Funded by:
U.S. Department of Energy
Vehicle Technologies Program
Program Manager: Gurpreet Singh & Leo Breton

Practical Combustion Kinetics with CUDA

GPU Technology Conference
March 20, 2015

Russell Whitesides & Matthew McNenly

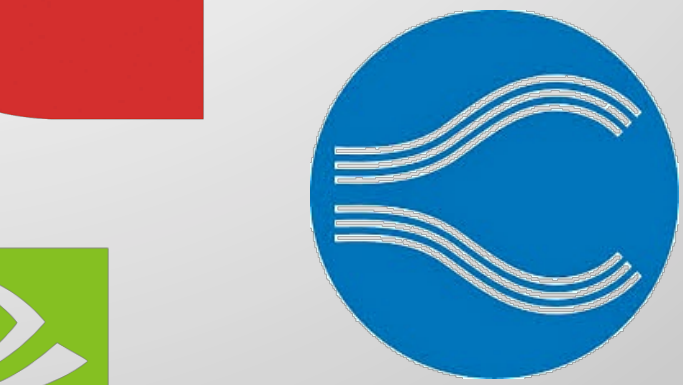
Lawrence Livermore National Laboratory

Session S5468

LLNL-PRES-668639
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Collaborators

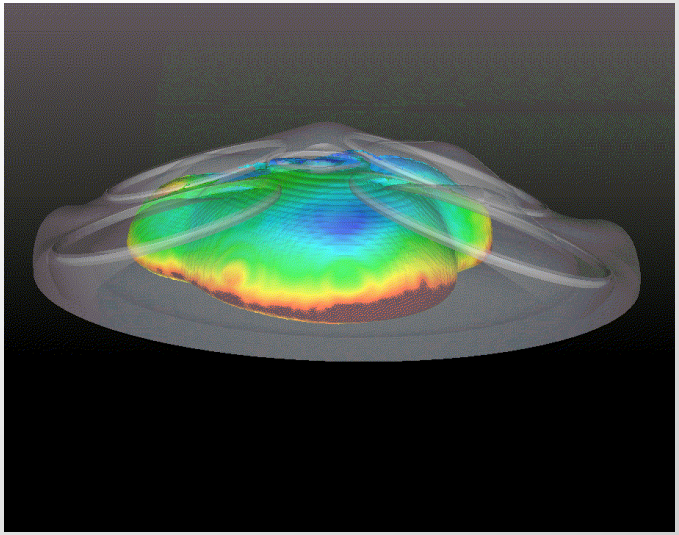
- Cummins Inc.
- Convergent Science
- NVIDIA
- Indiana University



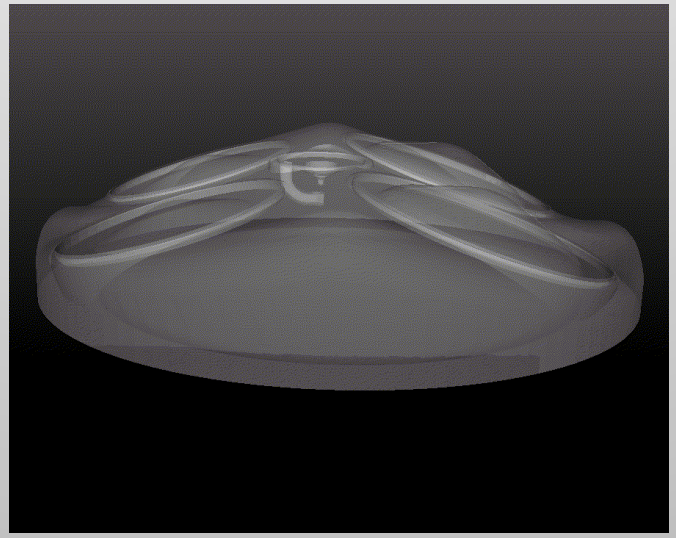
Good guys to work with.

Does

plus



equal

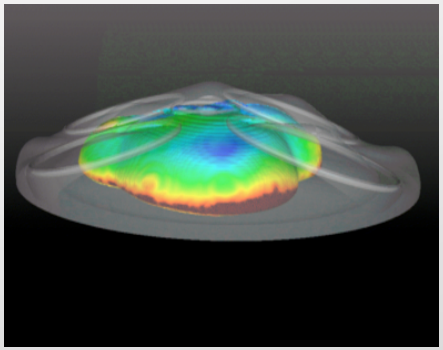


?

The big question.

Lots of smaller questions:

?



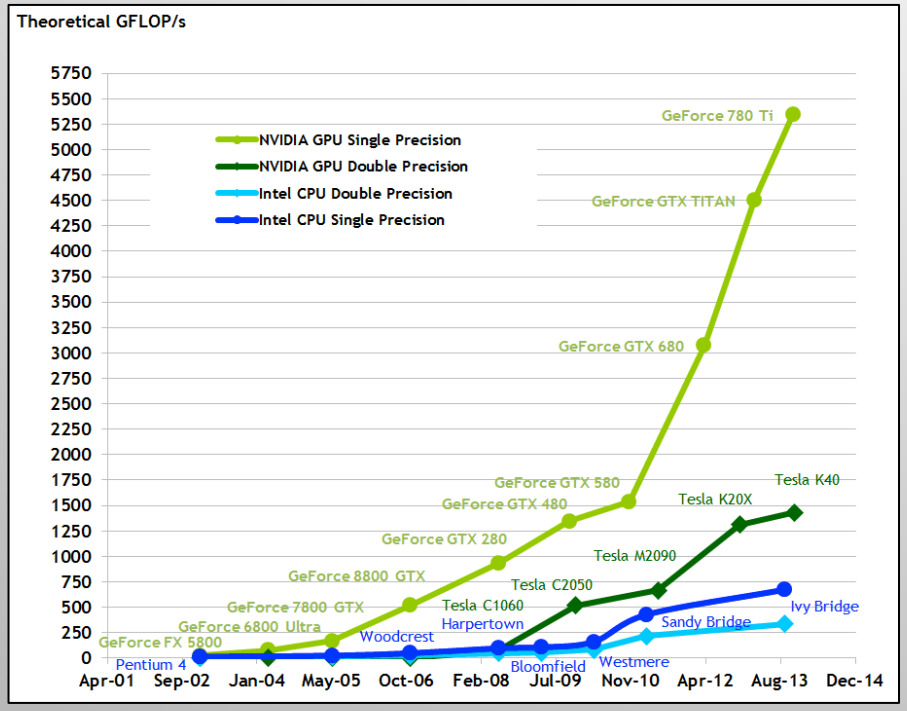
?

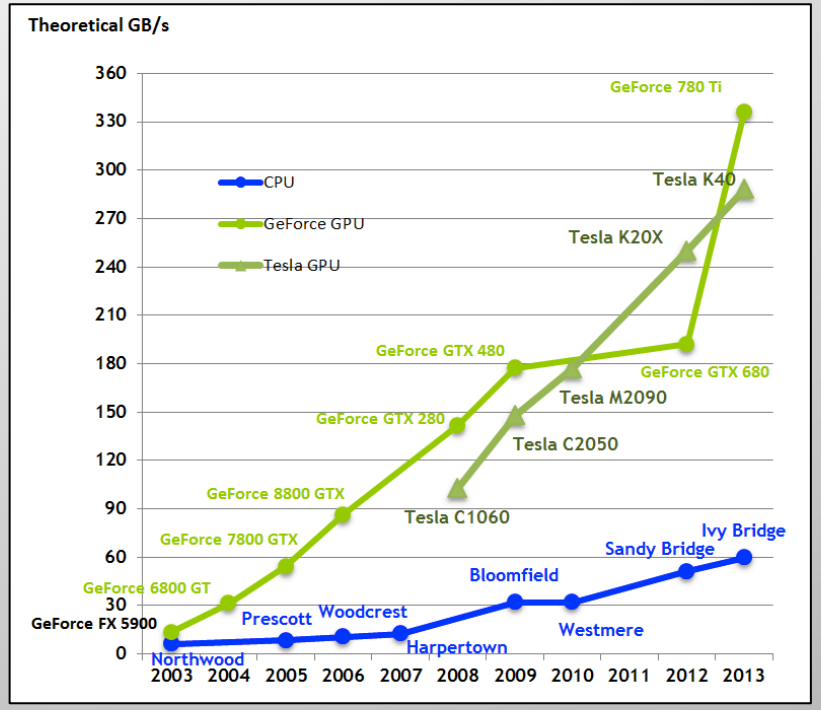
- What has already been done in this area?
- How are we approaching the problem?
- What have we accomplished?
- What's left to do?

There won't be a quiz at the end.

? NVIDIA GPUs/CUDA Toolkit

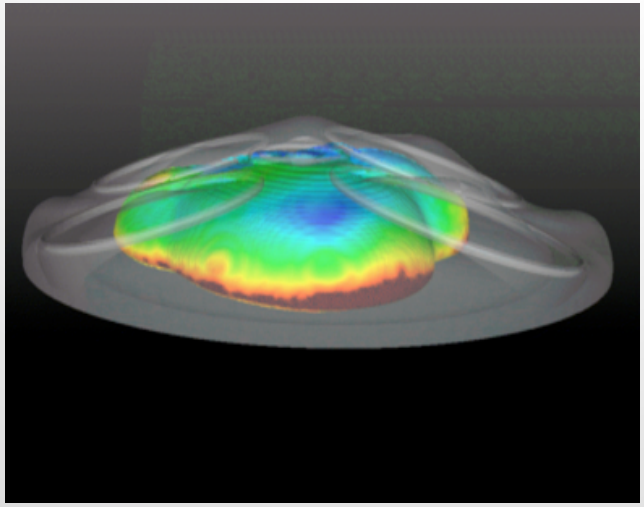
Why?





Data from NVIDIA's, *CUDA C Programming Guide Version 6.0*, 2014.

More FLOP/s, More GB/s, Faster Growth in Both.



?

- Reacting flow simulation
- Computational Fluid Dynamics (CFD)
- Detailed chemical kinetics
- Tracking 10-1000's of species
- ConvergeCFD (internal combustion engines)

Approach also used to simulate gas turbines, burners, flames, etc.

What has been done already in combustion kinetics on GPU's?

Recent review by Niemeyer & Sung [1]:

- Spafford, Sankaran & co-workers (ORNL) (first published 2010)
- Shi, Green & co-workers (MIT)
- Stone (CS&E LLC)
- Niemeyer & Sung (CWRU/OSU, UConn)

Most approaches use explicit or semi-implicit Runge-Kutta techniques
Some only use GPU for derivative calculation

From [1]:

> *"Furthermore, no practical demonstration of a GPU chemistry solver capable of handling stiff chemistry has yet been made. This is one area where efforts need to be focused."*

[1] K.E. Niemeyer, C.-J. Sung, Recent progress and challenges in exploiting graphics processors in computational fluid dynamics, J Supercomput. 67 (2014) 528–564. doi:10.1007/s11227-013-1015-7.

A few groups working (publicly) on this. Some progress has been made.

Problem: Can't directly port CPU chemistry algorithms to GPU

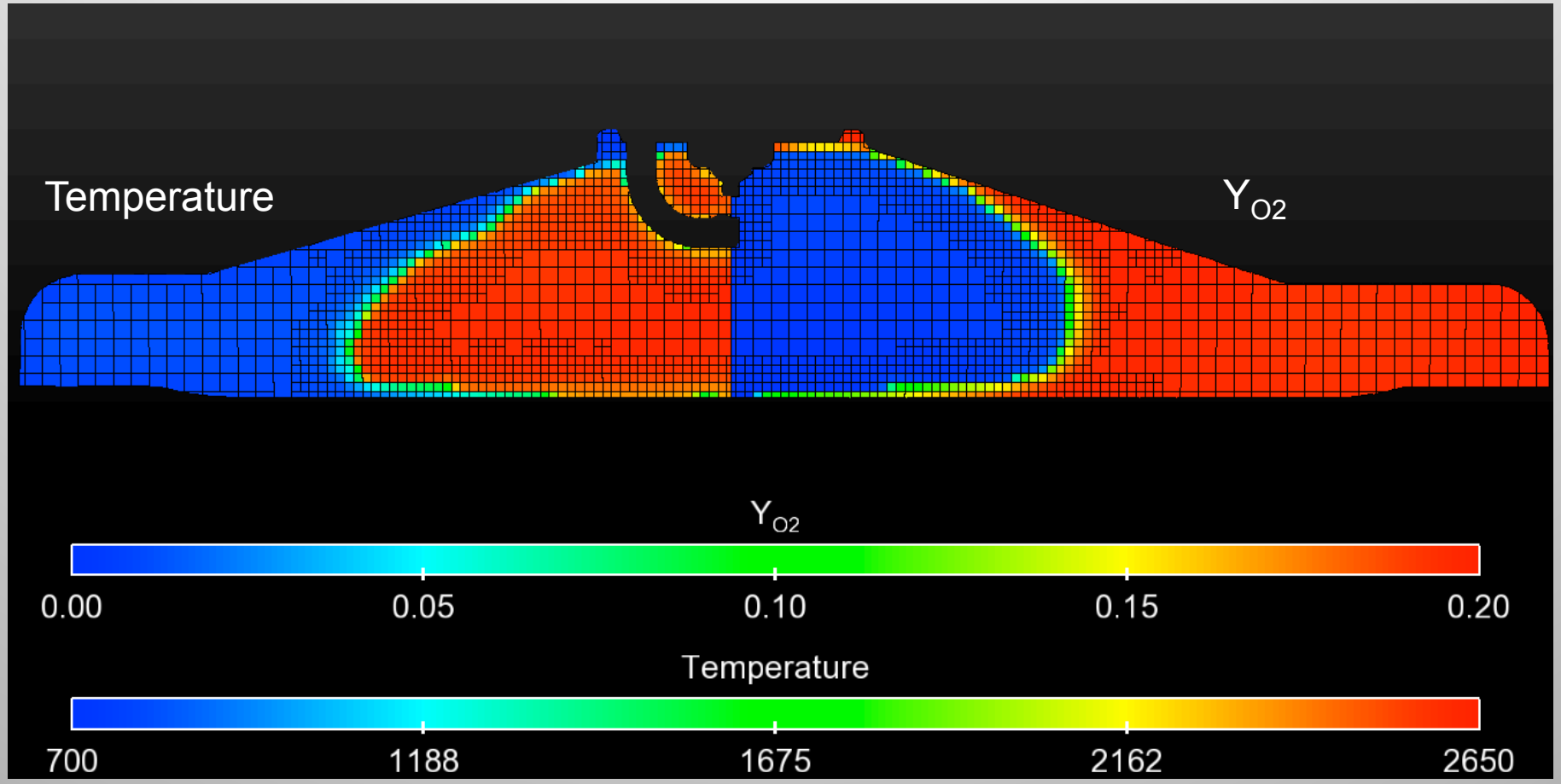
- GPUs need dense data and lots of it.
- Large chemical mechanisms are sparse.
- Small chemical mechanisms don't have enough data.

  (even large mechanisms aren't large *in GPU context*)

Solution: Re-frame many uncoupled reactor calculations into a single system of coupled reactors.

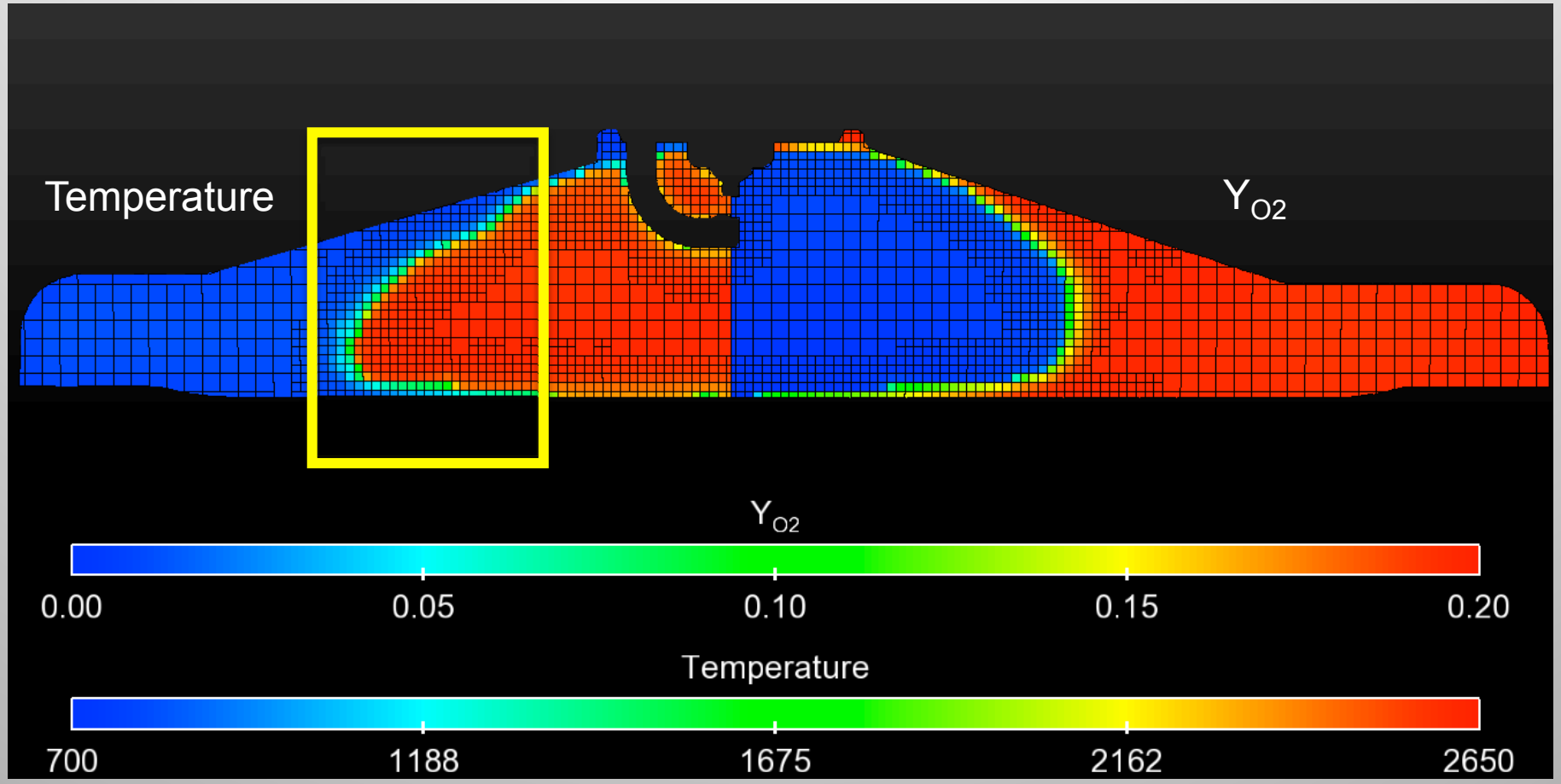
For chemistry it's not as simple as adding new hardware.

How do we solve chemistry on the CPU?



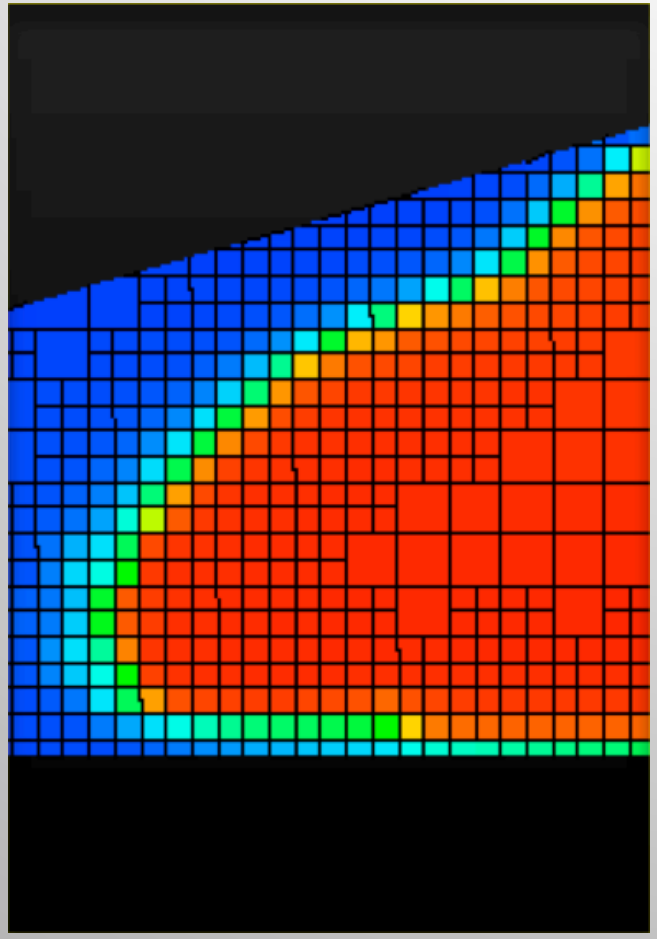
Example: Engine Simulation in Converge CFD

How do we solve chemistry on the CPU?



Example: Engine Simulation in Converge CFD

Detailed Chemistry in Reacting Flow CFD:

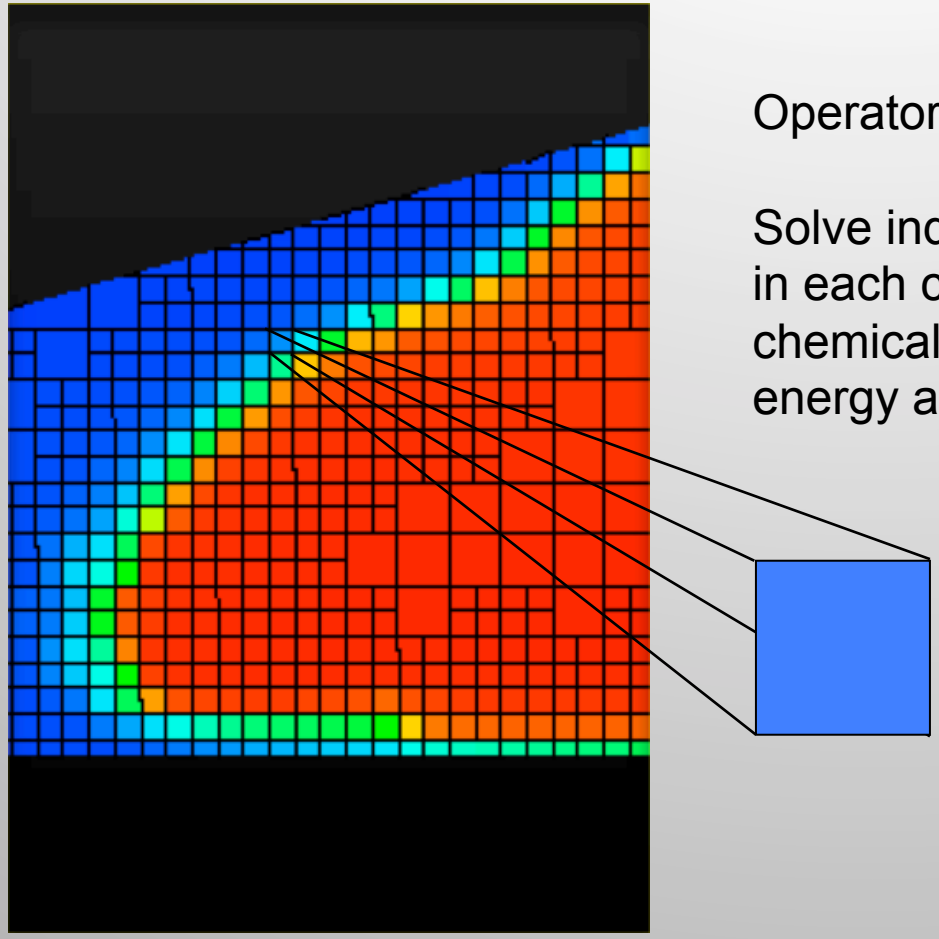


Operator Splitting Technique:

Solve independent Initial Value Problem in each cell (or zone) to calculate chemical source terms for species and energy advection/diffusion equations.

Each cells is treated as an isolated system for chemistry.

Detailed Chemistry in Reacting Flow CFD:

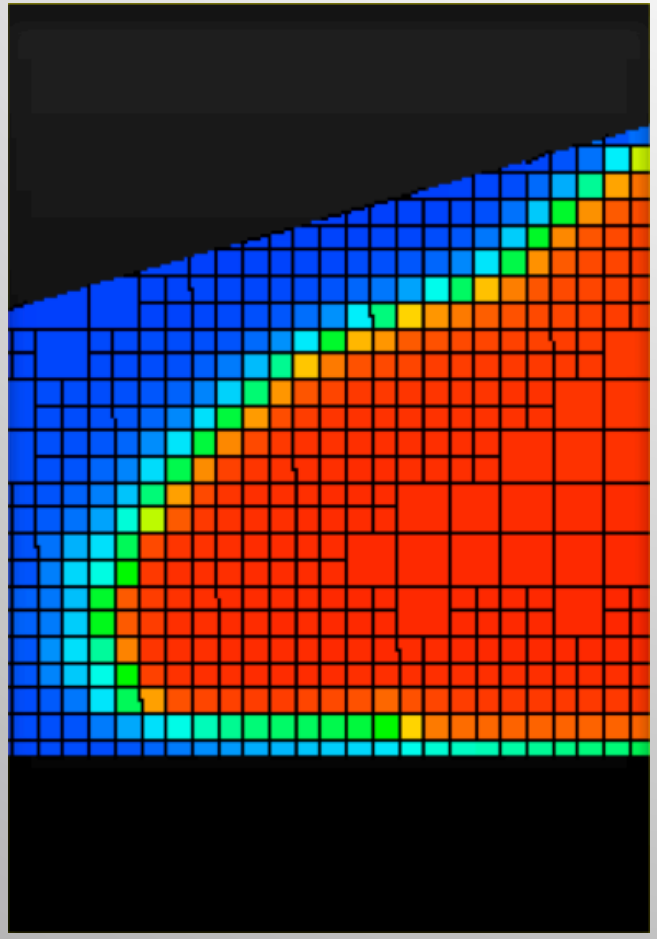


Operator Splitting Technique:

Solve independent Initial Value Problem in each cell (or zone) to calculate chemical source terms for species and energy advection/diffusion equations.

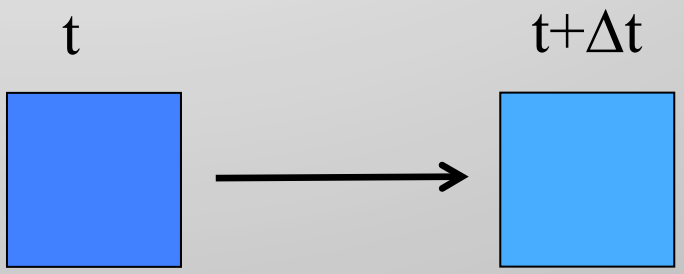
Each cells is treated as an isolated system for chemistry.

Detailed Chemistry in Reacting Flow CFD:



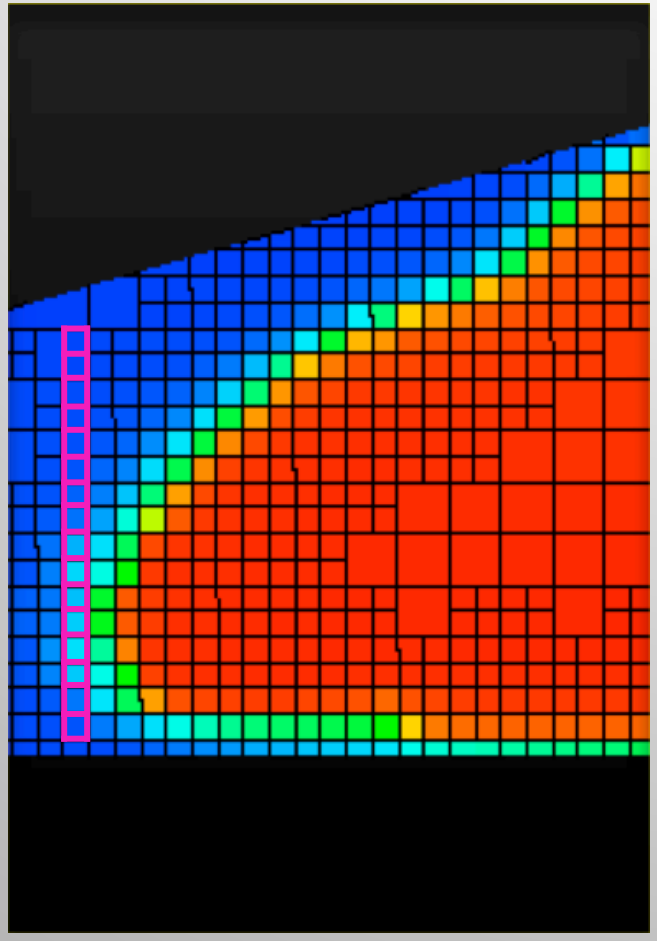
Operator Splitting Technique:

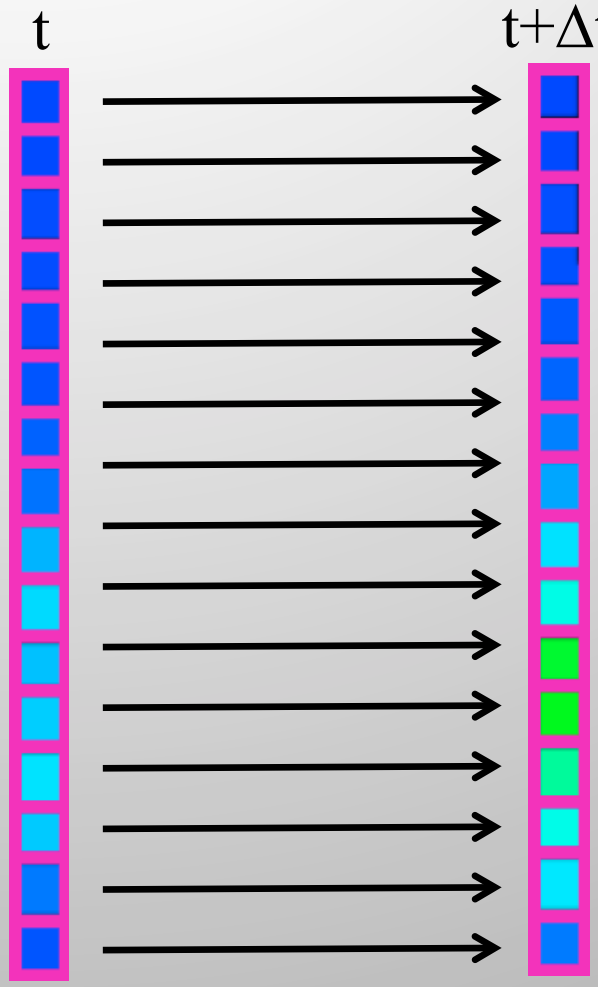
Solve independent Initial Value Problem in each cell (or zone) to calculate chemical source terms for species and energy advection/diffusion equations.



Each cells is treated as an isolated system for chemistry.

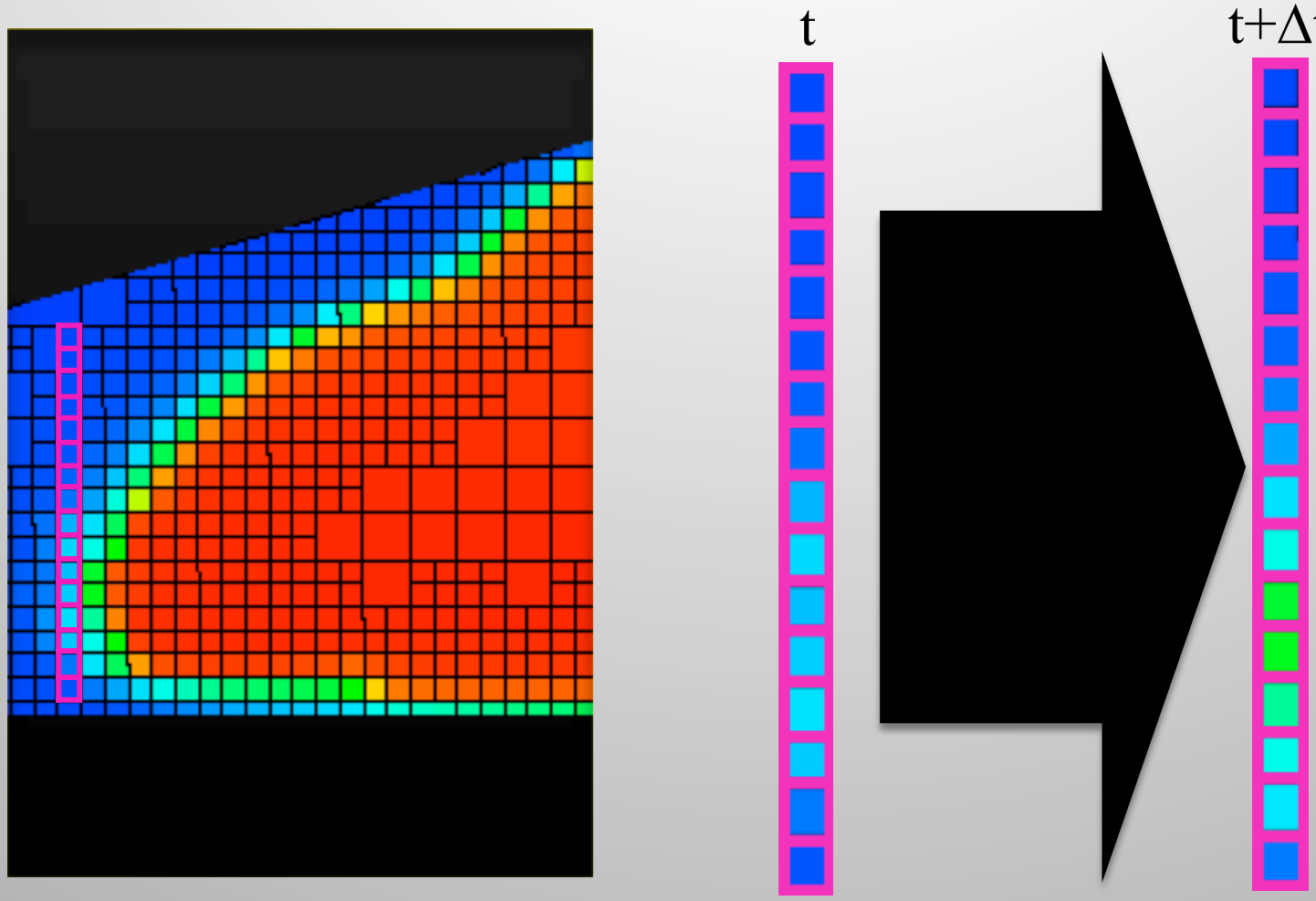
CPU (un-coupled) chemistry integration





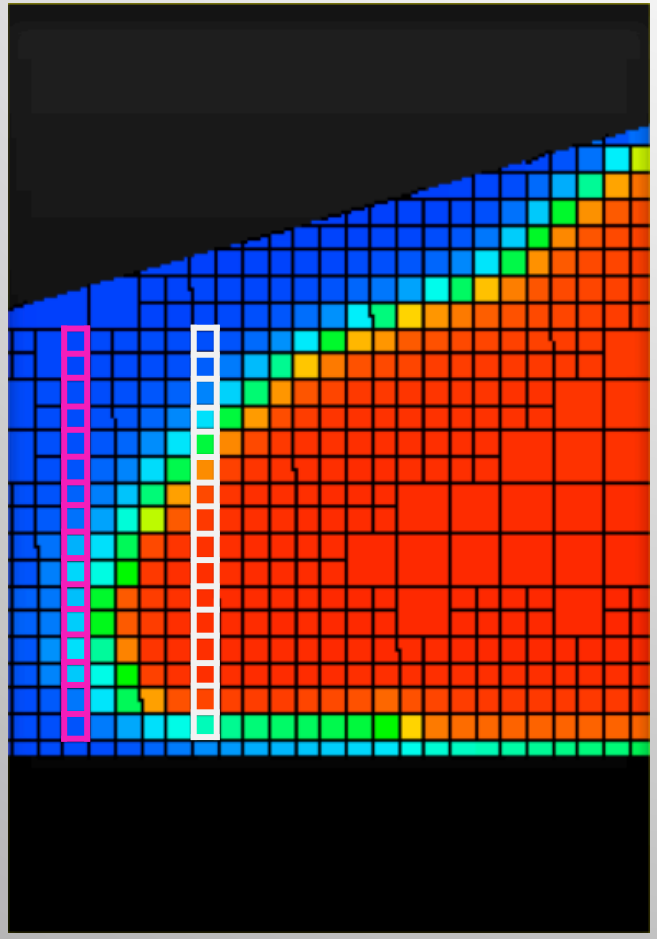
Each cells is treated as an isolated system for chemistry.

GPU (coupled) chemistry integration



For the GPU we solve chemistry simultaneously in large groups of cells.

What about variations in practical engine CFD?



VS.

If the systems are not similar how much extra work needs to be done?

What are the equations we're trying to solve?

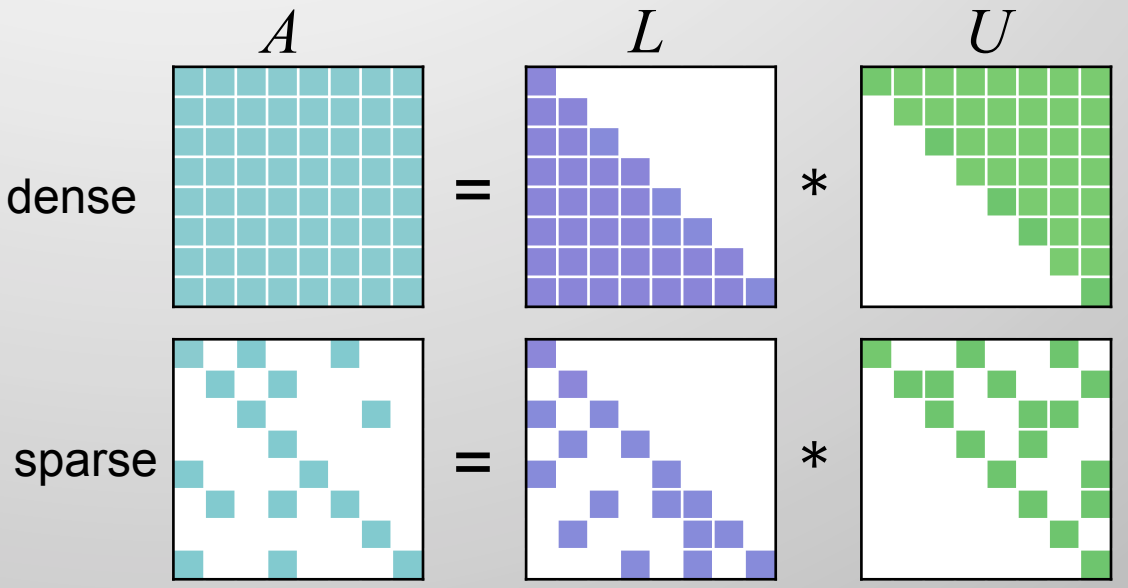
Derivative Equations
(vector calculations)

$$\frac{dy_i}{dt} = \frac{w_i}{\rho}\frac{dC_i}{dt}$$

$$\frac{dT}{dt} = -\frac{RT}{\rho c_v}\sum_{i}^{species} u_i \frac{dC_i}{dt}$$

Derivative represents system of equations to be solved (perfectly stirred reactor).

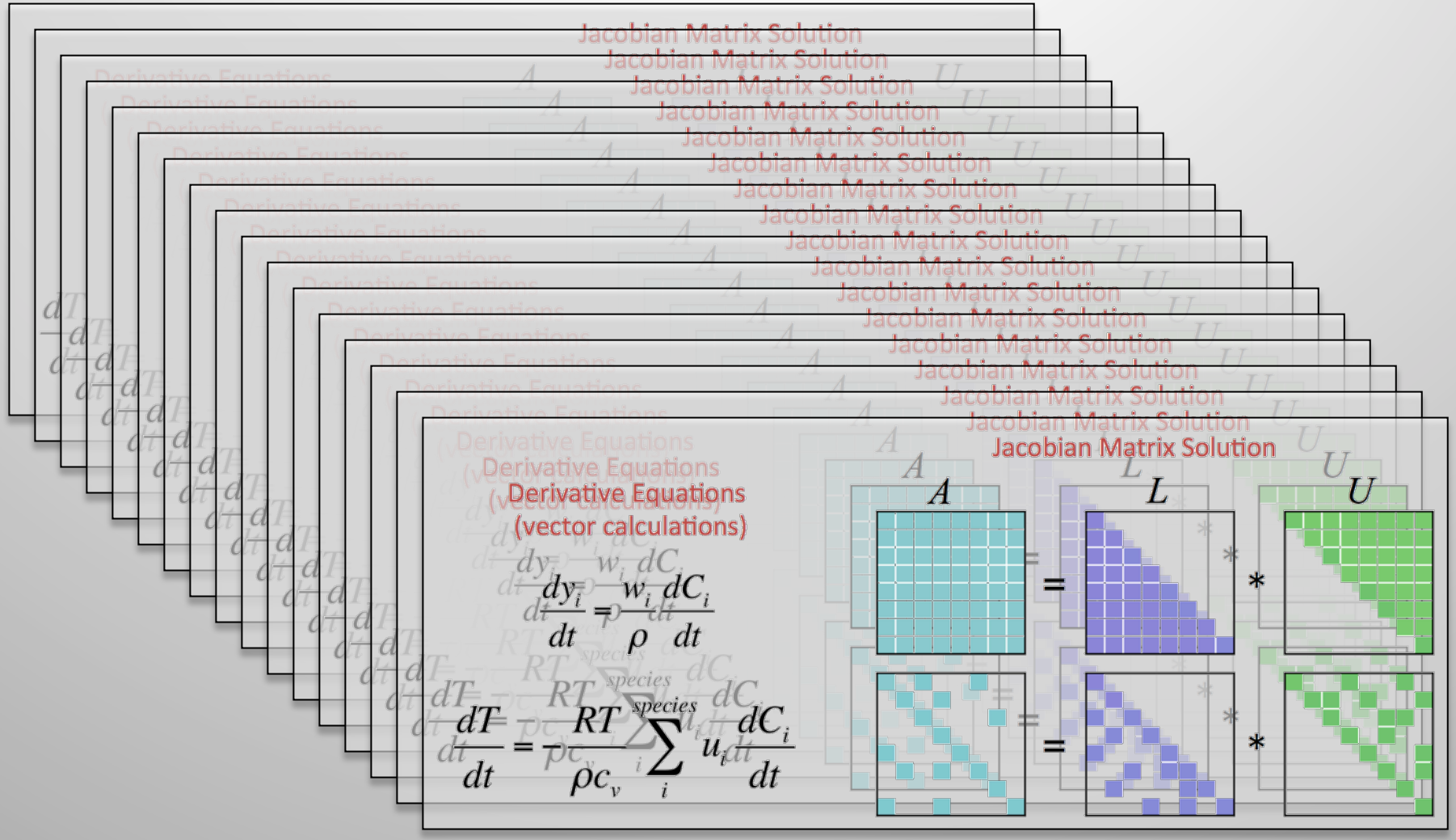
Jacobian Matrix Solution



- Matrix solution required due to stiffness
- Matrix storage in dense or sparse formats

Significant effort to transform fastest CPU algorithms to GPU appropriate versions.

We want to solve many of these simultaneously



Not as easy as copy and paste.

Example: Species production rates

Net rates of production

$$\frac{dC_i}{dt} = \sum_j^{create} R_j - \sum_j^{destroy} R_j$$

Chemical reaction rates of progress

$$R_i = k_i \prod_j^{species} C_j^{\nu_{ij}}$$

Chemical reaction step rate coefficients

Arrhenius Rates

$$k_i = A_i T^{n_i} e^{-\frac{E_{A,i}}{RT}}$$

Equilibrium Reverse Rates

$$k_i = k_{i,f} K_{eq} = k_{i,f} \exp\left(\sum_j^{prod} \frac{G_j^0}{RT} - \sum_j^{reac} \frac{G_j^0}{RT} \right)$$

Third-body enhanced Rates

$$k_i' = k_i \sum_j^{species} \alpha_j C_j$$

Fall-off rates

$$k_i' = k_i \ldots$$

Major component of derivative; Lots of sparse operations.

Example: Species production rates

Net rates of production

$$\frac{dC_i}{dt} = \sum_{j}^{create} R_j - \sum_{j}^{destroy} R_j$$

- Chemical species connectivity
- Generally sparsely connected
- Leads to poor memory locality
- Bad for GPU performance

Chemical reaction rates of

$$R_i = k_i \prod_{j}^{species} C_j^{\nu_{ij}}$$

Chemical reaction step rate coefficients

Arrhenius Rates

$$k_i = A_i T^{n_i} e^{-\frac{E_{A,i}}{RT}}$$

Equilibrium Reverse Rates

$$k_i = k_{i,f} K_{eq} = k_{i,f} \exp\left(\sum_{j}^{prod} \frac{G_j^0}{RT} - \sum_{j}^{reac} \frac{G_j^0}{RT} \right)$$

Third-body enhanced Rates

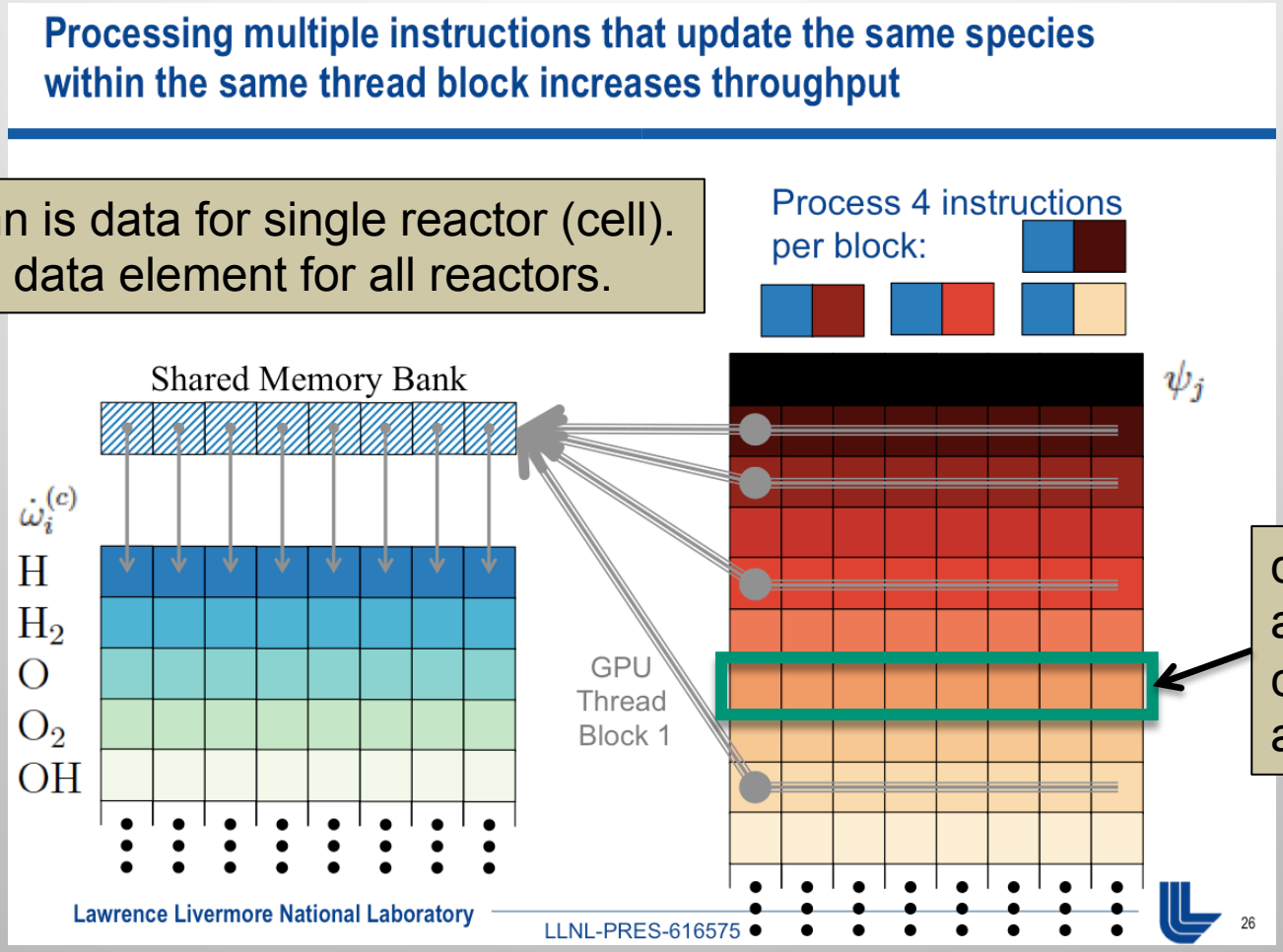
$$k_i' = k_i \sum_{j}^{species} \alpha_j C_j$$

Fall-off rates

$$k_i' = k_i \ldots$$

Major component of derivative; Lots of sparse operations.

Example: Species production rates



Approach: couple together reactors (or cells) and make smart use of GPU memory.

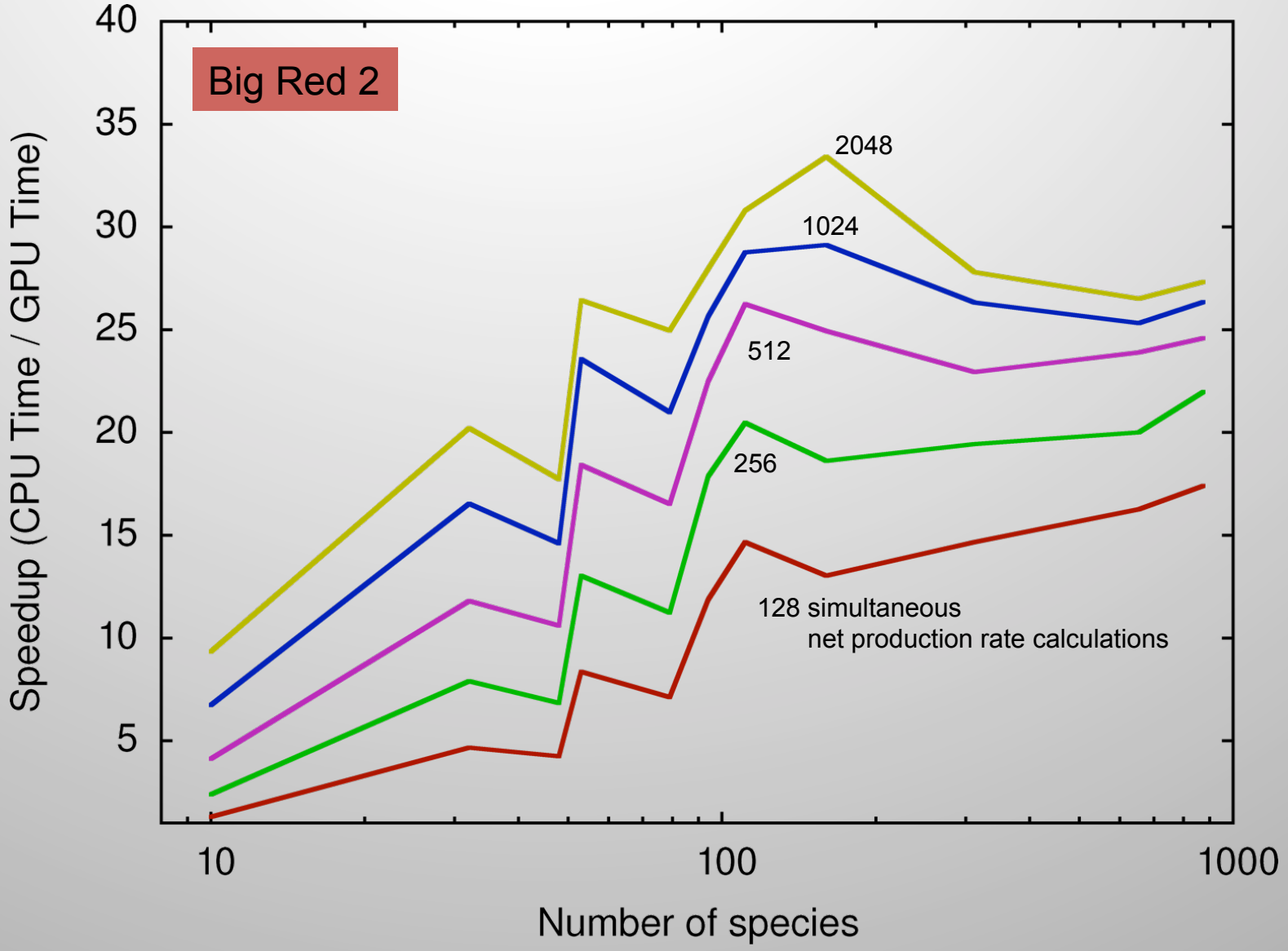
Benchmarking Platforms:

- Big Red 2
  - AMD Opteron Interlagos (16 core)
  - 1x-Tesla K20

- Surface (not pictured)
  - Intel Xeon E5-2670 (16 core)
  - 2x-Tesla K40m

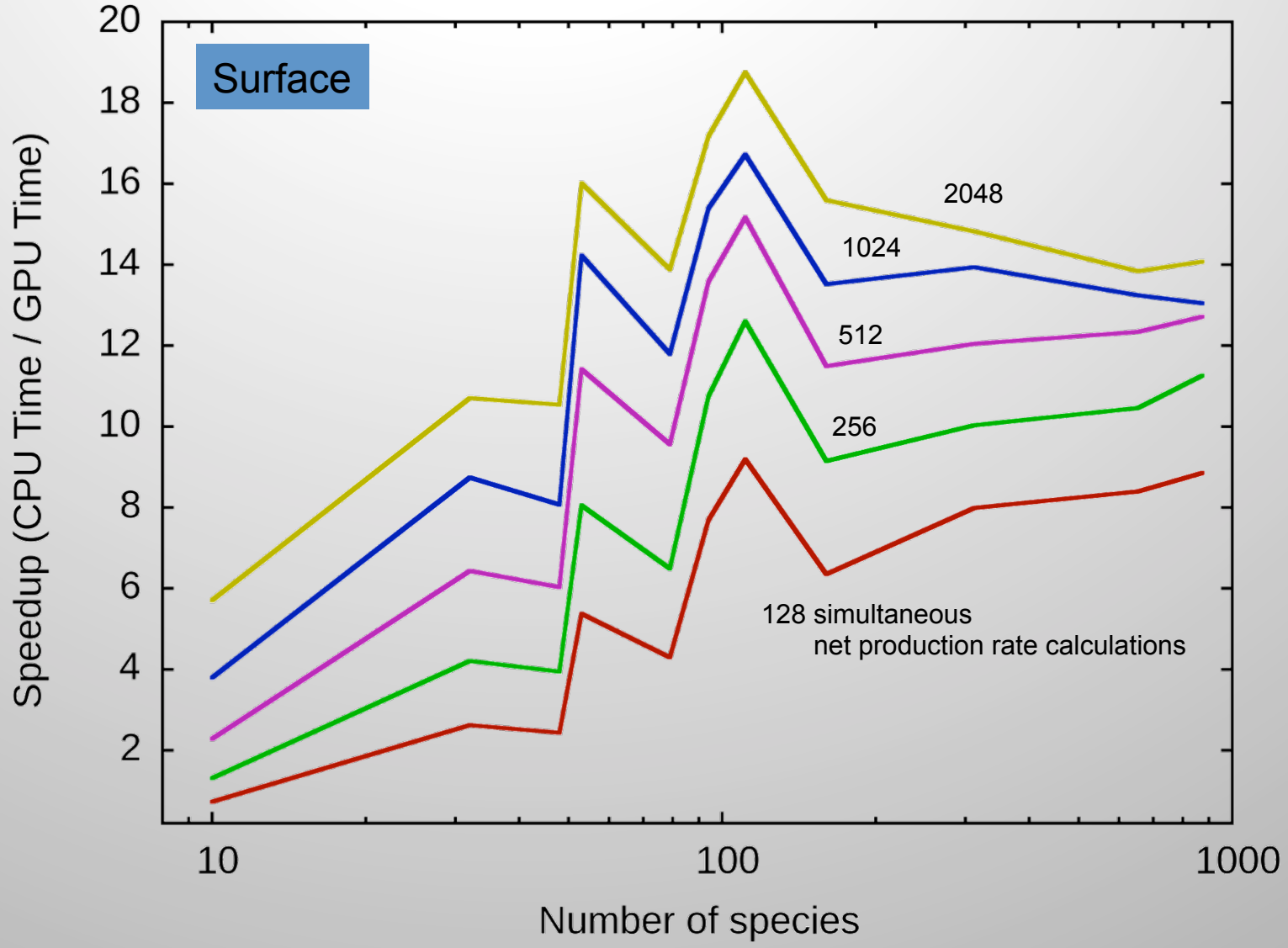
CPU and GPU Used Both Matter

$$\frac{dC_i}{dt}$$



Significant speedup achieved for species production rates.

$$\frac{dC_i}{dt}$$



Less speedup than Big Red 2 because the CPU is faster.

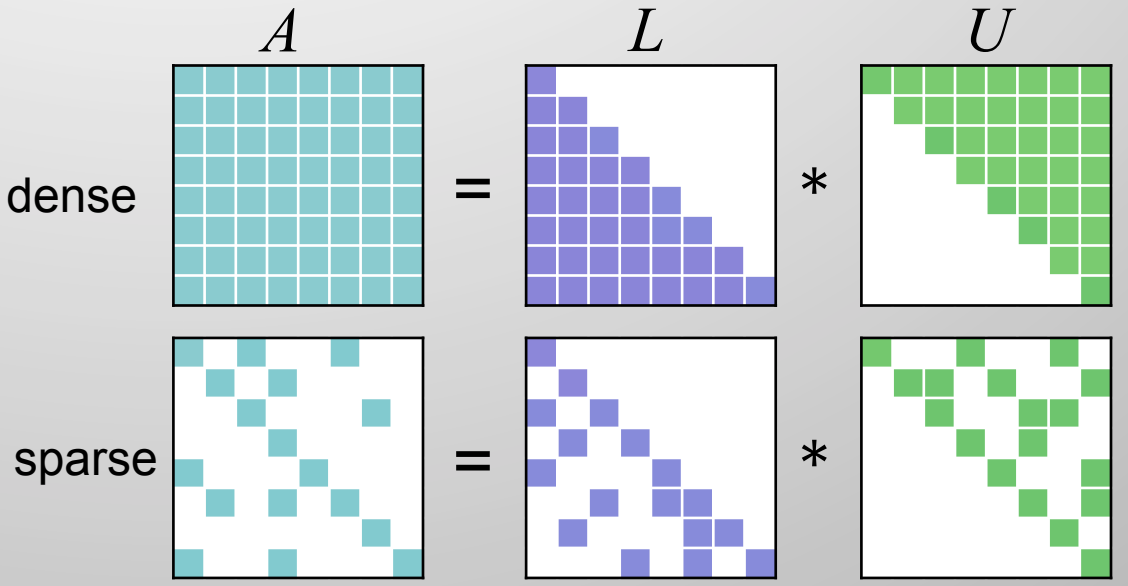
We have implemented or borrowed algorithms for the rest of the chemistry integration.

Derivative Equations
(vector calculations)

$$\frac{dy_i}{dt} = \frac{w_i}{\rho}\frac{dC_i}{dt}$$

$$\frac{dT}{dt} = -\frac{RT}{\rho c_v}\sum_{i}^{species} u_i \frac{dC_i}{dt}$$

Jacobian Matrix Solution



Need to put the rest of the calculations on the GPU.

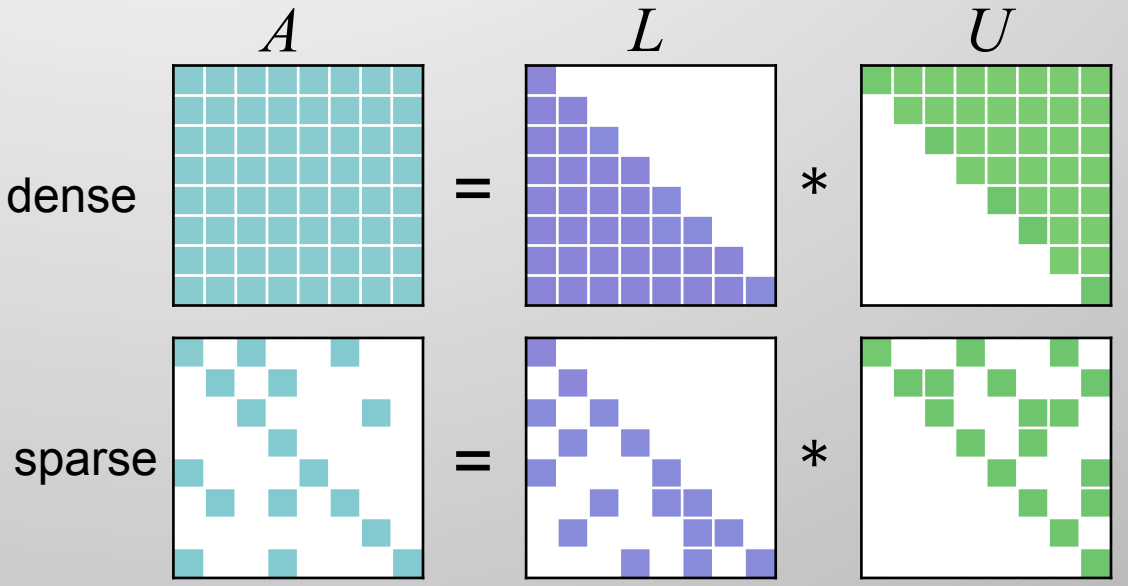
We have implemented or borrowed algorithms for the rest of the chemistry integration.

Derivative Equations
(vector calculations)

$$\frac{dy_i}{dt} = \frac{w_i}{\rho c_v} \frac{dC_i}{dt}$$

Apart from dC_i/dt, derivative is straightforward on GPU.

Jacobian Matrix Solution



Need to put the rest of the calculations on the GPU.

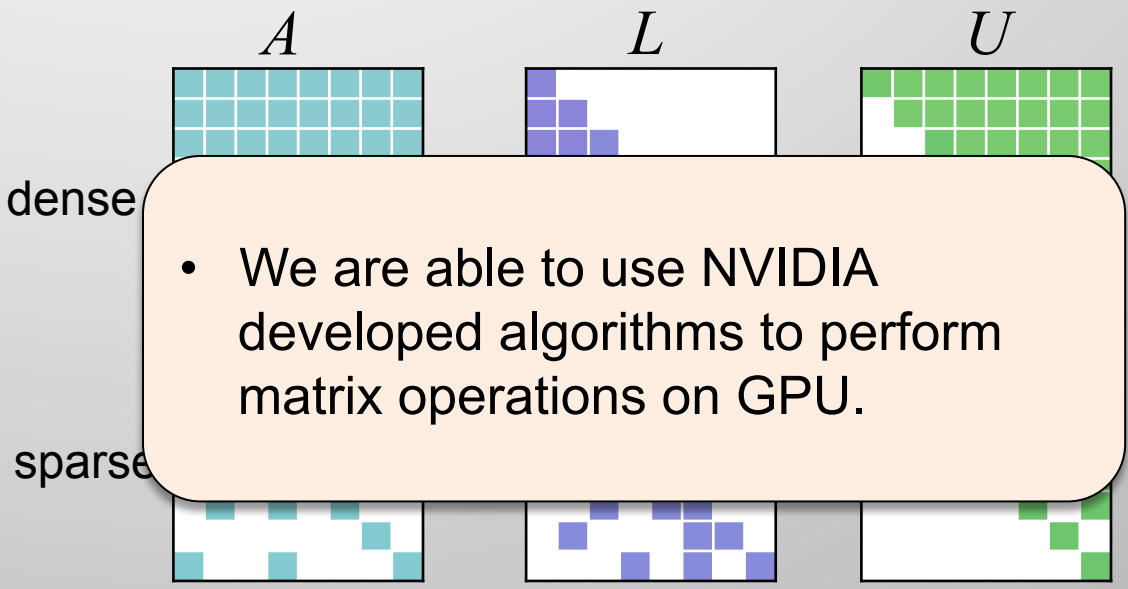
We have implemented or borrowed algorithms for the rest of the chemistry integration.

Derivative Equations (vector calculations)

$$\frac{dy_i}{dt} = \frac{w_i}{\rho c_v} \sum_i \frac{dC_i}{dt}$$

Apart from dC_i/dt, derivative is straightforward on GPU.

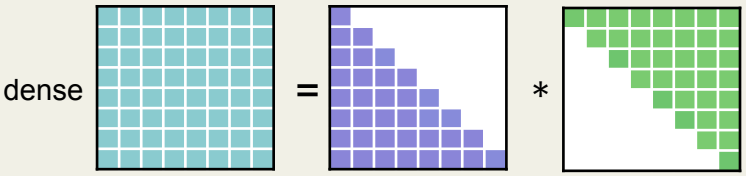
Jacobian Matrix Solution



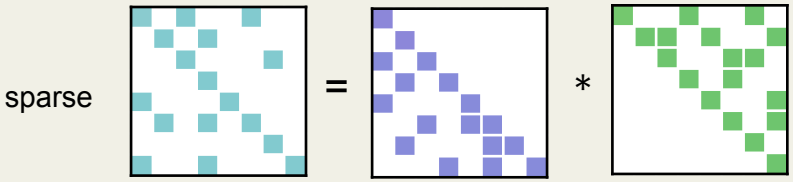
- We are able to use NVIDIA developed algorithms to perform matrix operations on GPU.

Need to put the rest of the calculations on the GPU.

Matrix Solution Methods



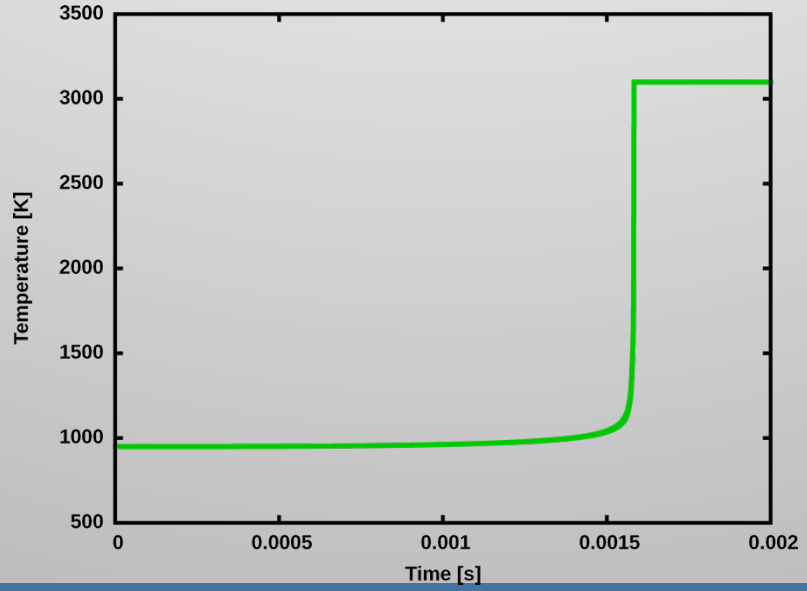
- CPU
  - LAPACK
    - dgetrf
    - dgetrs
- GPU
  - CUBLAS
    - dgetrfbatched
    - dgtribatched
    - batched matrix-vector multiplication



- CPU
  - SuperLU
    - dgetrf
    - dgetrs
- GPU
  - GLU (soon cusolverSP (7.0))
    - LU refactorization
      (SuperLU for first factor)
    - LU solve
    - Conglomerate matrix (<6.5)
    - Batched matrices (>= 6.5)
      (*2-4x faster*)

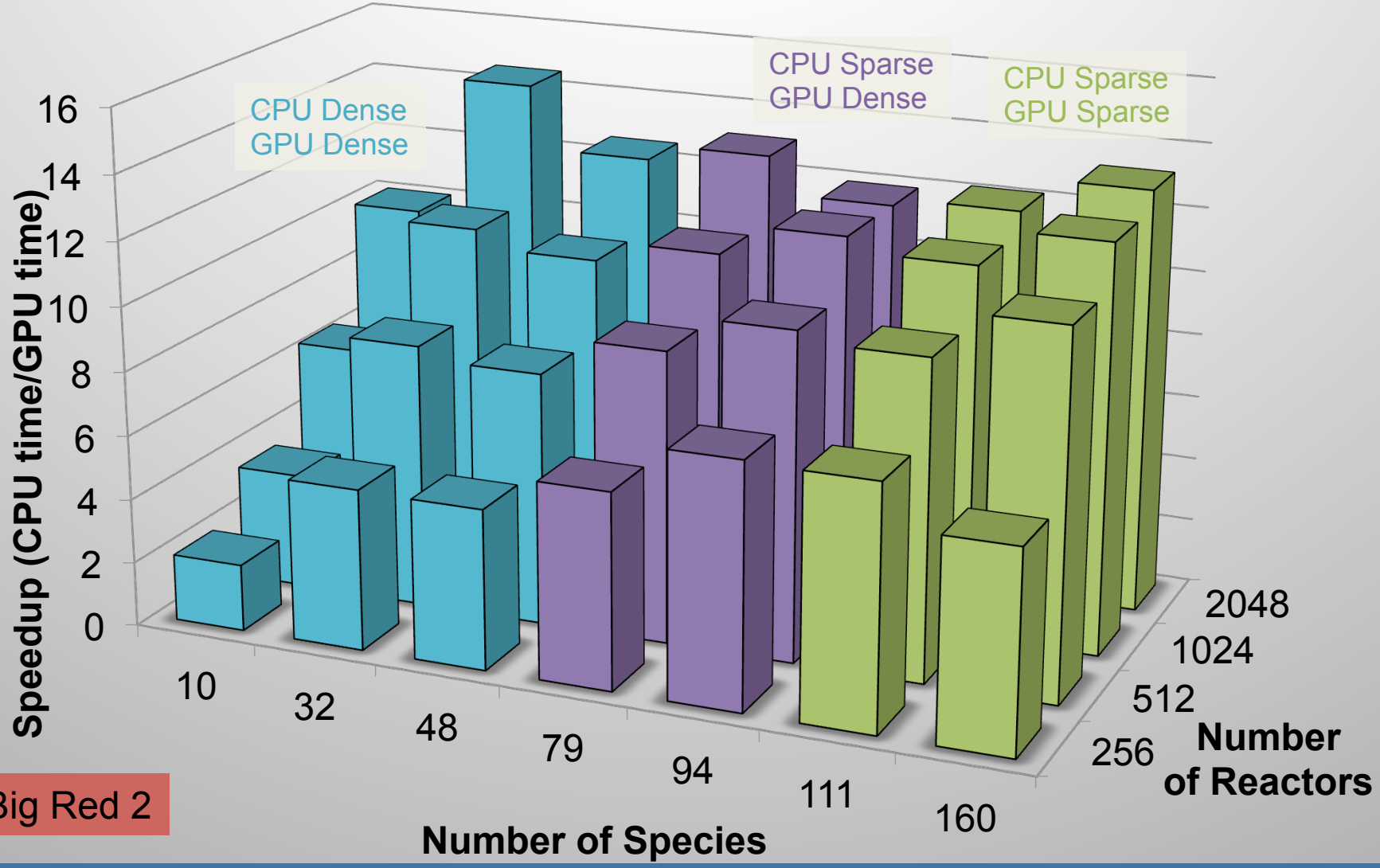
Test case for full chemistry integration

- Ignition delay time calculation (i.e. shock tube simulation):
  - 256-2048 constant volume reactor calculations
  - No coupling to CFD
  - Comparing CPU and GPU
    with both dense and sparse matrix operations



This provides a gauge of what the ideal speedup will be in CFD simulations.

0D, Uncoupled, Ideal Case: Max speedup



Big Red 2

As with dC_i/dt best speedup is for large number of reactors.

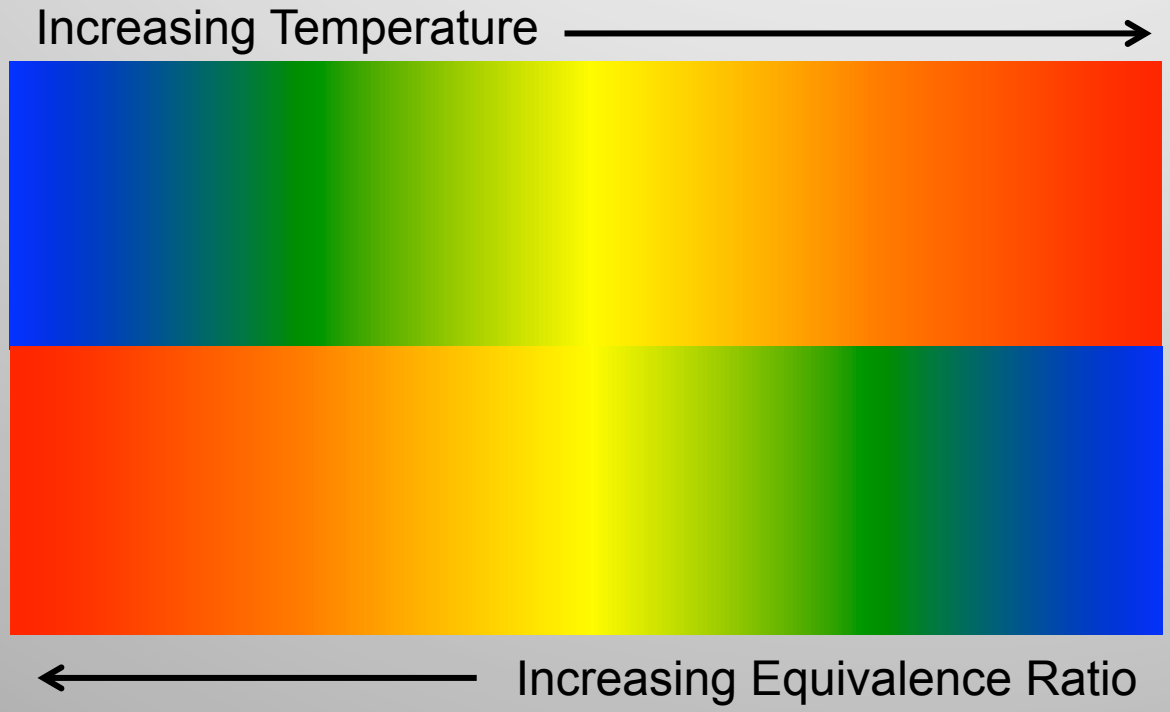
Synchronization Penalty Test Case:

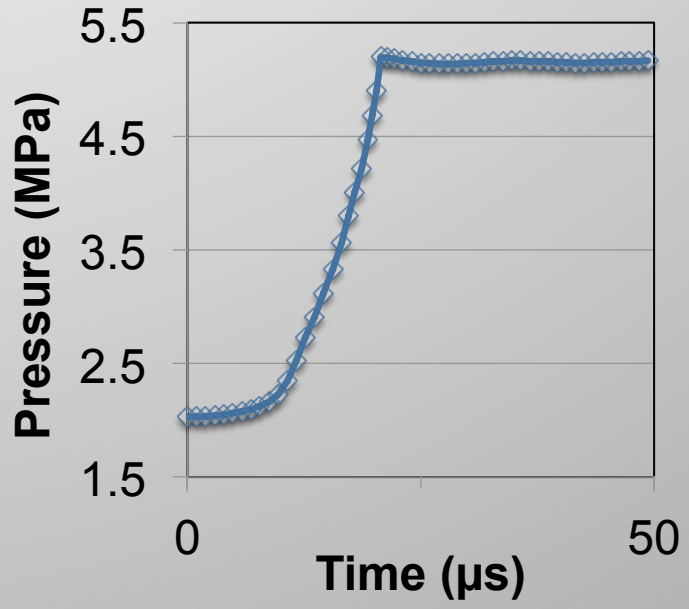
- Converge CFD
- Rectilinear volume (16x8x8 mm)
- Initial conditions:
  - Variable gradients in temperature (⬆) & phi (⬇)
  - Uniform zero velocity
  - Uniform pressure (20 bar)
- Boundary conditions:
  - No flux for all variables
- ~50 CFD steps capturing complete fuel conversion
- Every cell chemistry (2048 cells, 1 CPU core, 1 GPU device)
- 7 kinetic mechanisms from 10-160 species
- Solved with both sparse and dense matrix algorithms

Testing affect of non-identical reactors.

We compared the total chemistry cost for sequential auto-ignition in a constant volume chamber

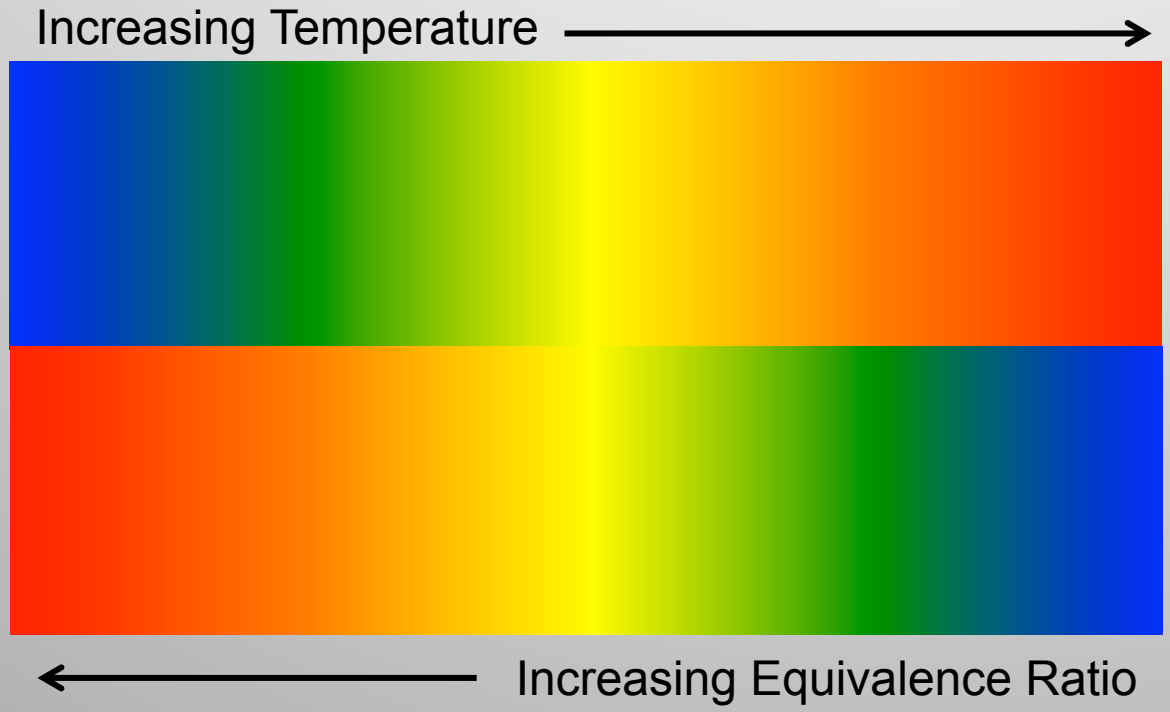
Initial Conditions:





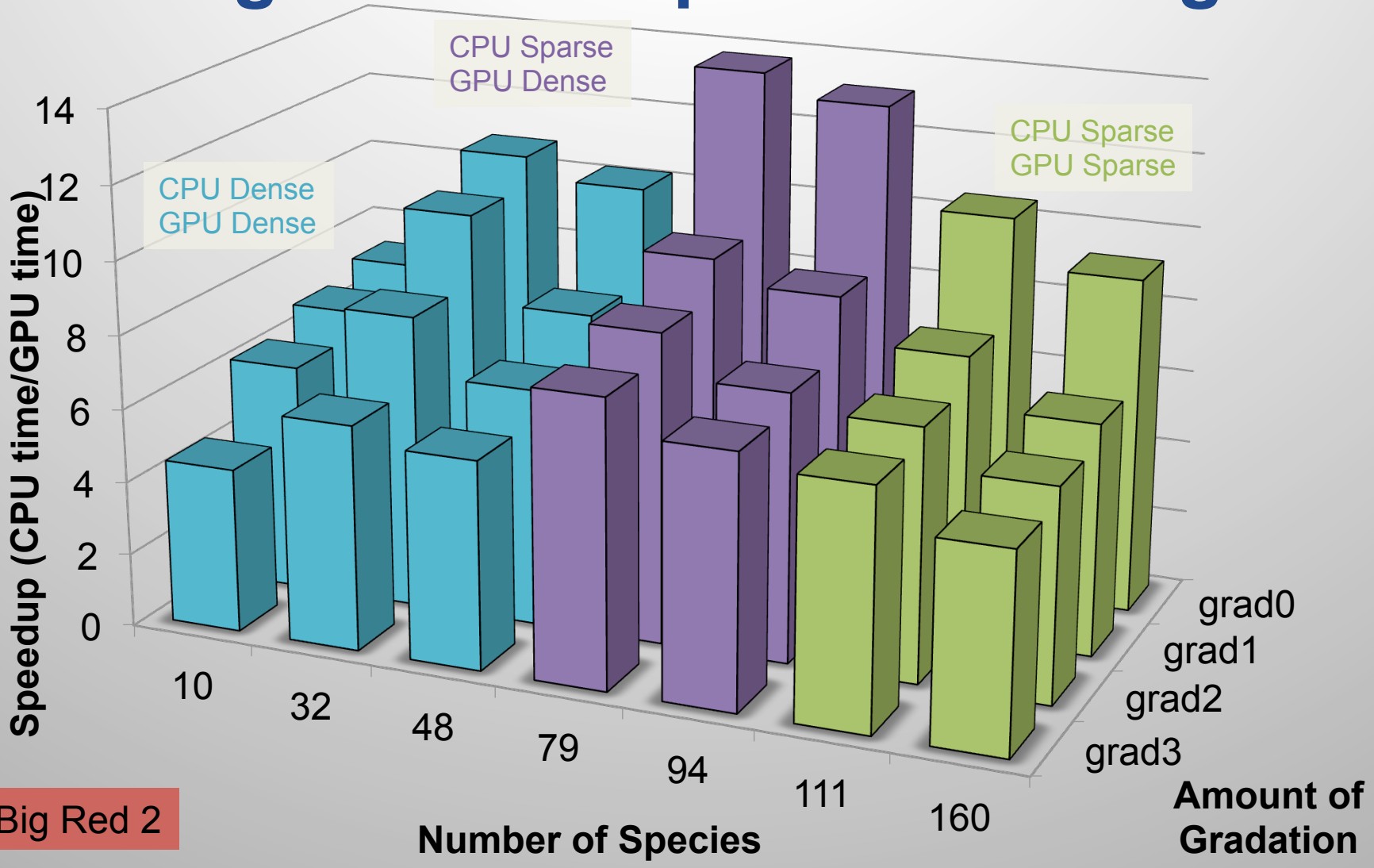
We compared the total chemistry cost for sequential auto-ignition in a constant volume chamber

Initial Conditions:



Condition	T spread	ϕ spread
Grad0	1450	1.0
Grad1	1400-1450	0.95-1.05
Grad2	1350-1450	0.90-1.10
Grad3	1250-1450	0.80-1.20

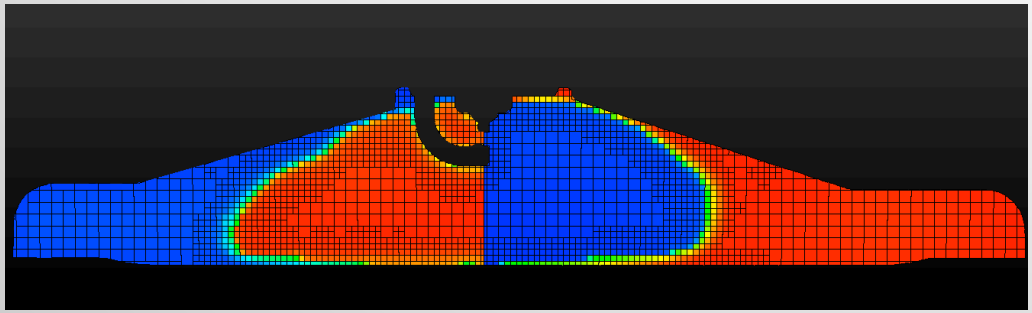
Converge GPU: Sequential Auto-ignition



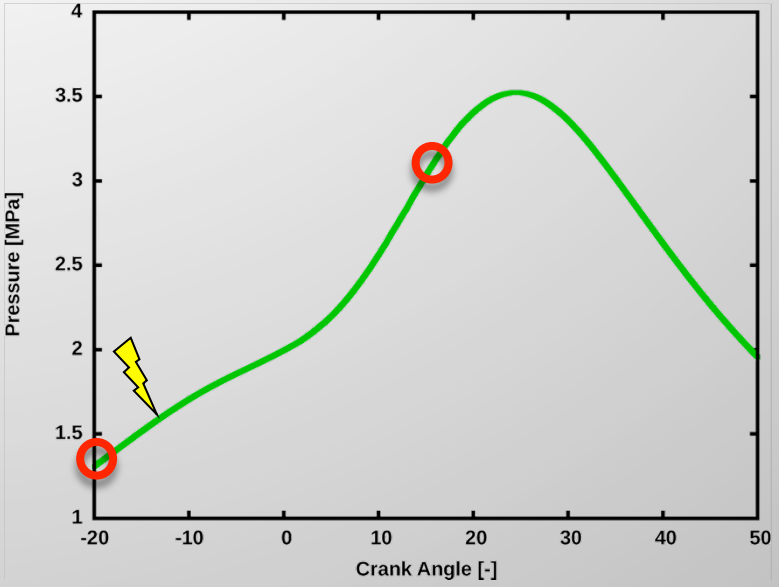
Big Red 2

Even in non-ideal case we find significant speedup.

Finally ready to run engine simulation on GPU



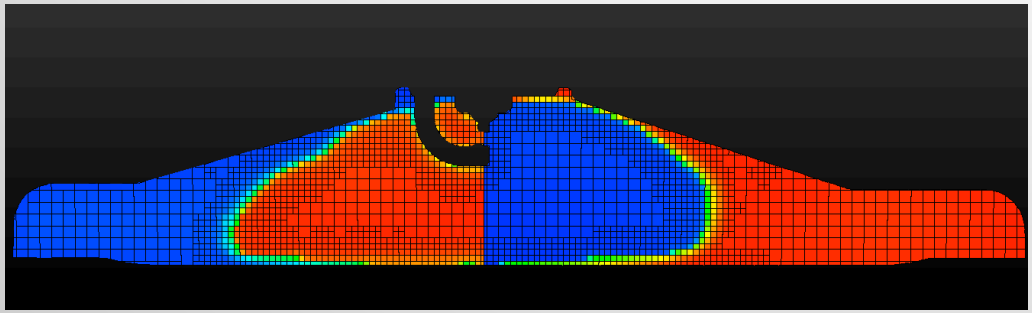
Big Red 2

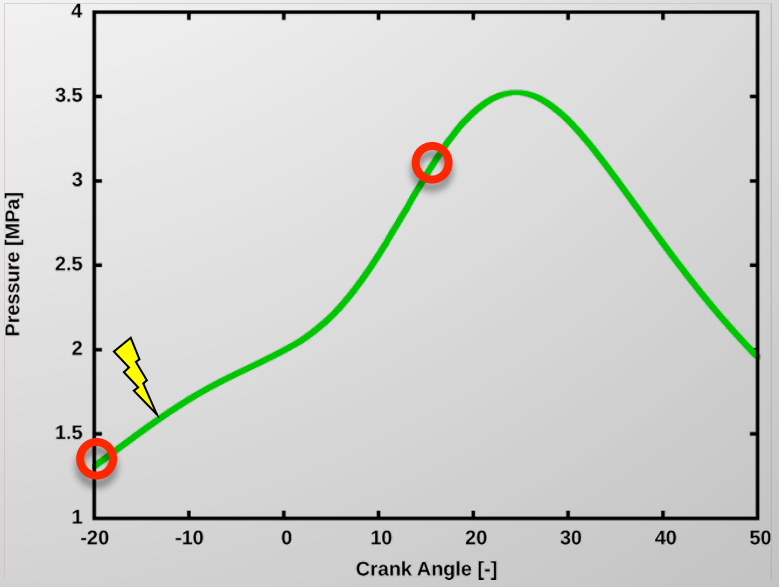


Compared cost of <u>every cell</u> chemistry from -20 to 15 CAD.
24 nodes of Big Red 2: 24 CPU **<u>cores</u>** vs. 24 GPU **<u>devices</u>**.
Should be close to worst case scenario w.r.t. synchronization penalty.

What's the speedup on a "real" problem?

Engine calculation on GPU





Big Red 2

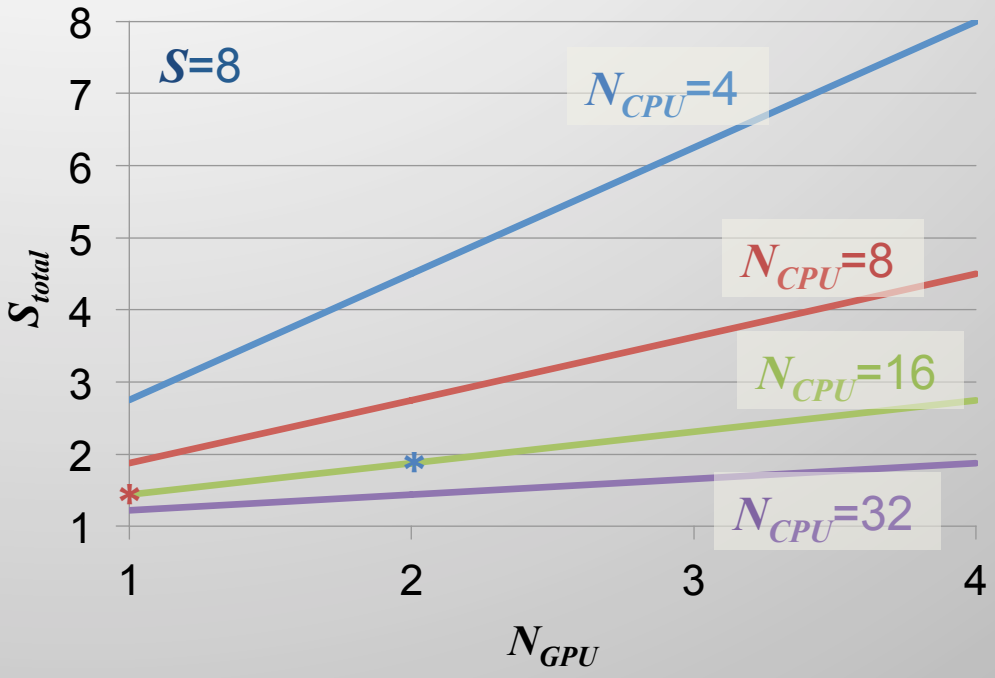
- 24 CPU cores = 53.8 hours
- 24 GPU devices = 14.5 hours
- Speedup = 53.8/14.5 = 3.7

Good speedup. With Caveats.

CPU-GPU Work-sharing

Ideal Case

$$S_{total} = \frac{\left(N_{CPU} + N_{GPU}\left(S-1\right)\right)}{N_{CPU}}$$

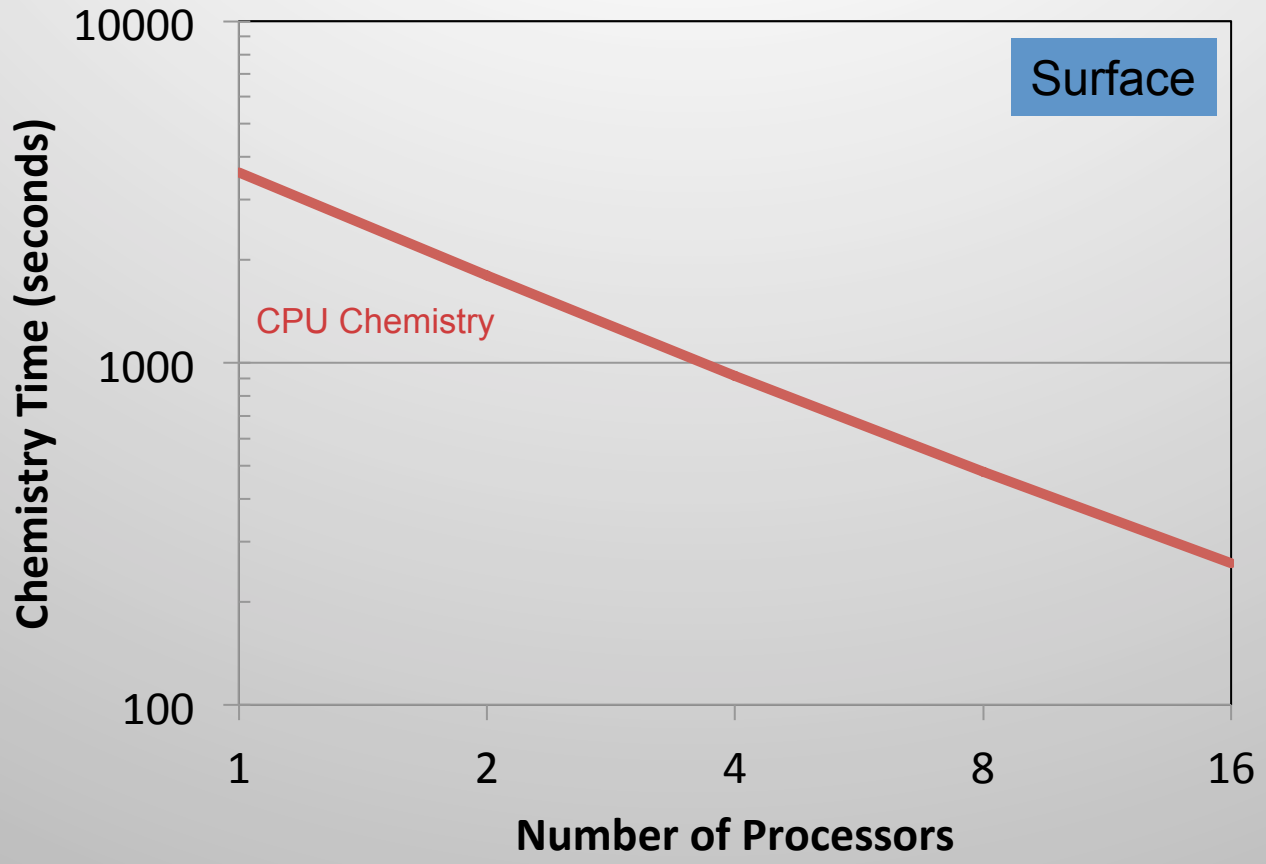


- GPU Speedup = S
- Number of CPU cores = N_{CPU}
- Number of GPU devices = N_{GPU}

* Big Red 2 (1.4375)
* Surface (1.8750)

Let's make use of the whole machine.

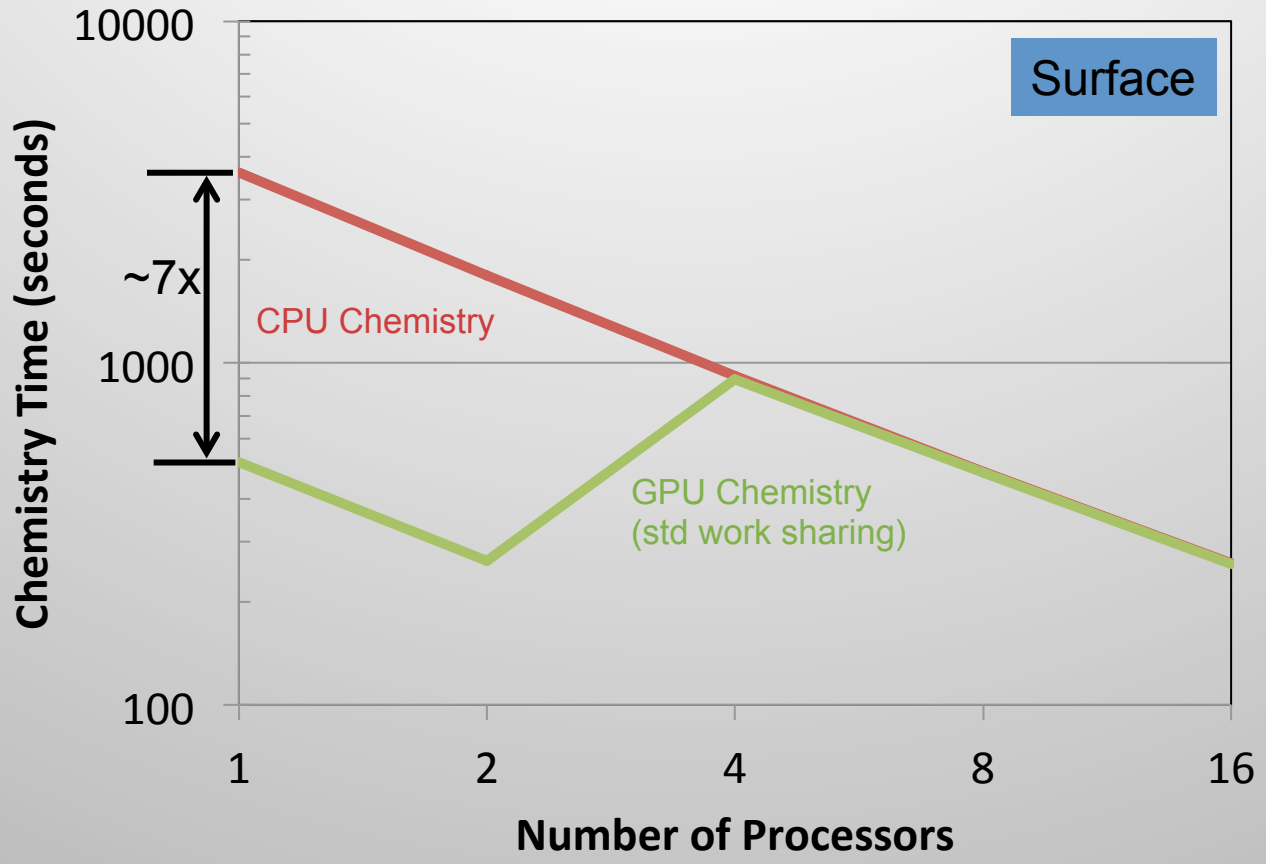
CPU-GPU Work-sharing: Strong scaling



Sequential auto-ignition case, grad0, 53 species, ~10,000 cells

Strong scaling is good for this problem on CPU.

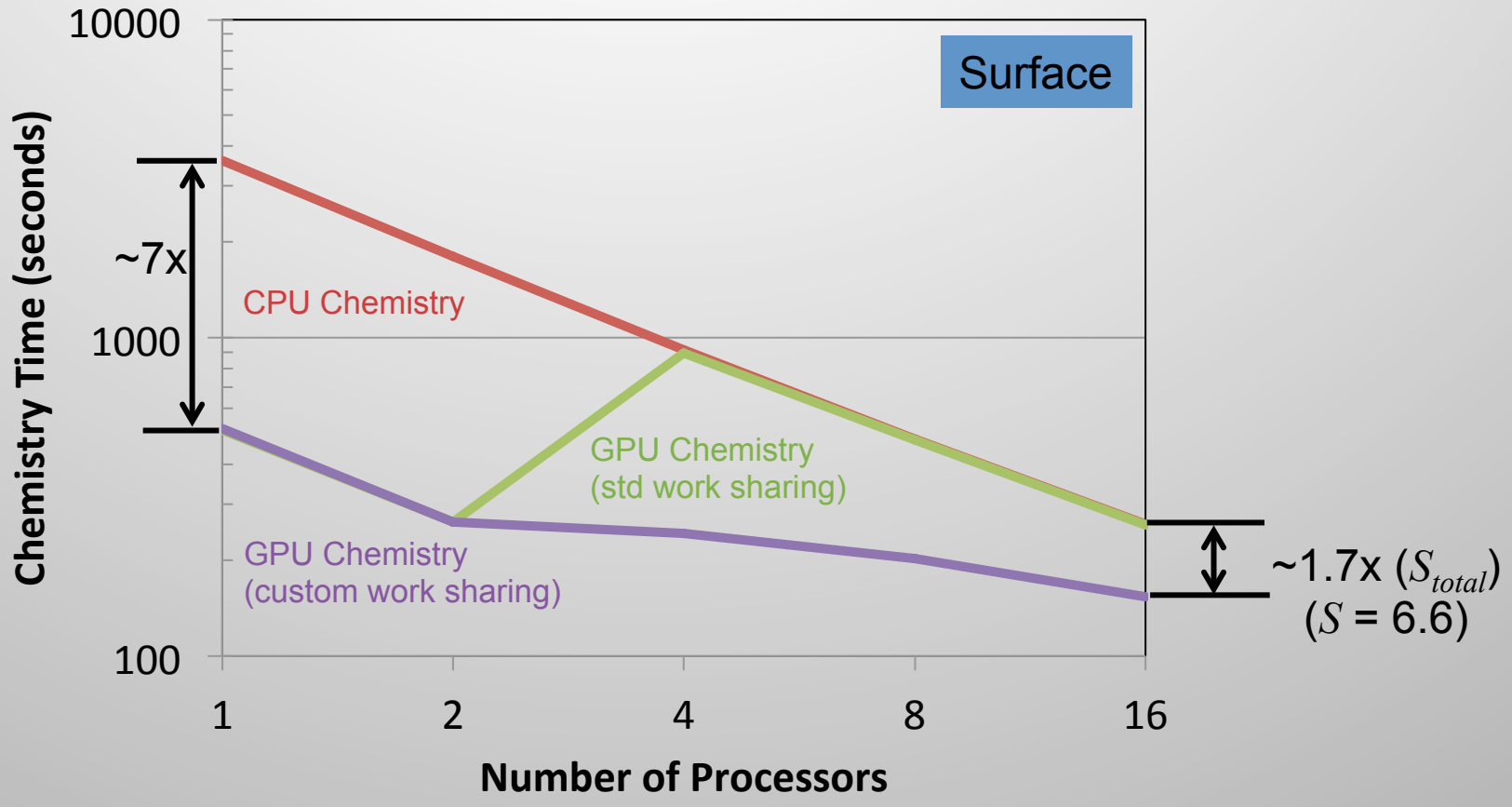
CPU-GPU Work-sharing: Strong scaling



Sequential auto-ignition case, grad0, 53 species, ~10,000 cells

Poor scaling with GPUS, if all processors get the same amount of work.

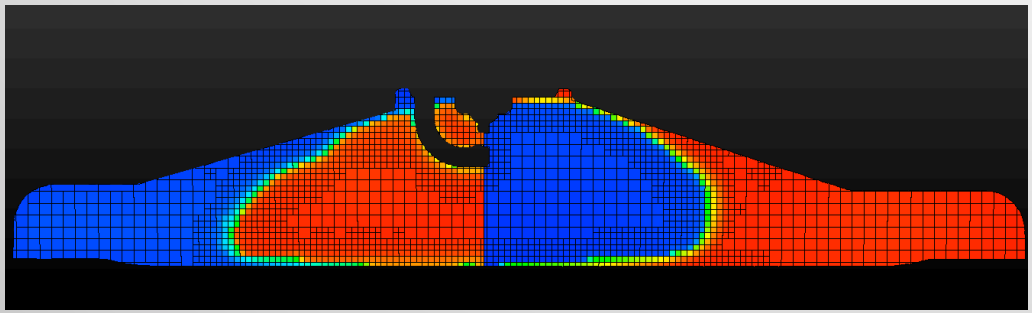
CPU-GPU Work-sharing: Strong scaling



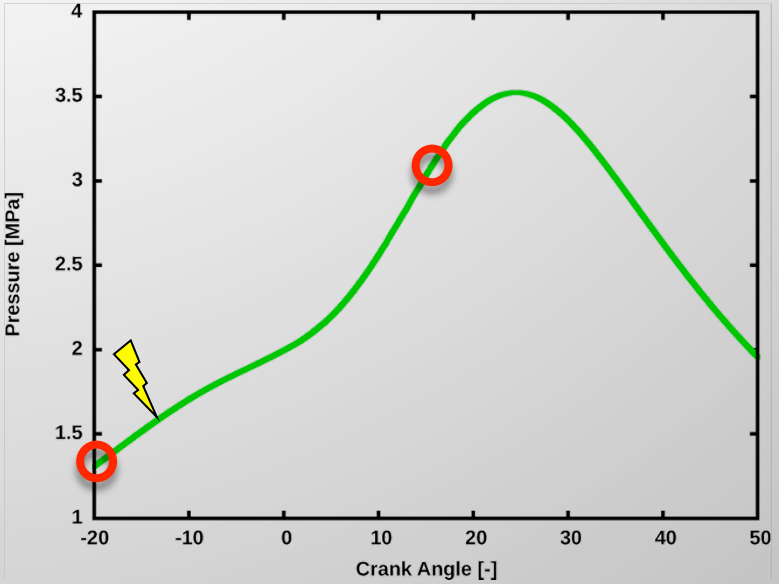
Sequential auto-ignition case, grad0, 53 species, ~10,000 cells

Better scaling if give GPU processors appropriate work load.

Proof of Principle: Engine calculation on GPU+CPU Cores



Surface

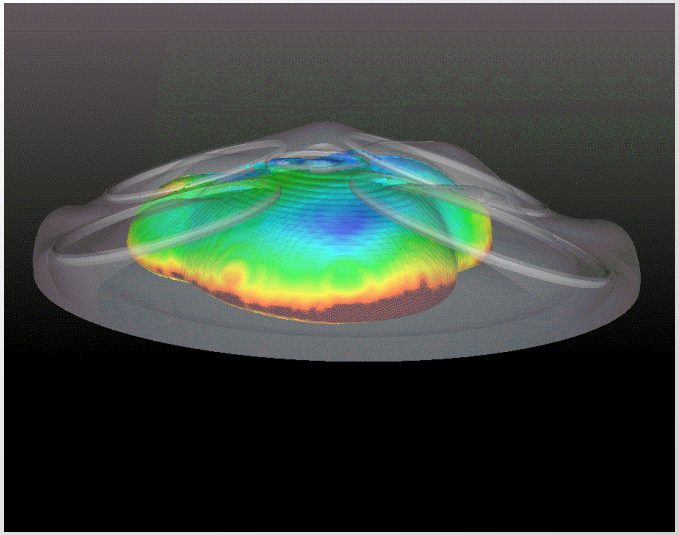


- 16 cpu cores = 21.2 hours
- 16 cpu cores + 2 GPU devices = 17.6 hours
- Speedup = 21.2/17.6 = 1.20 (S_{total}, S = 2.6)

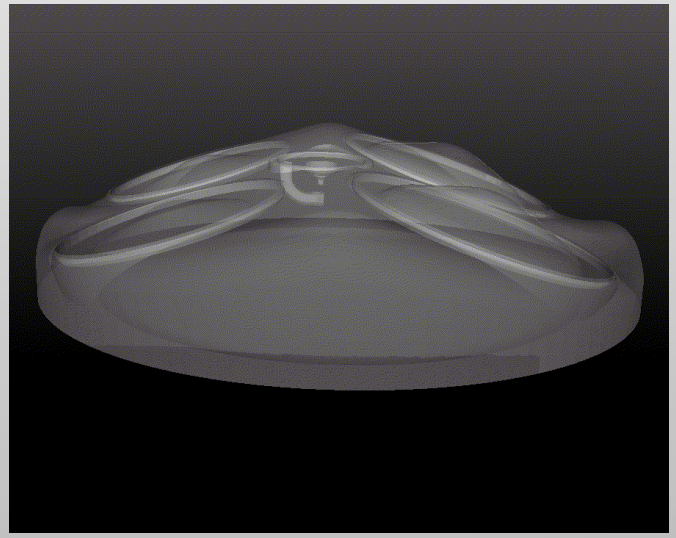
In line with expectations.

Does

plus



equal



?

Sort of.

Future directions

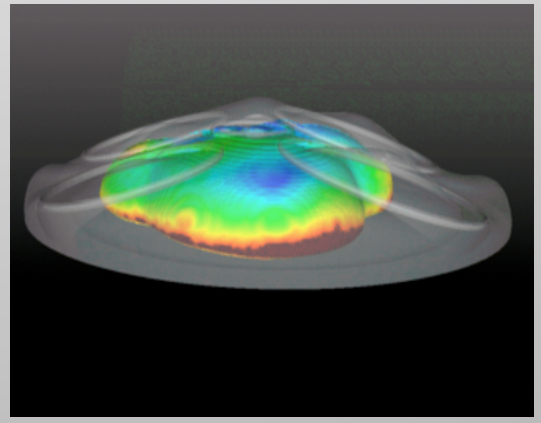
- Improve CPU/GPU parallel task management
  - Minimize synchronization penalty
  - Work stealing
- Improvements to derivative calculation
  - Custom code generation
  - Reframe parts as matrix multiplication
- Improvements to matrix calculations
  - Analytical Jacobian
  - Mixed precision calculations

Possibilities for significant further improvements.

Conclusions

- GPU chemistry for stiff integration implemented
- Implemented as Converge CFD UDF but flexible for incorporation in other CFD codes.
- Continuing development:
  - Further speedup envisioned
  - More work can improve applicability

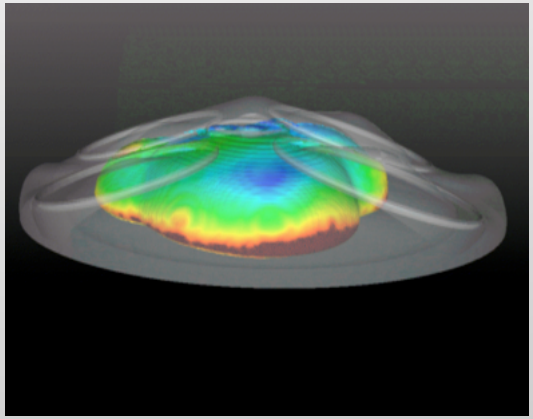
+



Thank you!

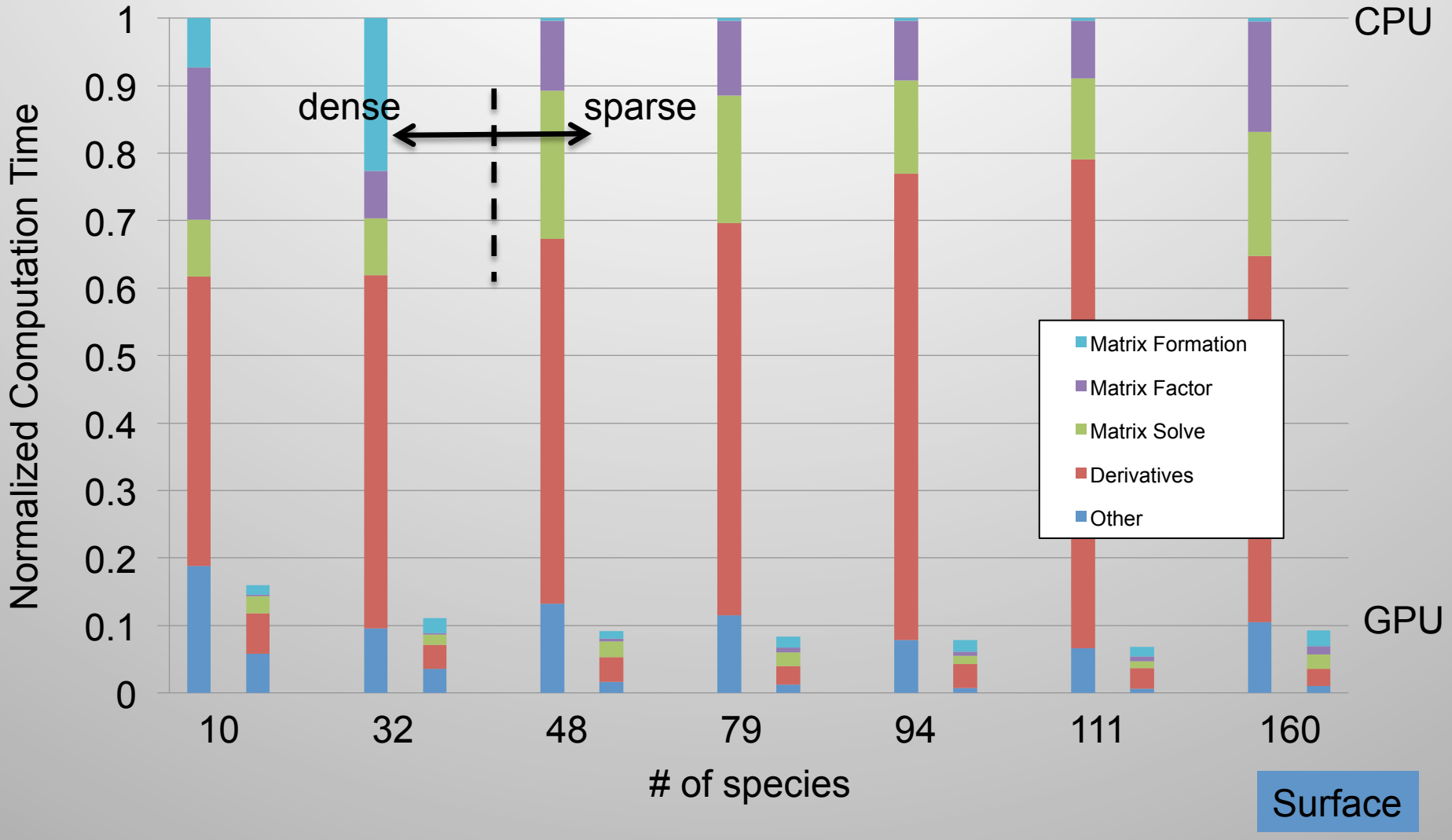
Supplemental Slides

+



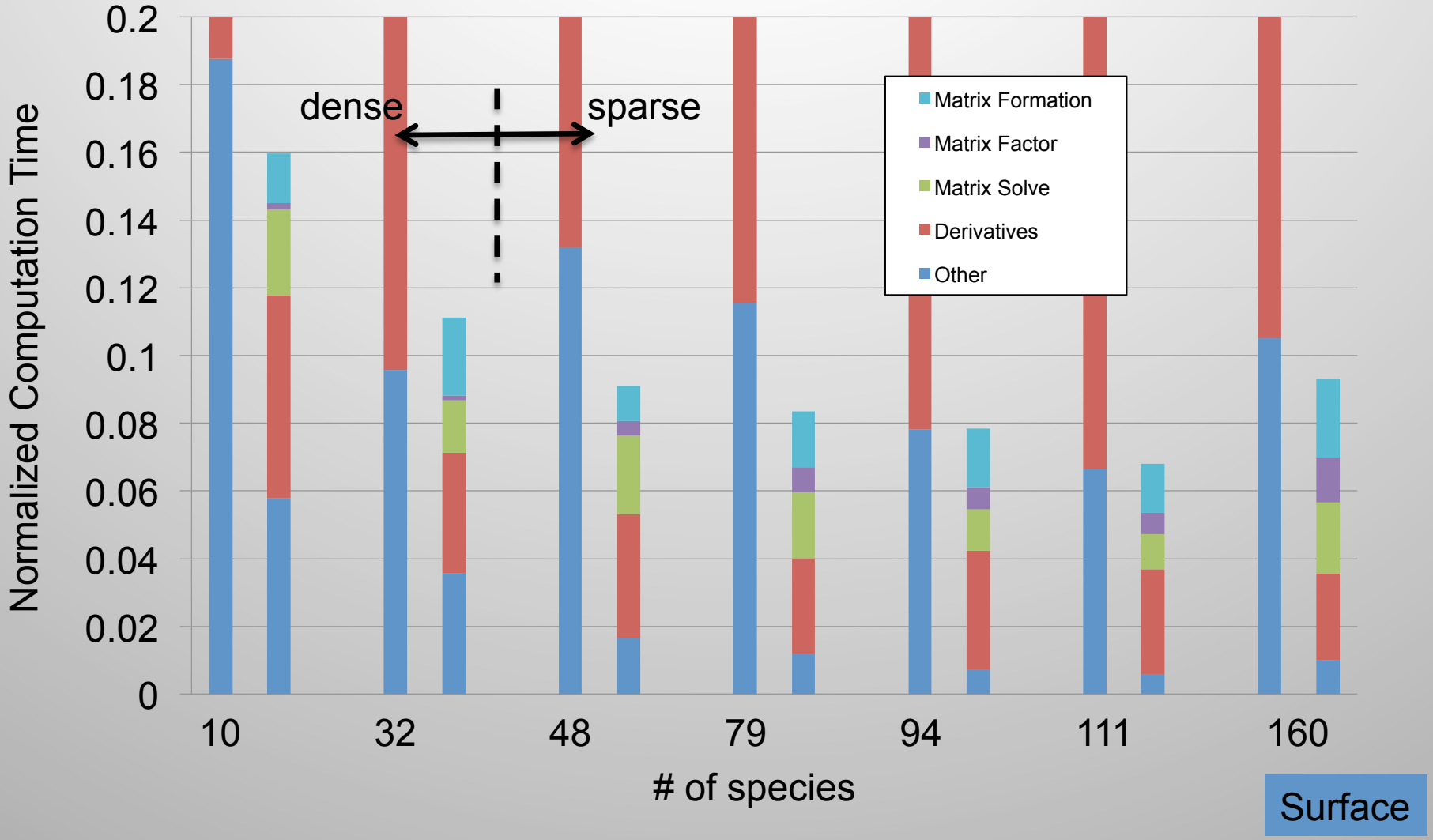
Just in case.

0D, Uncoupled, Ideal Case: Cost Breakdown



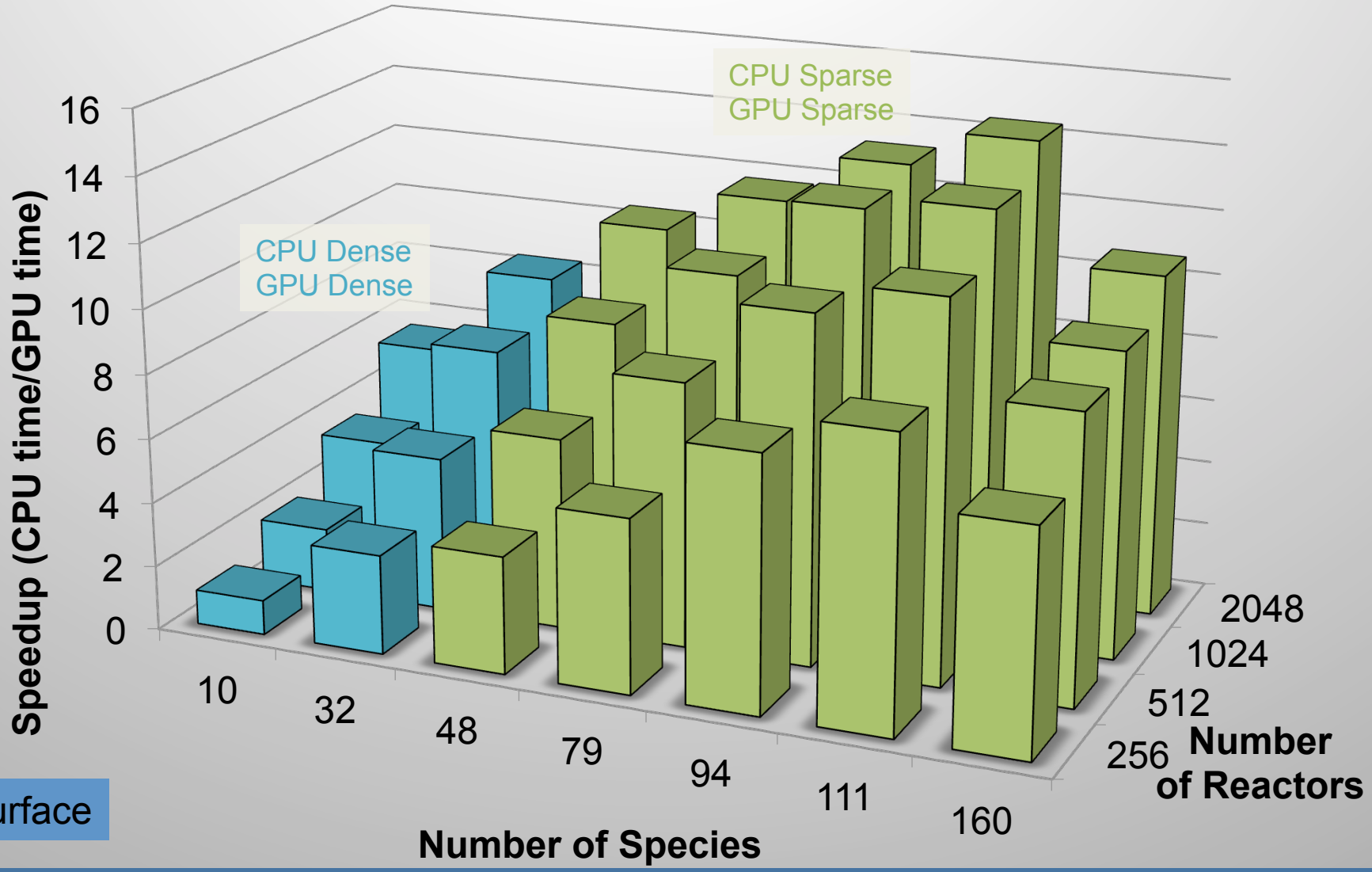
Evenly distributed costs both on CPU and GPU

0D, Uncoupled, Ideal Case: Cost Breakdown



Evenly distributed costs both on CPU and GPU

0D, Uncoupled, Ideal Case: Max speedup



As with dC_i/dt best speedup is for large number of reactors.