Accelerating the Implicit Integration of Stiff Chemical Systems with Emerging Multi-core Technologies

John C. Linford†
John Michalakes‡
Manish Vachharajani§
Adrian Sandu†

IMAGe TOY 2009 Workshop 2
8/19/09

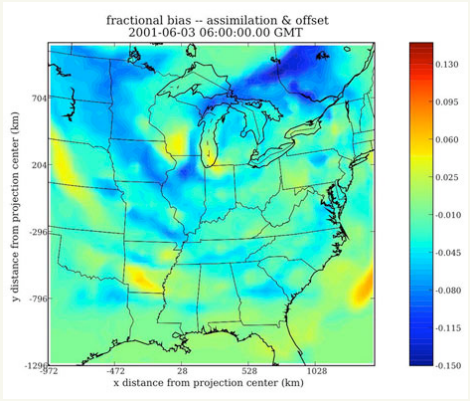
† Virginia Polytechnic Institute and State University, Department of Computer Science, Blacksburg VA
‡ National Center for Atmospheric Research, Boulder CO
§ University of Colorado at Boulder, Boulder CO

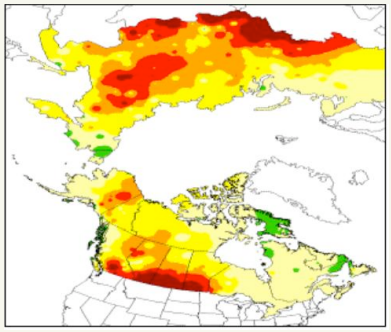
Outline

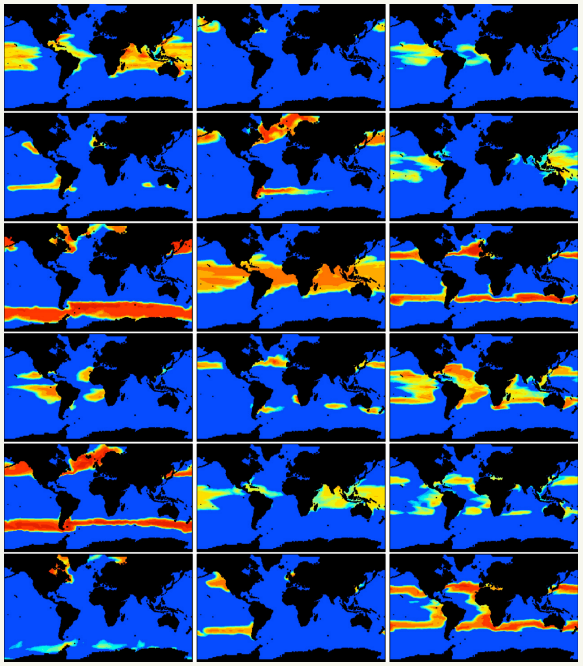
- Introduction and motivation
- The Kinetics Pre-Processor (KPP)
  - Rosenbrock solver
- Emerging multi-core architectures
  - Cell Broadband Engine Architecture (CBEA)
  - NVIDIA GTX 200 Architecture
  - Intel Quad-core Xeon 5400 series
- Case Study: RADM2 chemical mechanism from WRF/Chem
- Future Work
  - Multi-core KPP
  - Accelerated function offloading

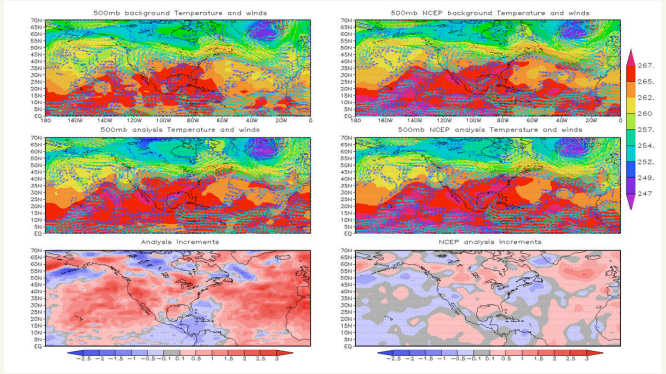
Chemical Kinetics Models

- Air Quality
  - WRF/Chem
  - CMAQ
  - STEM
- Wildfire
  - WRF/Fire
- Climate Change
  - GEOS/Chem

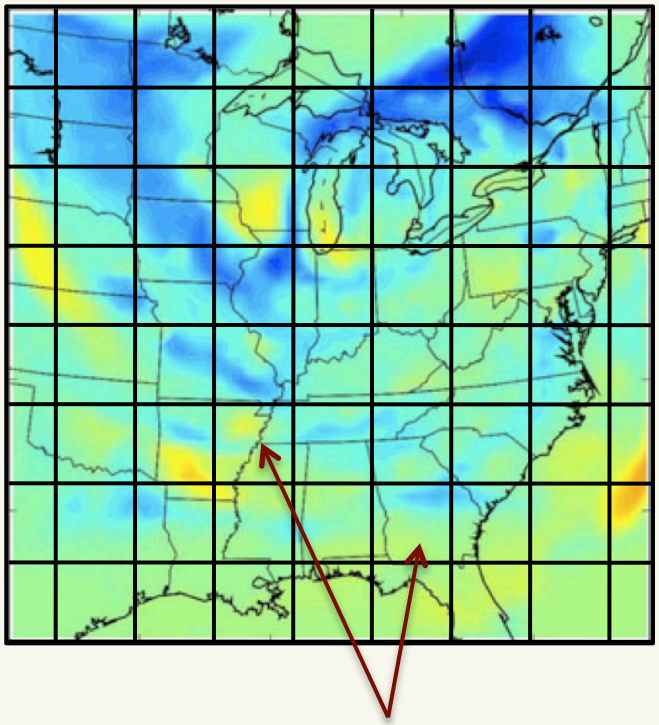




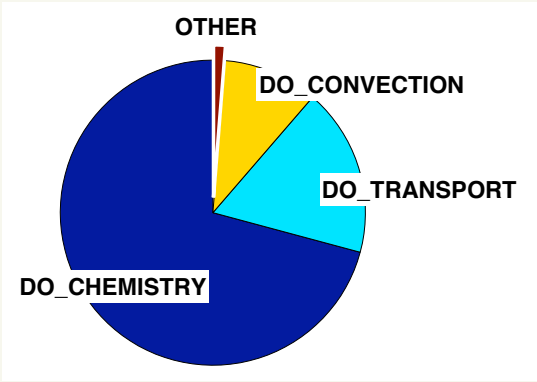




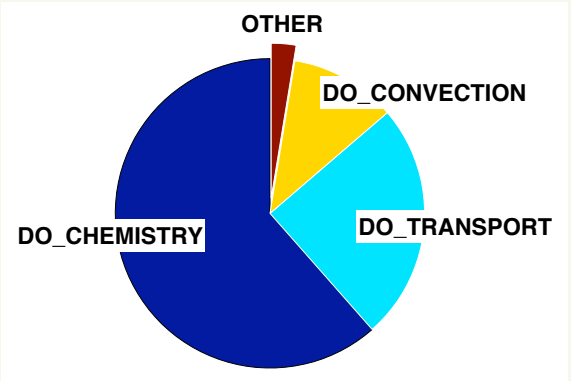
Chemical kinetics dominate computational time



Embarrassingly parallel on a regular fixed grid

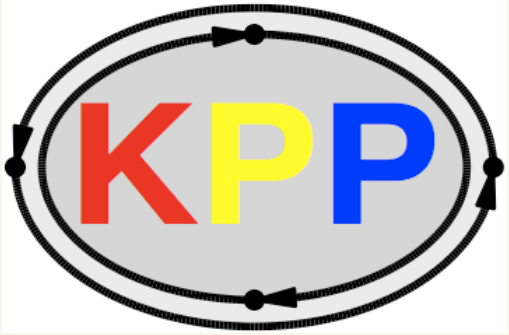


GEOS/Chem Computational Time: One Xeon core



GEOS/Chem Computational Time: Eight Xeon cores

The Kinetics Pre-Processor (KPP)



Virginia Tech

MAX-PLANCK-GESELLSCHAFT

- Generates C, Fortran or MATLAB code to compute:
  - the time-evolution of chemical species,
  - the Jacobian,
  - the Hessian, etc.
- Includes a library of widely-used solvers
  - Rosenbrock
  - Runge-Kutta, etc.
- Used by several models
  - WRF/Chem
  - GEOS/Chem
  - STEM

The KPP-generated n-stage Rosenbrock solver for chemical kinetics

- Sparse implementation
- Trades precision for speed when reasonable
- Typically outperforms backwards differentiation formulas (SMVGEAR)
- The Jacobian is generally inseparable
  - the solver itself cannot be parallelized,
  - but certain operations are highly parallel

Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)

Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$

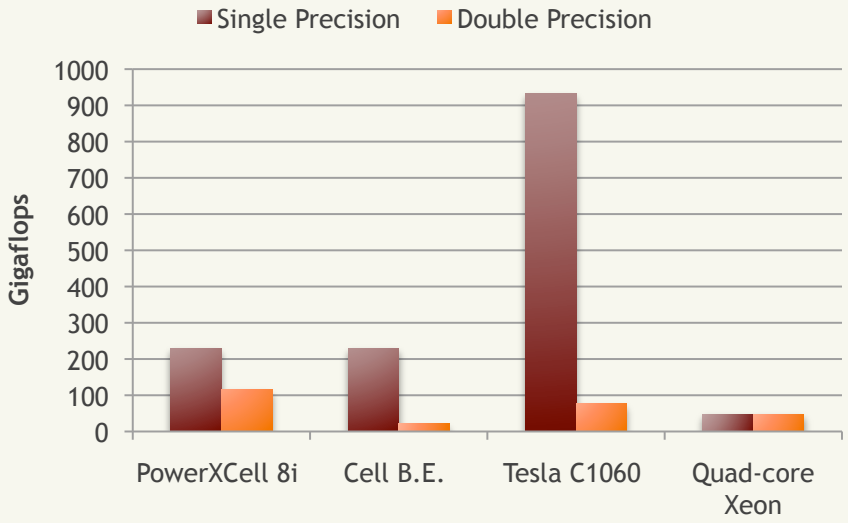
While $t \le t_{end}$

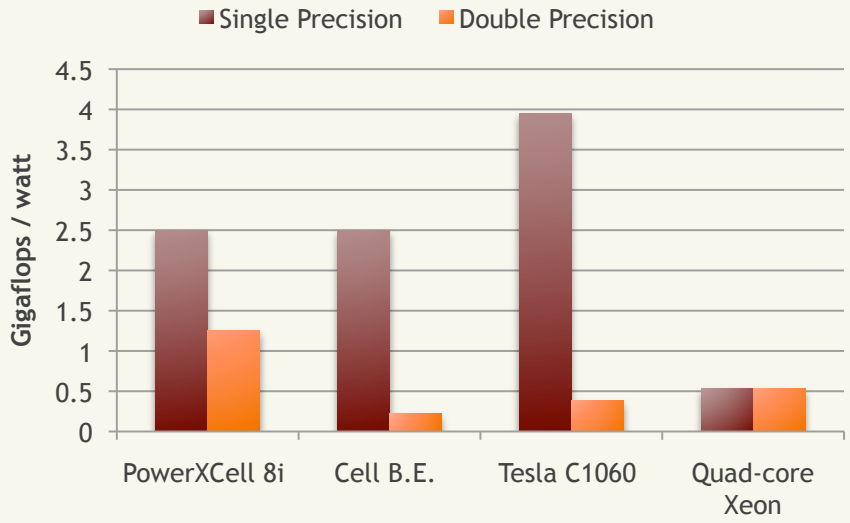
- $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
- $Jac_0 \leftarrow J(t,y)$
- $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
- For $s \leftarrow 1,2,\ldots,n$
  - Compute $Stage_s$ from Fcn and $Stage_{1\ldots(s-1)}$
  - Solve for $Stage_s$ implicitly using G
  - Update k(t,y) with meteorology (ρ, t, q, p)
  - Update Fcn from $Stage_{1\ldots s}$
- Compute Y_{new} from $Stage_{1\ldots s}$
- Compute error term E
- If $E \ge \delta$ then discard iteration, reduce h, restart
- Otherwise, $t \leftarrow t + h$ and proceed to next step

Finish : Result in Y_{new}

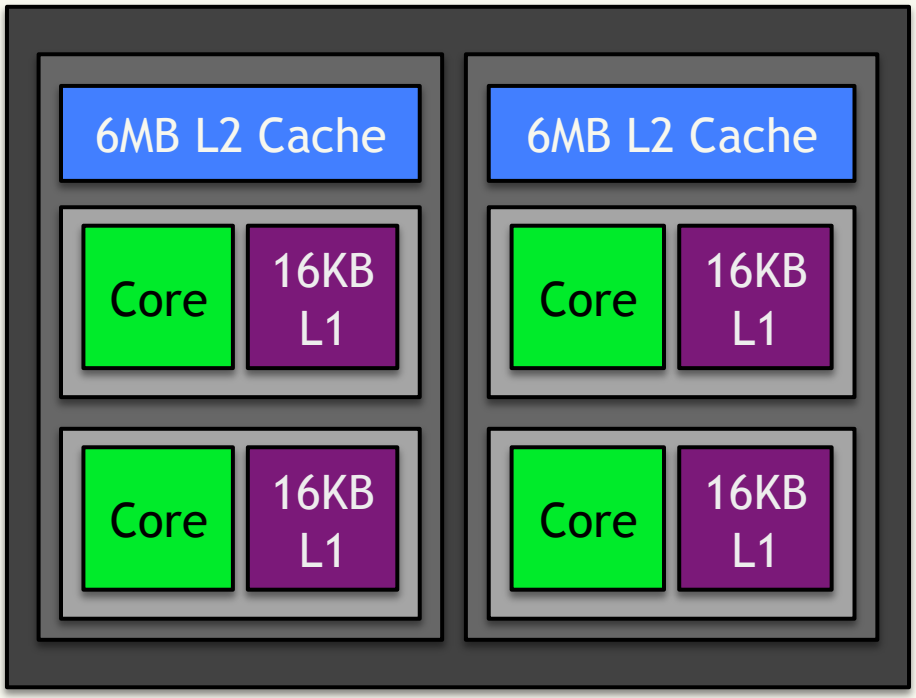
Peak performance of multi-core architectures

Gigaflops	PowerXCell 8i	Cell B.E.	Tesla C1060	Xeon 5400 Series
Single (per watt)	230.4 (2.5)	230.4 (2.5)	933.0 (3.95)	48.0 (0.53)
Double (per watt)	115.2 (1.25)	21.03 (0.22)	76.0 (0.38)	48.0 (0.53)
Max for 200 watts	460.8	460.8	933.0	96.0



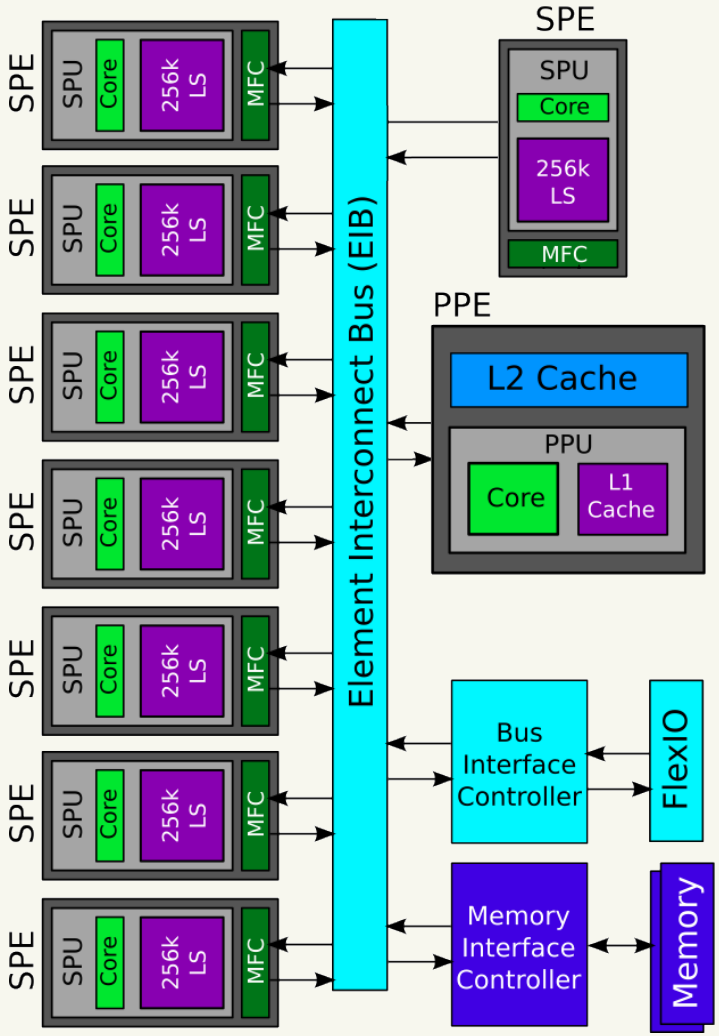


Intel Quad-core Xeon 5400 series: a general-purpose, widely-available multi-core architecture



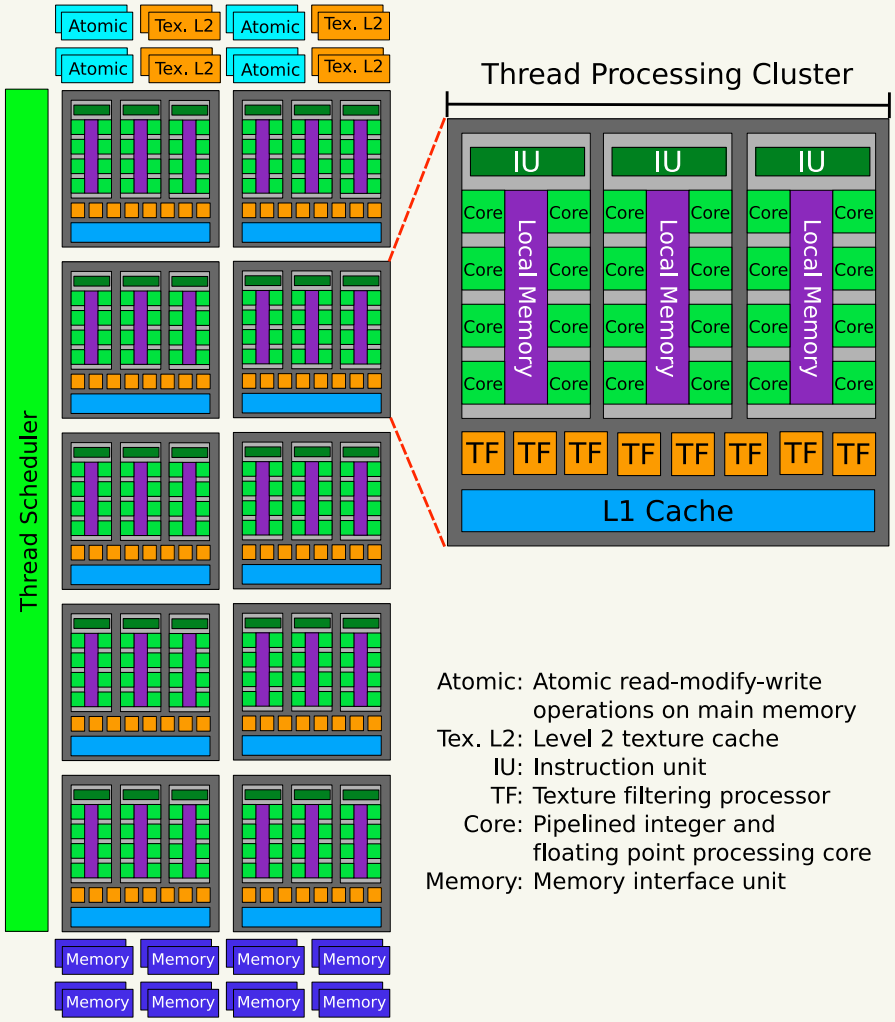
- 5400 series “Harpertown”
  - General-purpose x86 cores
  - Intel 64 architecture
  - SSE4 Extensions
  - Program with OpenMP, MPI, pthreads, etc.
  - 48 gigaflops peak
    - 0.53 per watt

Cell Broadband Engine CPU Architecture (CBEA): a heterogeneous high-performance architecture



- PowerXCell 8i
  - 1 dual-thread PowerPC
  - 8 accelerators (SPEs)
    - 128-bit vector processor
    - 256KB software-controlled local storage
    - DMA engine (MFC)
  - Element Interconnect Bus
    - 204.8GB/second
  - 230.5 SP gigaflops
    - 2.5 per watt
  - 115.25 DP gigaflops
    - 1.25 per watt

NVIDIA GTX 200 GPGPU: a versatile floating-point accelerator architecture



Atomic: Atomic read-modify-write operations on main memory
Tex. L2: Level 2 texture cache
IU: Instruction unit
TF: Texture filtering processor
Core: Pipelined integer and floating point processing core
Memory: Memory interface unit

- NVIDIA Tesla C1060
  - 4GB GDDR3 RAM
  - 30 multiprocessors, each with:
    - 8 SP cores
    - 1 shared DP unit
    - 16kb shared memory
    - 16kb register file
  - 933 SP gigaflops
    - 3.95 per watt
  - 76 DP gigaflops
    - 0.76 per watt
- Programmed with CUDA
  - C/C++ extension
  - NVIDIA proprietary
  - Architecture abstracted

Let's compare these architectures…

	Quad-core Xeon	CBEA	GTX 200
# threads	4	10	Thousands
Heterogeneity	None	On-chip	Host/device
SIMD cardinality	4	4	32 (warp size)
Per-core memory	3-6MB	256KB	16KB
# address spaces	1	2	2
# memory types	1	2	6
Mem. hierarcy ctl.	Fully implicit	Fully explicit	Mixed
Data alignment	Optional	Required	Encouraged

- Leverage SIMD ISAs with different cardinalities
- Track data movement across multiple address spaces
- Use each type of memory to best effect
- Avoid inefficient access patterns

Case study: RADM2 from WRF/Chem

- 59 species, 156 reactions
- ~15KB of data per grid cell
- 3-stage KPP-generated Rosenbrock integrator
- See http://ruc.fsl.noaa.gov/wrf/WG11/radm2.htm

RADM2 on Intel Quad-core Xeon

```
for j=1,2,…
 for k=1,2,…
  for i=1,2,…
   INTEGRATE()
   {
```

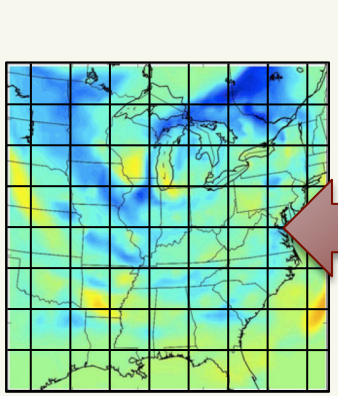
Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)
Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$
While $t \le t_{end}$
  $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
  $Jac_0 \leftarrow J(t,y)$
  $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
  For $s \leftarrow 1,2,\ldots,n$
    Compute $Stage_s$ from Fcn and $Stage_{1\ldots(s-1)}$
    Solve for $Stage_s$ implicitly using G
    Update $k(t,y)$ with meteorology (ρ, t, q, p)
    Update Fcn from $Stage_{1\ldots s}$
  Compute Y_{new} from $Stage_{1\ldots s}$
  Compute error term E
  If $E \ge \delta$ then discard iteration, reduce h, restart
  Otherwise, $t \leftarrow t + h$ and proceed to next step
Finish : Result in Y_{new}

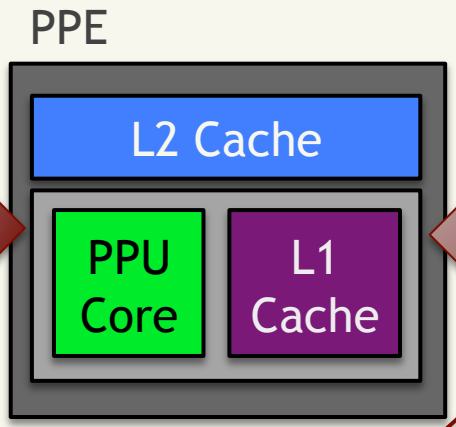
```
   }
```

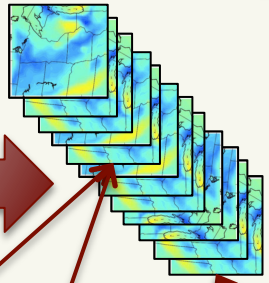
```
#pragma omp parallel for \
 default(none) \
 private(j,k,i,…) \
 shared(chem,moist,phy,…)
for j=1,2,…
 for k=1,2,…
  for i=1,2,…
   INTEGRATE()
   {
    ...
   }
```

3.5x faster than serial Fortran

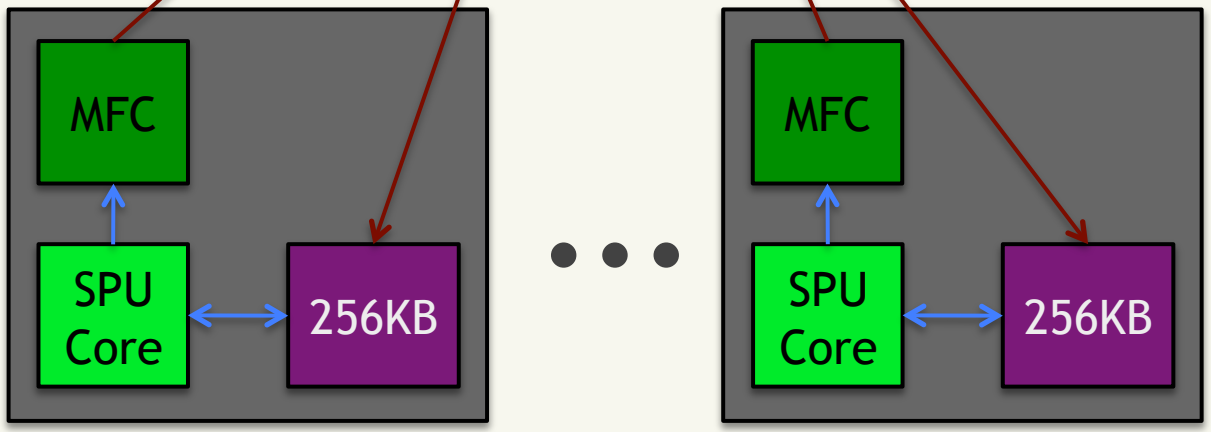
RADM2 on the CBEA: scalar master-worker







Large buffer of grid cells as user-defined type: `cell_t`



10x faster than serial Fortran

Vectorized n-stage Rosenbrock solver

Byte 0 (MSB)	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15 (LSB)
doubleword 0								*doubleword 1*							
word 0				*word 1*				*word 2*				*word 3*			
halfword 0		*halfword 1*		*halfword 2*		*halfword 3*		*halfword 4*		*halfword 5*		*halfword 6*		*halfword 7*	
char 0	*char 1*	*char 2*	*char 3*	*char 4*	*char 5*	*char 6*	*char 7*	*char 8*	*char 9*	*char 10*	*char 11*	*char 12*	*char 13*	*char 14*	*char 15*

Vectorized n-stage Rosenbrock solver

Vector element 1

Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)

Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$

While $t \le t_{end}$

- $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
- $Jac_0 \leftarrow J(t,y)$
- $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
- For $s \leftarrow 1,2,\ldots,n$
  - Compute $Stage_s$ from Fcn and $Stage_{1\ldots(s-1)}$
  - Solve for $Stage_s$ implicitly using G
  - Update k(t,y) with meteorology (ρ, t, q, p)
  - Update Fcn from $Stage_{1\ldots s}$
- Compute Y_{new} from $Stage_{1\ldots s}$
- Compute error term E
- If $E \ge \delta$ then discard iteration, reduce h, restart
- Otherwise, $t \leftarrow t + h$ and proceed to next step

Finish: Result in Y_{new}

Vector element 2

Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)

Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$

While $t \le t_{end}$

- $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
- $Jac_0 \leftarrow J(t,y)$
- $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
- For $s \leftarrow 1,2,\ldots,n$
  - Compute $Stage_s$ from Fcn and $Stage_{1\ldots(s-1)}$
  - Solve for $Stage_s$ implicitly using G
  - Update k(t,y) with meteorology (ρ, t, q, p)
  - Update Fcn from $Stage_{1\ldots s}$
- Compute Y_{new} from $Stage_{1\ldots s}$
- Compute error term E
- If $E \ge \delta$ then discard iteration, reduce h, restart
- Otherwise, $t \leftarrow t + h$ and proceed to next step

Finish: Result in Y_{new}

Vectorized n-stage Rosenbrock solver

Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)

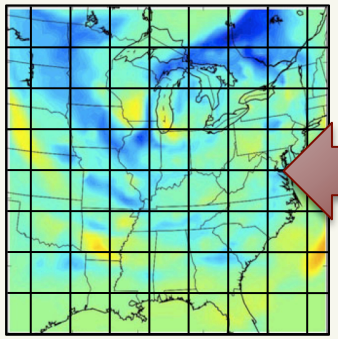
Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$

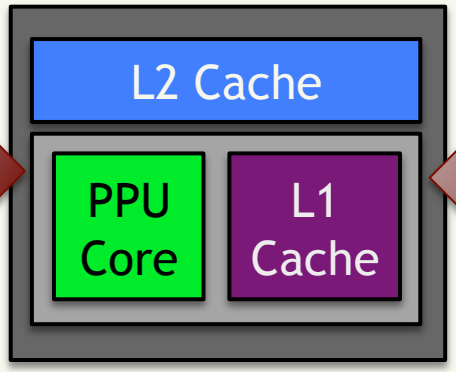
While $t \leq t_{end}$

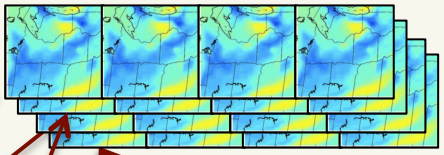
- $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
- $Jac_0 \leftarrow J(t,y)$
- $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
- For $s \leftarrow 1,2,\ldots,n$
  - Compute $Stage_s$ from Fcn and $Stage_{1\ldots(s\text{-}1)}$
  - Solve for $Stage_s$ implicitly using G
  - Update k(t,y) with meteorology (ρ, t, q, p)
  - Update Fcn from $Stage_{1\ldots s}$
- Compute Y_{new} from $Stage_{1\ldots s}$
- Compute error term $E = \max(E_1, E_2, \ldots, E_{vn})$
- If $E \geq \delta$ then discard iteration, reduce h, restart
- Otherwise, $t \leftarrow t + h$ and proceed to next step

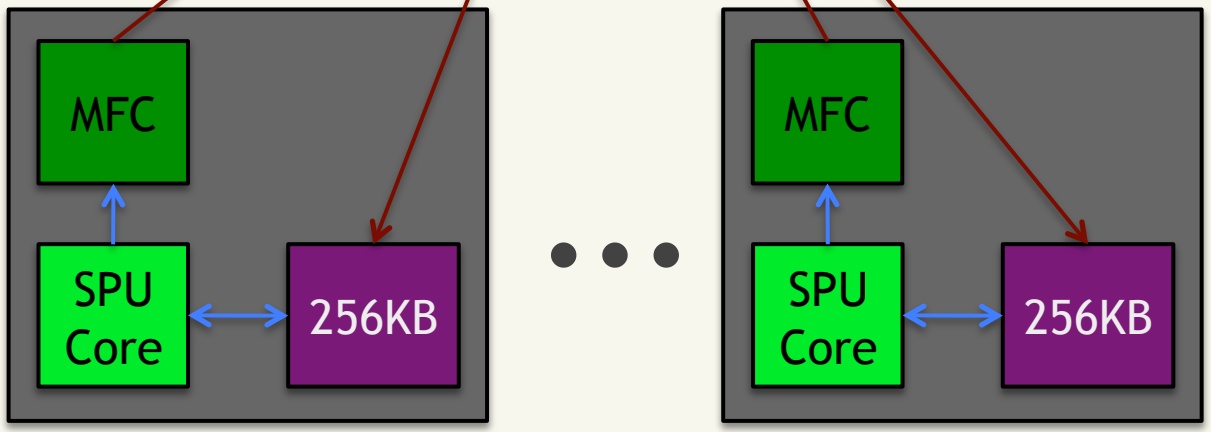
Finish: Result in Y_{new}

RADM2 on the CBEA: vector master-worker









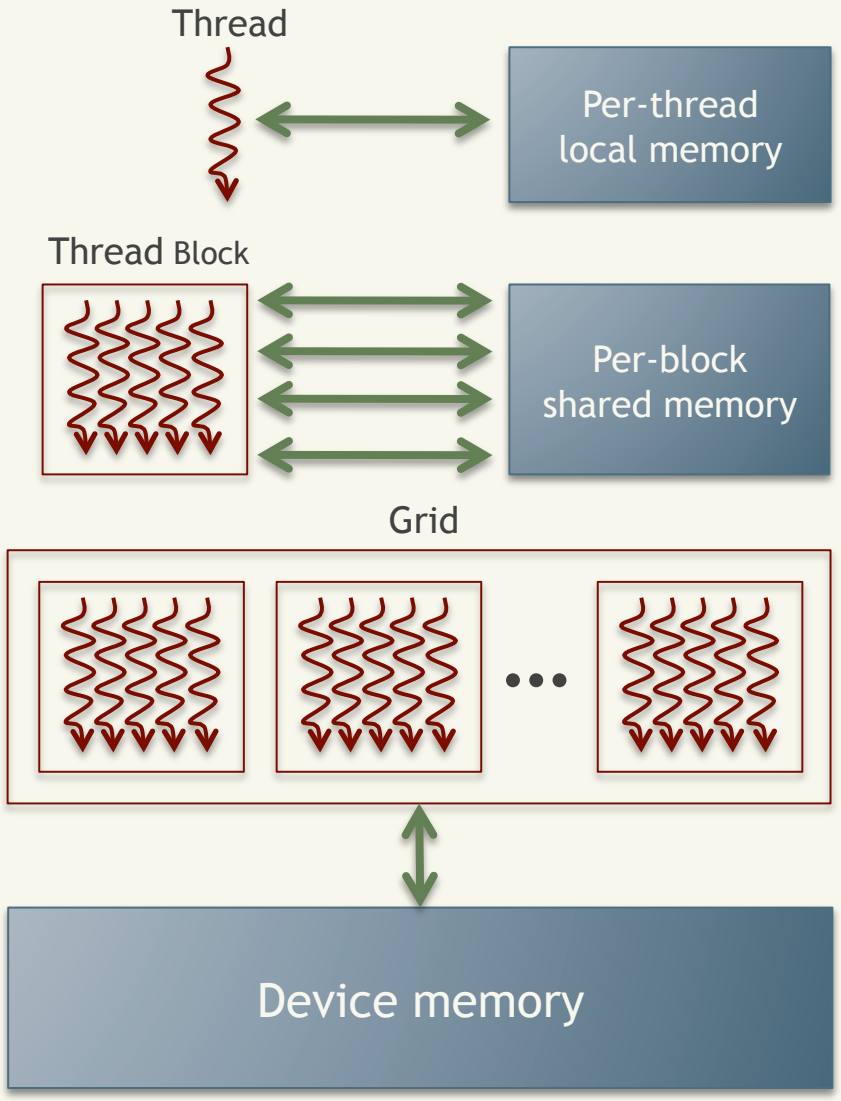
40x faster than serial Fortran

RADM2 on the CBEA: Future work

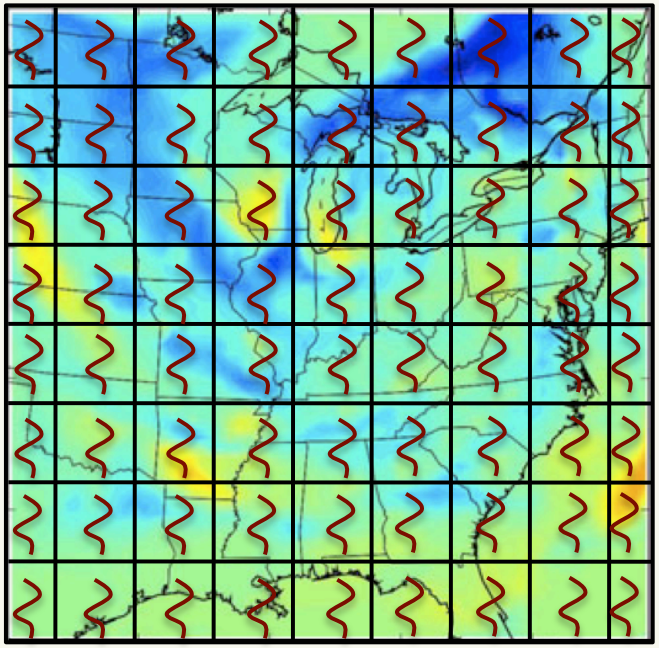
- Only one PPE thread. Three unused hardware threads!
- No dynamic branch prediction.
- Store constant data in only one local store.
- Use intrinsic operations whenever possible.

RADM2 on CUDA: one-thread-per-cell

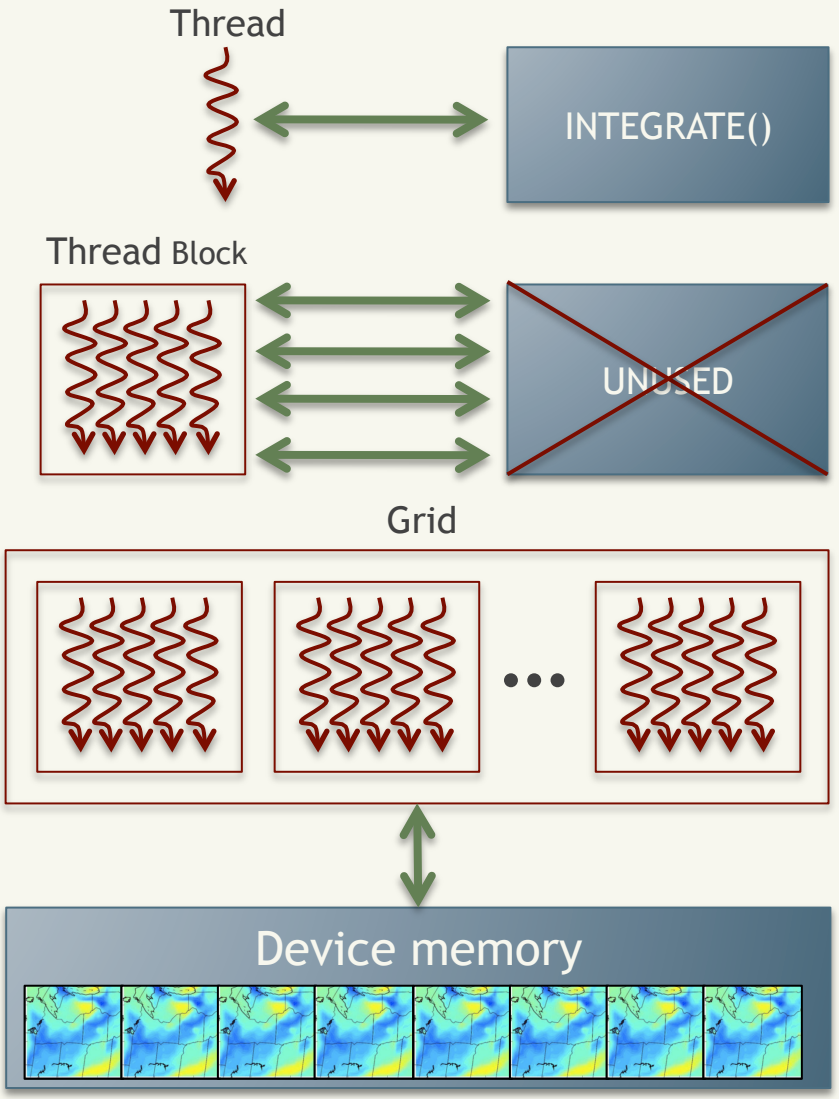
- The GPU *device* is a co-processor to the CPU *host*.
- The device runs many lightweight *threads*.
- Threads execute *kernels*.
- Threads are grouped into *blocks*.
- Blocks are aggregated into a *grid*.
- Blocks are processed as SIMD groups of threads called *warps*.



RADM2 on CUDA: one-kernel implementation



2.2x faster than serial Fortran



RADM2 on CUDA: multi-kernel implementation

Host

Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)

Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$

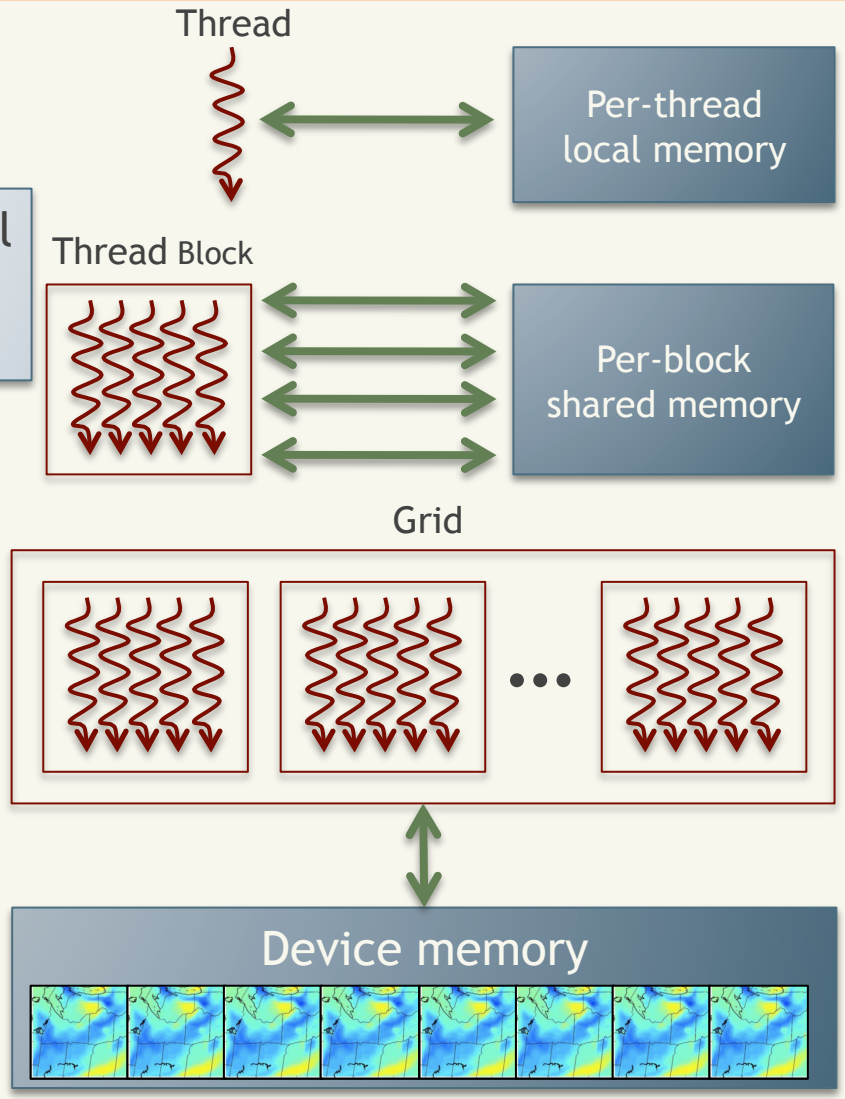
While $t \leq t_{end}$

- $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
- $Jac_0 \leftarrow J(t,y)$
- $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
- For $s \leftarrow 1,2,\ldots,n$
  - Compute $Stage_s$ from Fcn and $Stage_{1\ldots(s-1)}$
  - Solve for $Stage_s$ implicitly using G
  - Update k(t,y) with meteorology (ρ, t, q, p)
  - Update Fcn from $Stage_{1\ldots s}$
- Compute Y_{new} from $Stage_{1\ldots s}$
- Compute error term $E = \max(E_1, E_2, \ldots, E_{vn})$
- If $E \geq \delta$ then discard iteration, reduce h, restart
- Otherwise, $t \leftarrow t + h$ and proceed to next step

Finish: Result in Y_{new}

An optimized kernel is defined for each operation.

Device



RADM2 on CUDA: multi-kernel implementation

Host

Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)

Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$

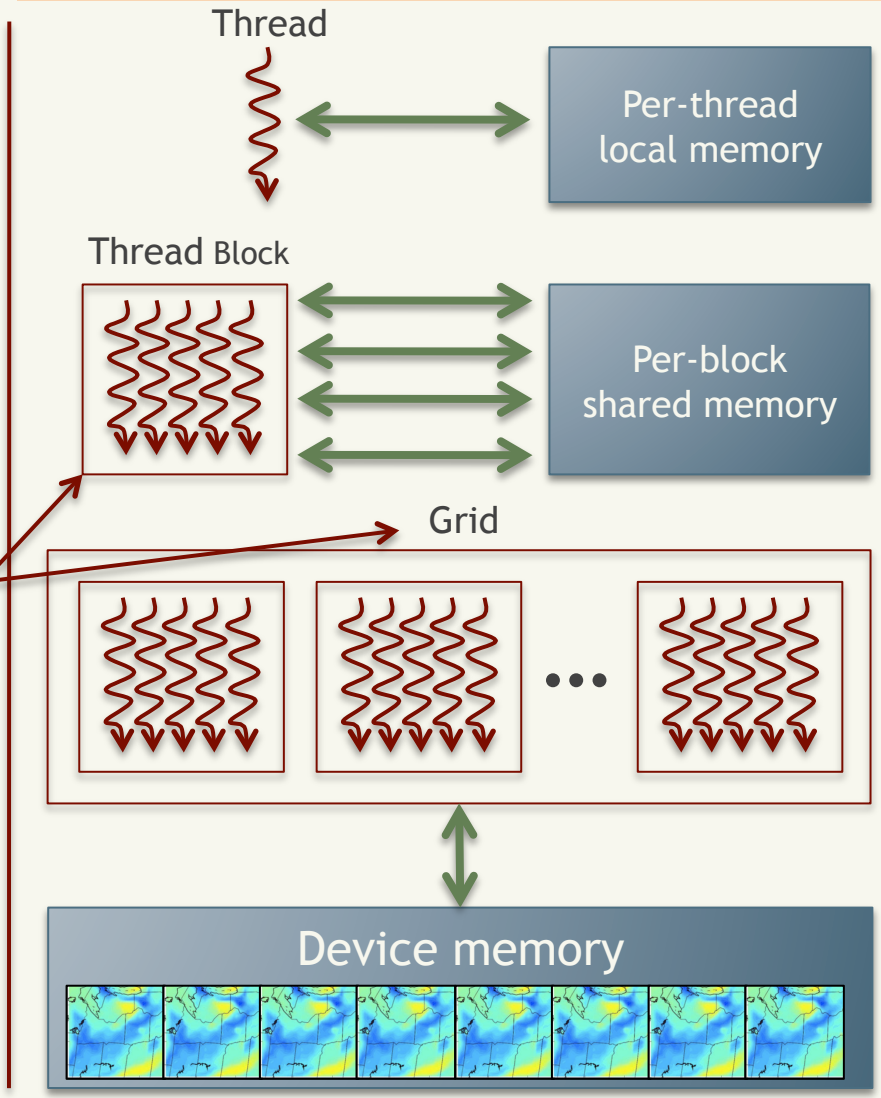
While $t \le t_{end}$

- $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
- $Jac_0 \leftarrow J(t,y)$
- $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
- For $s \leftarrow 1,2,\ldots,n$
  - Compute $Stage_s$ from Fc
  - Solve for $Stage_s$ implicitl
  - Update k(t,y) with meteo
  - Update Fcn from $Stage_{1..}$
- Compute Y_{new} from $Stage_{1\ldots s}$
- Compute error term $E = \max(E_1, E_2, \ldots, E_{vn})$
- If $E \ge \delta$ then discard iteration, reduce h, restart
- Otherwise, $t \leftarrow t + h$ and proceed to next step

Finish: Result in Y_{new}

Block size, grid size, etc. are optimized for each kernel

Device



RADM2 on CUDA: multi-kernel implementation

Host

Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)

Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$

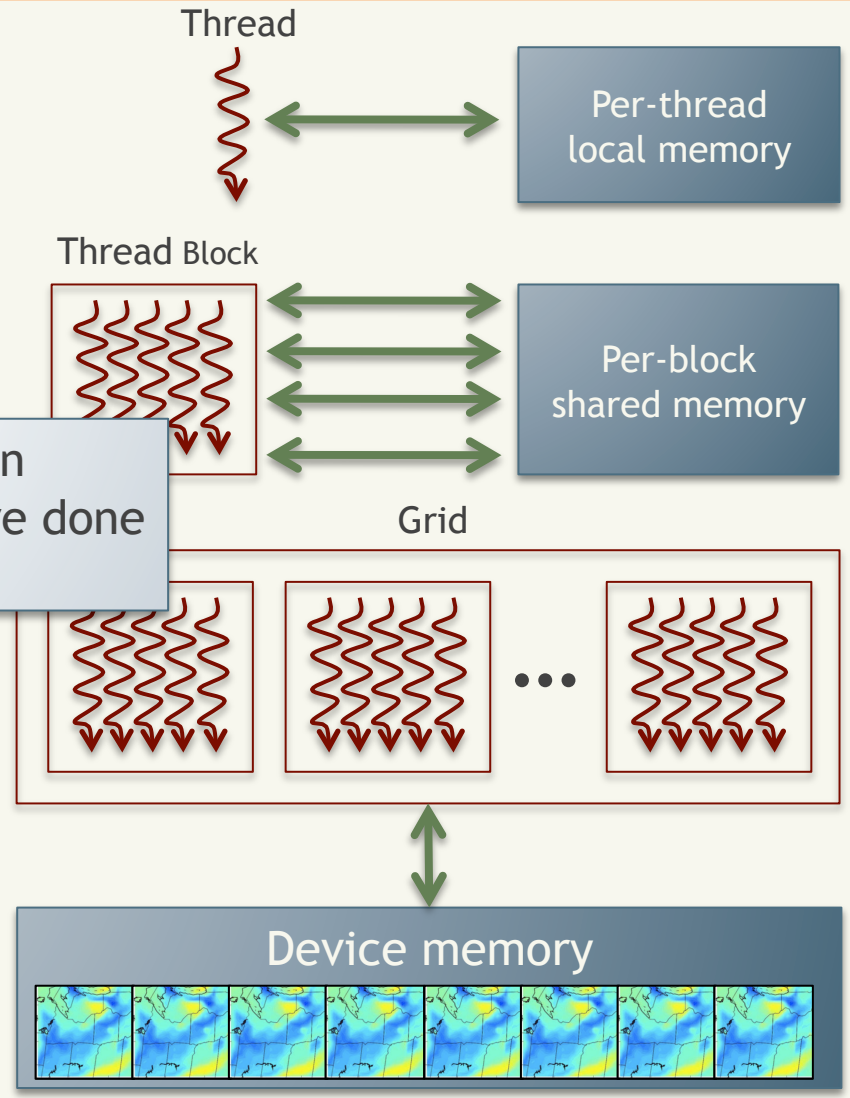
While $t \le t_{end}$

- $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
- $Jac_0 \leftarrow J(t,y)$
- $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
- For $s \leftarrow 1,2,\ldots,n$
  - Compute $Stage_s$ from Fcn and $Stage_{1\ldots(s-1)}$
  - Solve for $Stage_s$ implicitly using G
  - Update k(t,y) with meteorology (ρ, t, q, p)
  - Update Fcn from $Stage_{1\ldots s}$
- Compute Y_{new} from $Stage_{1\ldots s}$
- Compute error term $E = \max(E_1, E_2, \ldots, E_{vn})$
- If $E \ge \delta$ then discard iteration, reduce h, restart
- Otherwise, $t \leftarrow t + h$ and proceed to next step

Finish: Result in Y_{new}

Whole-domain reductions are done on the host

Device



RADM2 on CUDA: multi-kernel implementation

Host

Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)

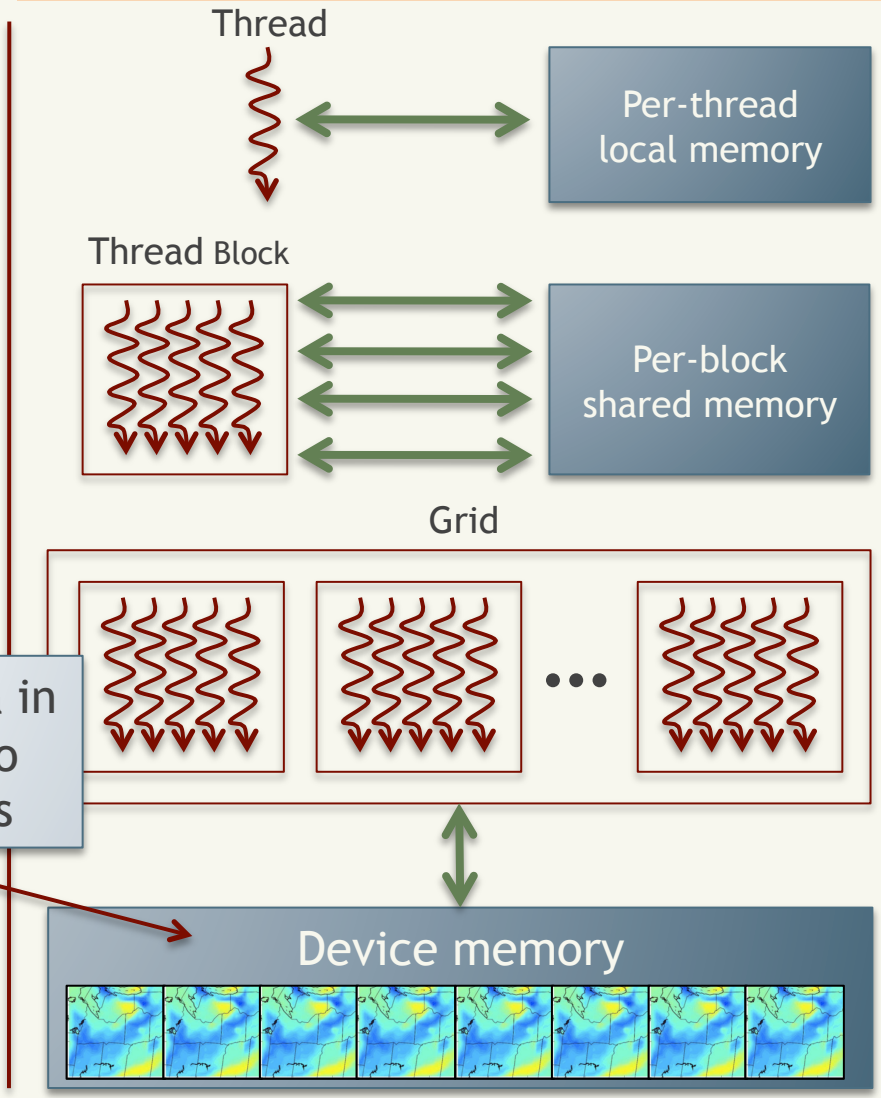
Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$

While $t \le t_{end}$

- $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
- $Jac_0 \leftarrow J(t,y)$
- $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
- For $s \leftarrow 1,2,\ldots,n$
  - Compute $Stage_s$ from Fcn and $Stage_{1\ldots(s-1)}$
  - Solve for $Stage_s$ implicitly using G
  - Update k(t,y) with meteorology (ρ, t, q, p)
  - Update Fcn from $Stage_{1\ldots s}$
- Compute Y_{new} from $Stage_{1\ldots s}$
- Compute error term $E = \max(E_1, E_2$
- If $E \ge \delta$ then discard iteration, reduc
- Otherwise, $t \leftarrow t + h$ and proceed to next step

Finish: Result in Y_{new}

Device



Reorganized data in device memory to coalesce accesses

RADM2 on CUDA: multi-kernel implementation

Host

Initialize $k(t,y)$ from starting concentrations and meteorology (ρ, t, q, p)

Initialize time variables $t \leftarrow t_{start}$, $h \leftarrow 0.1 \times (t_{end} - t_{start})$

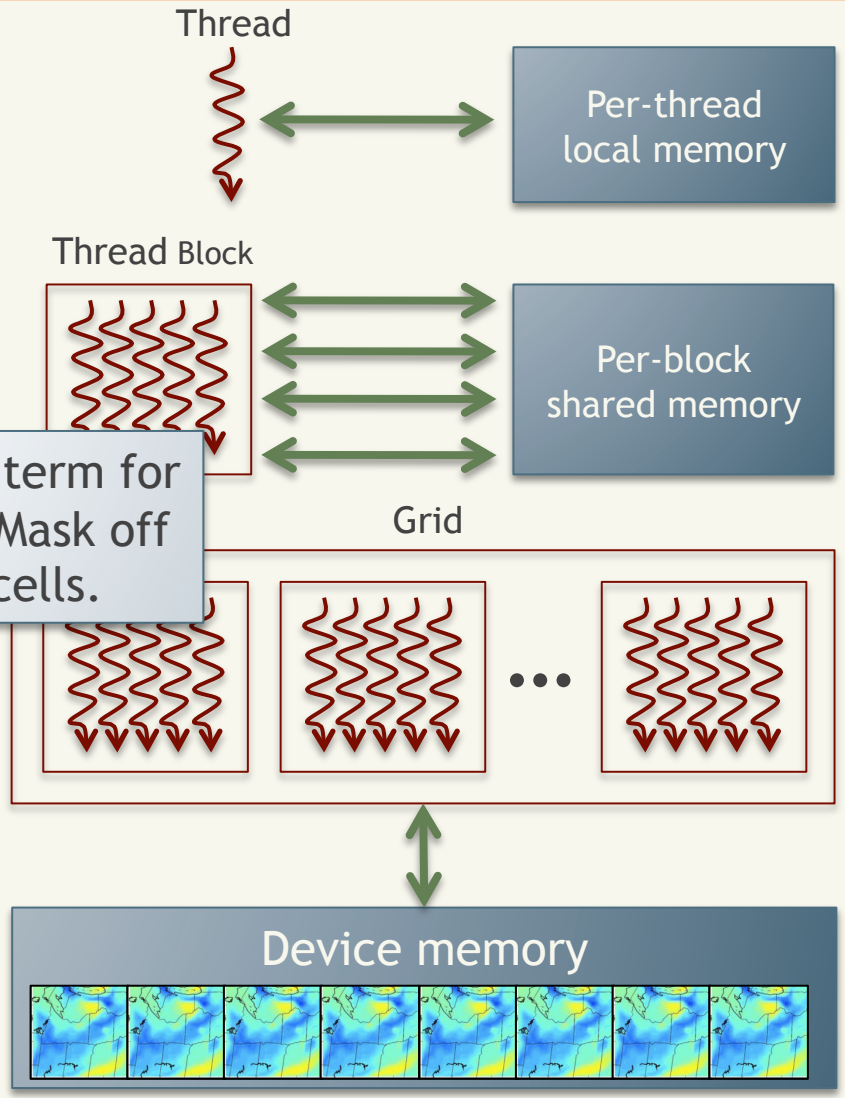
While $t \le t_{end}$

- $Fcn_0 \leftarrow Fcn \leftarrow f(t,y)$
- $Jac_0 \leftarrow J(t,y)$
- $G \leftarrow \text{LU\_DECOMP}(\frac{1}{h\gamma} - Jac_0)$
- For $s \leftarrow 1,2,\ldots,n$
  - Compute $Stage_s$ from Fcn and $Stage_{1\ldots(s-1)}$
  - Solve for $Stage_s$ implicitly using G
  - Update k(t,y) with meteorology (ρ, t, q, p)
  - Update Fcn from $Stage_{1\ldots s}$
- Compute Y_{new} from $Stage_{1\ldots s}$
- Compute error term $E = \max(E_1, E_2, \ldots, E_{vn})$
- If $E \ge \delta$ then discard iteration, reduce h, restart
- Otherwise, $t \leftarrow t + h$ and proceed to next step

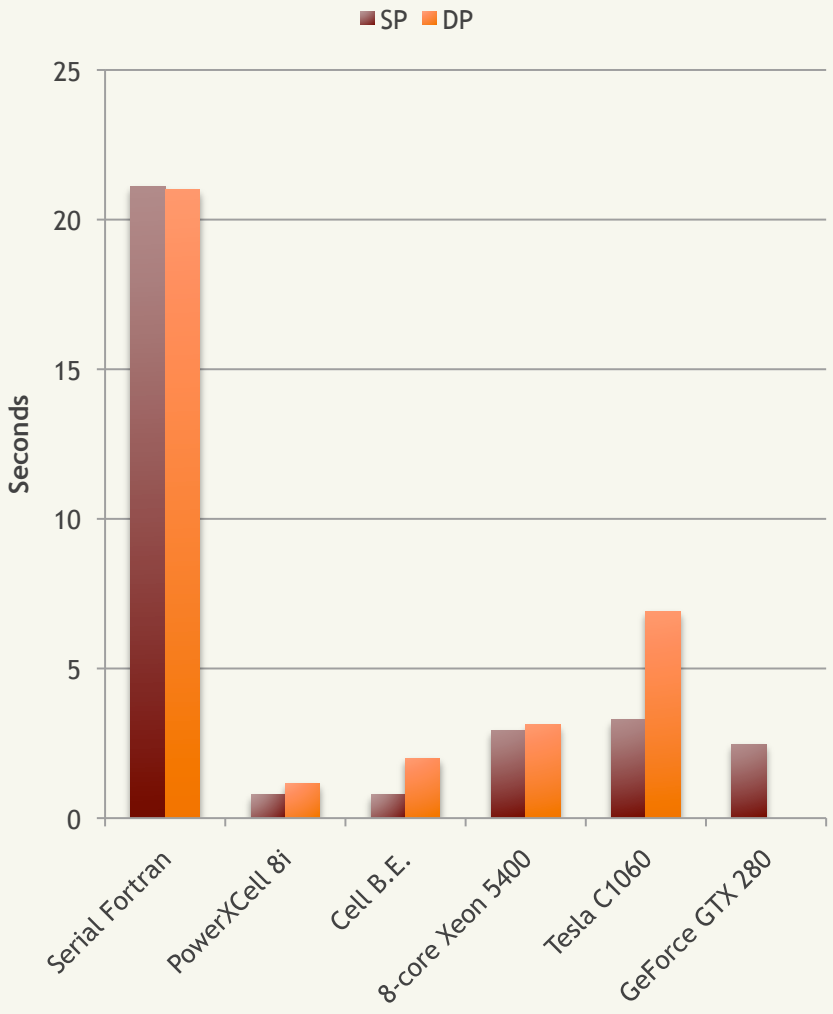
Finish: Result in Y_{new}

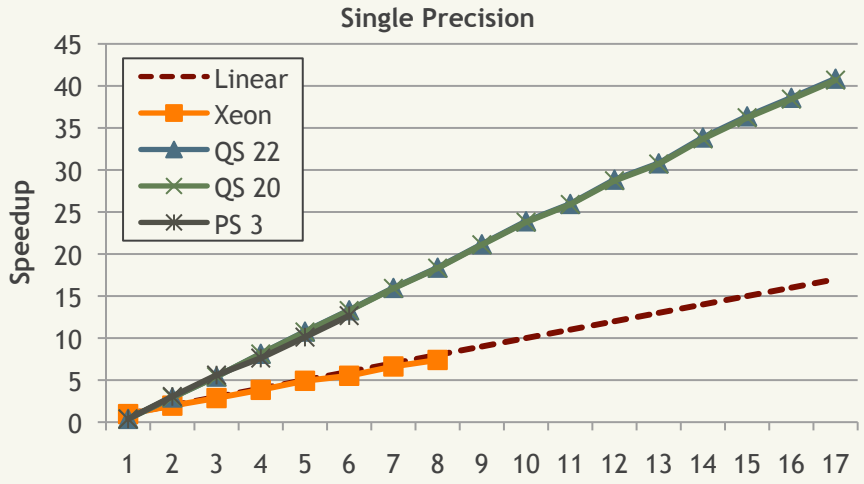
Store error term for each cell. Mask off converged cells.

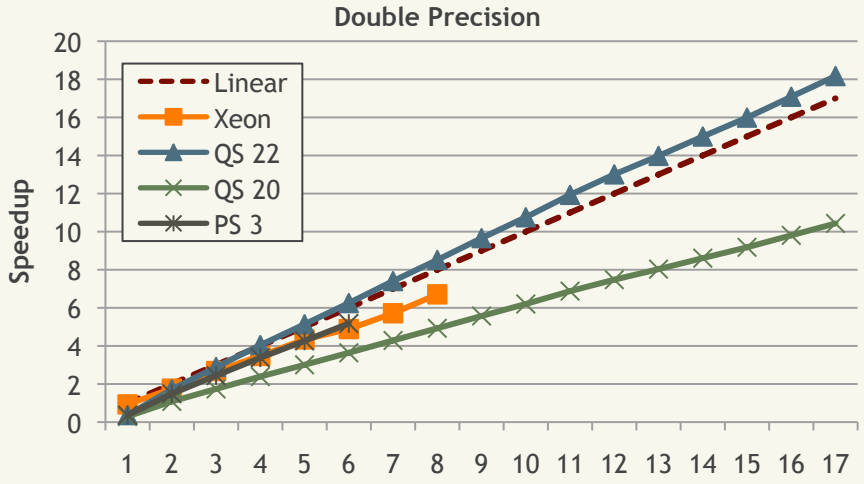
Device



Performance: WRF/Chem RADM2 mechanism







Summary analysis and comparison

PowerXCell 8i (CBEA)

- ☺ Explicitly-managed memory hierarchy and “large” on-chip memory results in good performance.
- ☹ The lack of high-level language features, mature tools, and support make the CBEA hard to program.
- ☹ A good understanding of the CBEA is required.

Tesla C1060 (GTX 280)

- ☹ Small on-chip memory and high memory latency results in disappointing performance.
- ☺ CUDA makes GPGPU programming easy and intuitive.
- ☺ A good understanding of the GTX 200 architecture is strongly encouraged.

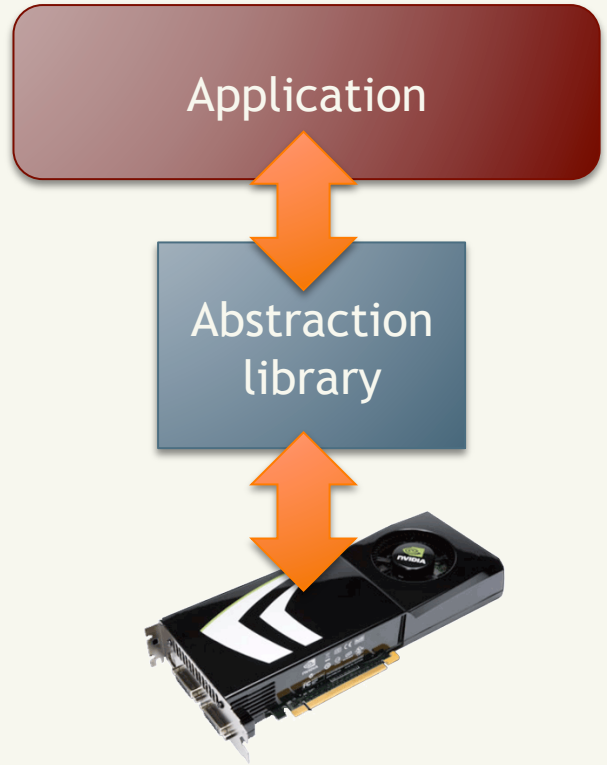
Future Work

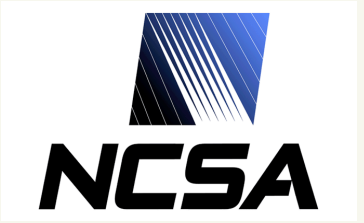
Multi-core KPP

- Find common features and describe them to the code generation routines
  - Instruction cardinality
  - Memory access patterns
- Use solvers from RADM2 case study as a generic template
- Already can do SAPRC…

Accelerated function offloading

- IBM Dynamic Application Virtualization (DAV)





http://www.mmm.ucar.edu/wrf/WG2/GPU/Chem.htm