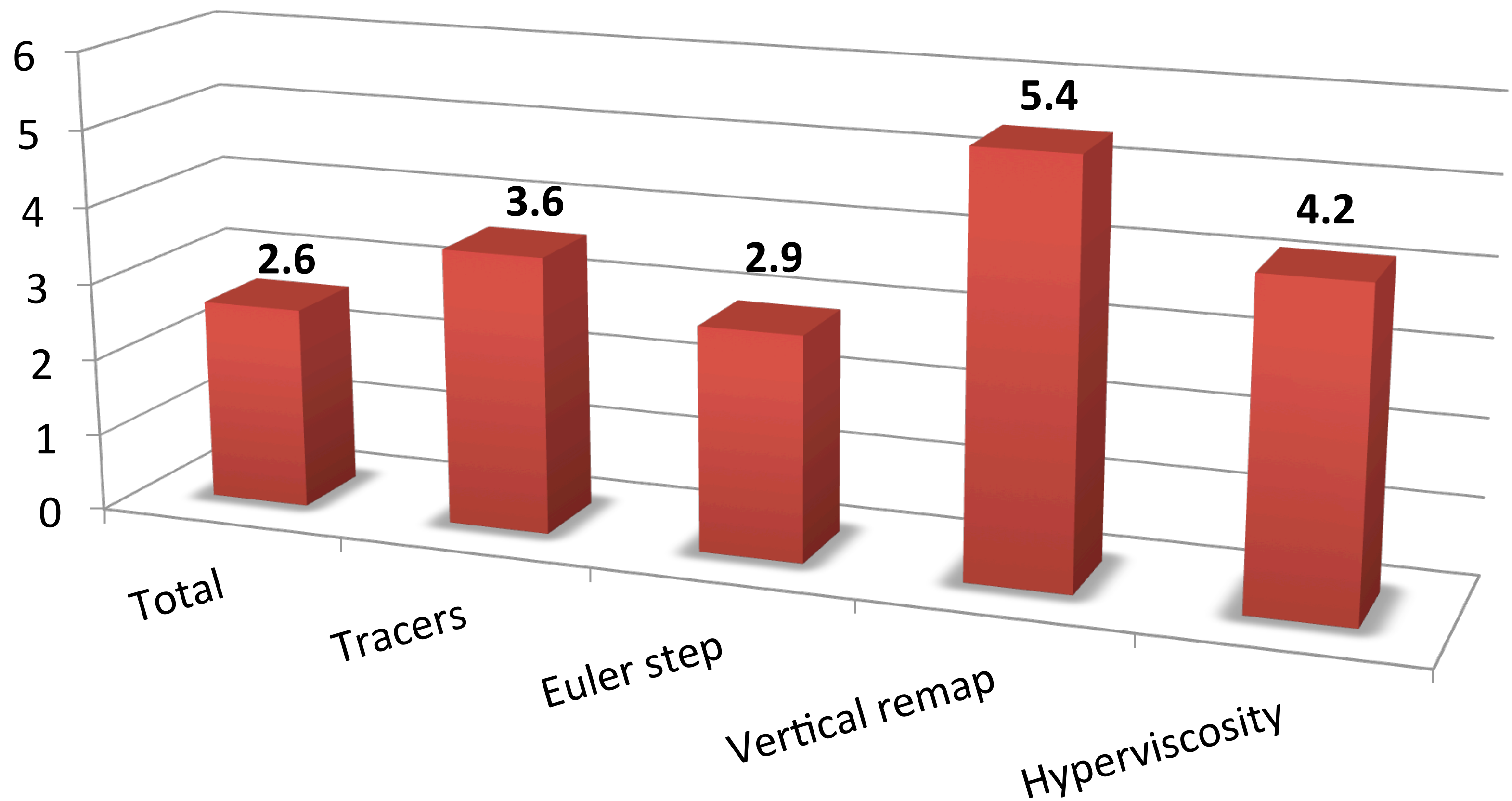
- Memory coalescing in kernels
  - Know how threads are accessing GPU DRAM, rethread if necessary
- Use of shared memory
  - Load data from DRAM to shared memory (coallesced)
  - Reuse as often as possible before re-accessing DRAM
  - Watch out for banking conflicts
- Overlapping kernels, CPU, PCI-e, & MPI
  - Perform independent CPU code during GPU kernels, PCI-e, & MPI
  - Break up & stage computations to overlap PCI-e, MPI, & GPU kernels
- PCI-e copies: consolidate if small, break up & pipeline if large
- GPU's user-managed cache made memory optimizations that are more difficult on a non-managed cache

Porting Challenges

- Data structures: derived types of derived types of derived types
  - Very difficult for directives
- Interaction with the community
  - Reproducibility: bit for bit same answer across any MPI decomp
  - Likely useful to validate GPU-based results before science
  - Double precision is currently a requirement
- Dynamical core is <u>still</u> rapidly evolving
  - About to be accepted as the default core
  - This means lots of testing and changes
- CUDA Fortran: Still evolving
  - Many layers for something to go wrong. Hard to pinpoint.
  - New versions of compiler, CUDA, GPU, driver usually mean new bugs

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



Why Was Vertical Remap So Fast?

- Originally used splines for reconstruction
  - Splines require a linear solve → vertical dependence within loops
  - Vertical index could not be threaded, only horizontal
- We replaced reconstruction with Piecewise Parabolic Method
  - Vertically independent → vertical index was threaded → 30x more threads
- Original remapping used a summation to reduce flops
  - Summations are vertically dependent and harder to thread
- We changed it to do two integrations instead
  - This double the work for remapping
  - But it also reduced data requirements and dependence
- As a result, all data in the reconstruction and remap fit into cache
  - Only accesses to DRAM were at the very beginning and end of kernel with a lot of work in between, all done in-cache
  - Thus, >5x speed-up over PPM remap on CPU

Why Was Vertical Remap So Fast?

- Originally used splines for reconstruction
  - Splines require a linear solve → vertical dependence within loops
  - Vertical index could not be threaded, only horizontal
- We repl
  - Ver s
- Origina
  - Sur
- We ch
  - This
  - But
- As a res e
  - Only accesses to DRAM were at the very beginning and end of kernel with a lot of work in between
  - Thus, >5x speed-up over PPM remap on CPU

- **If Increasing The Workload**
  - **Allows More Threading**
  - **Decreases Data Dependence**
  - **Decreases Local Data Requirements**
- **Then It's Worth Investigating**

Questions?

Usefulness Of Porting To Accelerators

- You understand your code's challenges for many threads
- You will often refactor the algorithms themselves
  - Vertical remap: splines + summation → PPM + two integrations
  - More flops, but more independence and less data movement
- You will change the way you thread
  - Higher-level hoisting of OpenMP to allow more parallelism
  - More data-independent work, more flops
  - Better staging through cache, less data in cache (less thrashing)
- Incorporating changes into CPU code almost always speeds up the CPU code
  - This changes perspective on code refactoring cost-benefit

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, these will not be sequential accesses on GPUs

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) = ...
```

- Memory accessed in the order of instructions
  - coefs(1,1,1,1,…)
  - coefs(2,1,1,1,…)
  - coefs(3,1,1,1,…)
  - coefs(1,2,1,1,…)
  - coefs(2,2,1,1,…)
  - …

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) = ...
```

- Memory accessed in the order of threads
  - coefs(1,1,1,1,…)
  - coefs(1,2,1,1,…)
  - |
  - coefs(1,N,1,1,…)
  - coefs(1,1,2,1,…)
  - coefs(1,2,2,1,…)

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) = ...
```

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(i,j,k,q,ie,1) = ...
coefs(i,j,k,q,ie,2) = ...
coefs(i,j,k,q,ie,3) = ...
```

- Memory accessed in the order of threads
  - coefs(1,1,1,…)
  - coefs(2,1,1,…)
  - |
  - coefs(N,1,1,…)
  - coefs(1,2,1,…)
  - coefs(2,2,1,…)