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

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

2013 Programming weather, climate, and earth-system models on heterogeneous multi-core platforms
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

**Dynamical Core**

1. “Dynamics”: wind, energy, & mass
2. “Tracer” Transport: (H\(_2\)O, CO\(_2\), O\(_3\), ...)
   Transport quantities not advanced by the dynamics

[Image: http://esse.engin.umich.edu/groups/admg/dcmip/jablonowski_cubed_sphere_vorticity.png]
What is CAM-SE?

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

**Dynamical Core**

1. “Dynamics”: wind, energy, & mass
2. “Tracer” Transport: \((H_2O, CO_2, O_3, \ldots)\)
   
   Transport quantities not advanced by the dynamics

**Physics Packages**

Resolve anything interesting not included in dynamical core (moist convection, radiation, chemistry, etc)
Gridding, Numerics, & Target Run

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

http://www-personal.umich.edu/~paullric/A_CubedSphere.png
Gridding, Numerics, & Target Run

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

http://www-personal.umich.edu/~paullric/A_CubedSphere.png
Gridding, Numerics, & Target Run

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

http://www-personal.umich.edu/~paullnic/A_CubedSphere.png
Gridding, Numerics, & Target Run

- 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 16K nodes with about 60% parallel efficiency
- Must simulate 1-2 thousand times faster than real time
- With 10 second CAM-SE time step, need $\leq 10$ 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
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**

**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.

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

Edges are averaged, and the average replaces both 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
  - 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
Porting Strategy: Pack/Exchange/Unpack
Porting Strategy: Pack/Exchange/Unpack

- Pack external elements that participate with MPI
Porting Strategy: Pack/Exchange/Unpack

- Pack external elements that participate with MPI
- Send Cycles over MPI and PCI-e
Porting Strategy: Pack/Exchange/Unpack

- Pack external elements that participate with MPI
- Send Cycles over MPI and PCI-e
- Pack and unpack internal elements during MPI / PCI-e
Porting Strategy: Pack/Exchange/Unpack

- Pack external elements that participate with MPI
- Send Cycles over MPI and PCI-e
- Pack and unpack internal elements during MPI / PCI-e
- MPI_irecv and PCI-e to GPU
Porting Strategy: Pack/Exchange/Unpack

- Pack external elements that participate with MPI
- Send Cycles over MPI and PCI-e
- Pack and unpack internal elements during MPI / PCI-e
- MPI_irecv and PCI-e to GPU
- Unpack external elements that participate with MPI
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 (coalesced)
  – 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
Code Changes Since Last Presentation

- Vertical remap officially changed to PPM (3x cheaper per call)
- Vertical remap subcycled over (called 5x fewer times)
- Dynamics time step increased 4x via 5-stage RK time stepping
- Surface pressure hyperdiffusion added
- Pack-Exchange-Unpack now done differently
Speedup: Kepler GPU vs 1 Interlagos / Node

- Benchmarks performed on XK7 using end-to-end wall timers
- All PCI-e and MPI communication included

- Best CPU case: 8 processes, 2 threads per process
- Best GPU case: 4 processes, 4 threads per process

- 3.1x speedup for tracer transport routines
- 2.3x speedup for the dynamical core (“HOMME”)
- 2.1x speedup for all of CAM-SE
Multicore Problems We All Have To Deal With

- Cache is expensive, it will relatively decrease
- Network is expensive, it will relatively slow down

1. If you have data, you better keep it and do something with it
2. Flexibly expose many data-independent threads

- Domain science (more useful answer + faster runtime)
  - Algorithms (communication avoidance, time step, data reuse)
  - Smart coding (blocking, caching, vectorizing, overlapping, etc)
  - Languages (ridding overconstrained MPI / OpenMP overheads)
  - Software Engineering (one source code if at all possible)
Questions?