Faster Kinetics: Accelerate Your Finite-Rate Combustion Simulation with GPUs

Christopher P. Stone, Ph.D.
Computational Science and Engineering, LLC

Kyle Niemeyer, Ph.D.
Oregon State University

Outline

- Combustion Kinetics: Operator Splitting
- Stiff ODE Integration: Algorithms
- CUDA Implementations
- Benchmarks: VODE vs. RKx
- Performance Analysis
- Summary

Combustion Kinetics

- Must solve Navier-Stokes and species conservation equations: massive PDE system.

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \bar{\rho}\tilde{u}_i}{\partial x_i} = 0$$

$$\frac{\partial \bar{\rho}\tilde{u}_i}{\partial t} + \frac{\partial}{\partial x_j}\left[\bar{\rho}\tilde{u}_i\tilde{u}_j + \bar{p}\delta_{ij} - \bar{\tau}_{ij} + {\tau_{ij}}^{sgs}\right] = 0$$

$$\frac{\partial \bar{\rho}\tilde{E}}{\partial t} + \frac{\partial}{\partial x_i}\left[\left(\bar{\rho}\tilde{E} + \bar{p}\right)\tilde{u}_i + \bar{q} - \tilde{u}_j\bar{\tau}_{ij} + {H_i}^{sgs} + {\sigma_i}^{sgs}\right] = 0$$

$$\frac{\partial \bar{\rho}\tilde{Y}_k}{\partial t} + \frac{\partial}{\partial x_i}\left[\bar{\rho}\tilde{Y}_k\tilde{u}_i + \bar{\rho}\widetilde{Y_k}\tilde{V}_{i,k} + {\Phi_{i,k}}^{sgs} + {\Theta_{i,k}}^{sgs}\right] = \overline{\dot{\omega}_k}^{sgs}$$

- This coupled PDE system is too expensive to solve when we have finite-rate kinetics, large mechanisms, and fine meshes.
- Instead, decouple reaction terms spatially and solve them over the CFD time-step. …. ***Operator Splitting***

Combustion Kinetics

- Integrate convection-diffusion and reaction components separately with CFD time-step: still 2nd-order time accurate
- Valid when reaction time-scales are much faster than convection-diffusion time-scales.
- Transforms massive PDE into thousands or millions of ***independent*** ODE systems that can be solved by a stiff solver in parallel.

N_s+1 coupled ODE system

$$\begin{cases} \dfrac{dy_i}{dt} = \dfrac{\dot{\omega}_i}{\rho} \cdots i = 1, N_s \\ \dfrac{dT}{dt} = -\dfrac{1}{\rho c_p}\sum_{i=1}^{N_s} h_i \dot{\omega}_i \end{cases}$$

Non-linear RHS Function

$$\dot{\omega}_i = \sum_{j=1}^{N_i} \upsilon_{ij} A_j T^{\beta_j} e^{-\frac{E_j}{RT}} \left(\prod_{k=1}^{N_i} c_k{}^{\upsilon' kj} - \frac{1}{K^c{}_j} \prod_{k=1}^{N_i} c_k{}^{\upsilon'' kj} \right)$$

Combustion Kinetics

- ODEs integration still expensive due to
  1. Numerical stiffness
  2. Costly RHS reaction function – scales as ~ N_s
  3. Jacobian matrix factorization – scales as ~ $(N_s)^3$
- Targeting "modest" mechanisms: 10's < N_s < 100's
- Embarrassingly parallel on host assuming thousands of grid points per core
- But must be careful with variable workload
  1. Each ODE is unique: different initial conditions.
  2. Adaptive step-size: different number of steps per ODE.

Keys to GPU Performance

- Stream data from global memory: optimal bandwidth if threads ***within a warp*** read contiguous data from global memory
- Oversubscribe threads to hide latency: run many warps per SM to hide slow global memory
- Warp-level vector processing: optimal throughput if each thread ***within a warp*** executes the same instruction on different data (SIMD)

Must use all 3 for best performance.

SIMD processing challenging in ODE setting.

Mapping ODEs to the GPU

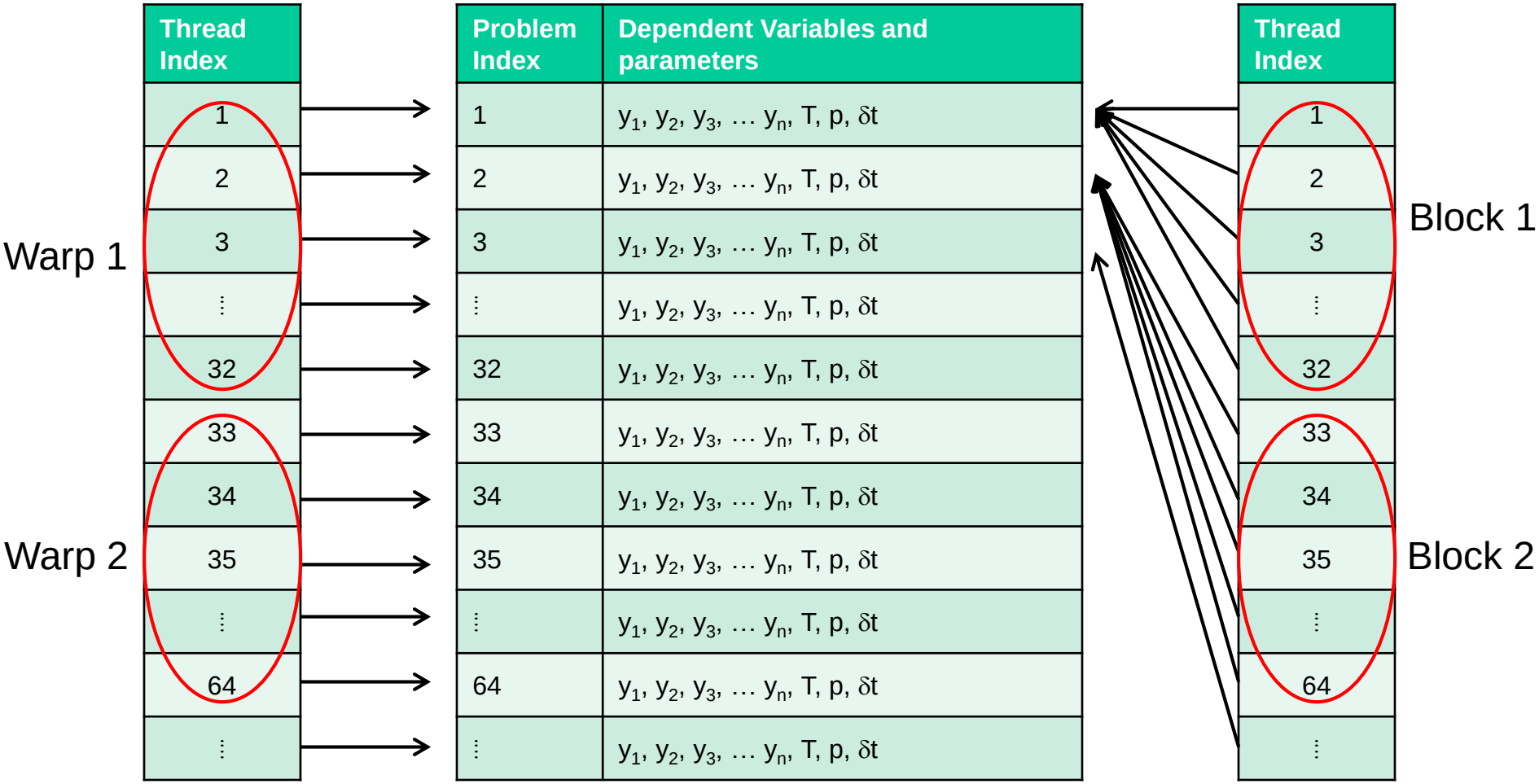
A. ODE solver logic runs on device
   1. One-thread-per-ODE: 10,000+ ODEs concurrently
   2. One-block-per-ODE: 100+ ODEs concurrently

B. ODE solver logic runs on host
   1. One-kernel-per-ODE: 1 ODE at a time
   2. Offload expensive components: RHS, Jacobian, etc.
   3. Not considered in this study: must have very large mechanisms to reach enough parallelism

Mapping ODEs to the GPU

One-thread-per-ODE

One-block-per-ODE



Mapping ODEs to the GPU

one-thread-per-ODE

1. Straightforward: replicate serial code
2. Warp-level conflict between ODEs
3. Only global memory
4. O(10k) problems for full occupancy

one-block-per-ODE

1. Complex: exploit parallelism within solver and across ODEs
2. No warp-level conflict between ODEs
3. Only shared memory: limits problem size
4. O(100) problems for full occupancy

Stiff ODE Solvers

- VODE: Variable-coefficient ODE solver
  - Backwards differentiation formula (BDF)
  - Maintained by LLNL since early 80's
  - Flavors: DVODE (Fortran); CVODE (C/C++)
  - 1-5th order implicit BDF with non-linear Newton-Raphson iteration and h/p-adaption.
- Solves generic system of ODEs with user-defined RHS function

$$\dot{y} = f(t, y);\ \ y(t_0) = y_0; y \in \mathbb{R}^N$$

VODE Stiff Algorithm

1. Estimate initial step-size: $h_0 = h_0(f(y,t_0),\varepsilon)$
2. Initialize order p = 1
3. While ($t < t_1$)
   - A. Predict y(t+h) using (p-1)-th order polynomial extrapolation
   - B. Correct y(t+h) using p-th order ($1 \leq p \leq 5$) BDF
     - i. Compute (approximate) Jacobian matrix (J) … if necessary
     - ii. Factorize iteration matrix $M = (I - \gamma J)$ … if necessary
     - iii. While (not converged): iterate Newton-Raphson solver
       - a) Compute RHS function at y^m
       - b) Solve correction $\delta y^{m+1} = M^{-1} f(y^m)$
   - C. If solution accepted: t = t + h
   - D. Adjust *h* and *p*
4. Interpolate solution backwards if overshoot ($t > t_1$)

Heuristic optimization: only execute if it will help run-time

Stiff ODE Solvers

- VODE implicit BDF algorithm highly optimized to reduce *sequential* execution time for stiff problems

… but …

- Complex implicit logic makes parallel (SIMD) execution less efficient: *Newton iterations, p-adaption, Jacobian recycling, selective factorization*

- Fixed-order, explicit schemes should have greater parallel efficiency in SIMD environments
- … but usually perform poorly on stiff problems.
- Sacrifice numerical efficiency for higher computational throughput.

RKF Algorithm

1. Estimate initial step-size: $h_0 = h_0(f(y,t_0),\varepsilon)$
2. <u>While</u> ($t < t_1$)
   A. Compute trial y(t+h) and ||err||
   B. If solution accepted: t = t + h
   C. Adjust *h*

Cash-Karp (RKCK) and Dormand-Prince (DOPRI) other RK variants similar to Fehlberg (RKF)

- Explicit 4^{th}-order with 5^{th}-order error estimation
- Adaptive step-size for error control
- 6 RHS function evaluations per step
- Minimal branching within time-loop: high SIMD efficiency
- Variable number of steps still likely between ODEs

RKC Algorithm

1. Estimate initial step-size: $h_0 = h_0(f(y,t_0),\varepsilon)$
2. <u>While</u> $(t < t_1)$
   A. Estimate spectral radius $\rightarrow$ s.
   B. Compute trial y(t+h) with s stages and ||err||
   C. If solution accepted: t = t + h
   D. Adjust h

- "Stablized" Runge-Kutta-Chebyshev (RKC) still explicit but can handle more stiffness than RKF.
- 2^{th}-order scheme with adaptive h for error control.
- Number of stages $s \geq 2$ estimated every 25 steps – or after rejection – based on spectral radius.
- Adaptive *s* introduces additional variability.

Benchmark Details

- Host: Opteron 6134 2.3 GHz … baseline DVODE is SERIAL!
- GPU: Fermi m2050

- ODE Parameters:
  - Relative tolerance = 10^{-10}
  - Absolute tolerance = 10^{-13}
  - $\delta t \sim 10^{-7}$ sec

- Simulation database constructed from stochastic Counter-Flow Linear-Eddy Model (LEM-CF) with many different fuel-oxidizer conditions:
  - 19 mil samples available
  - ~ 300 cells per simulation

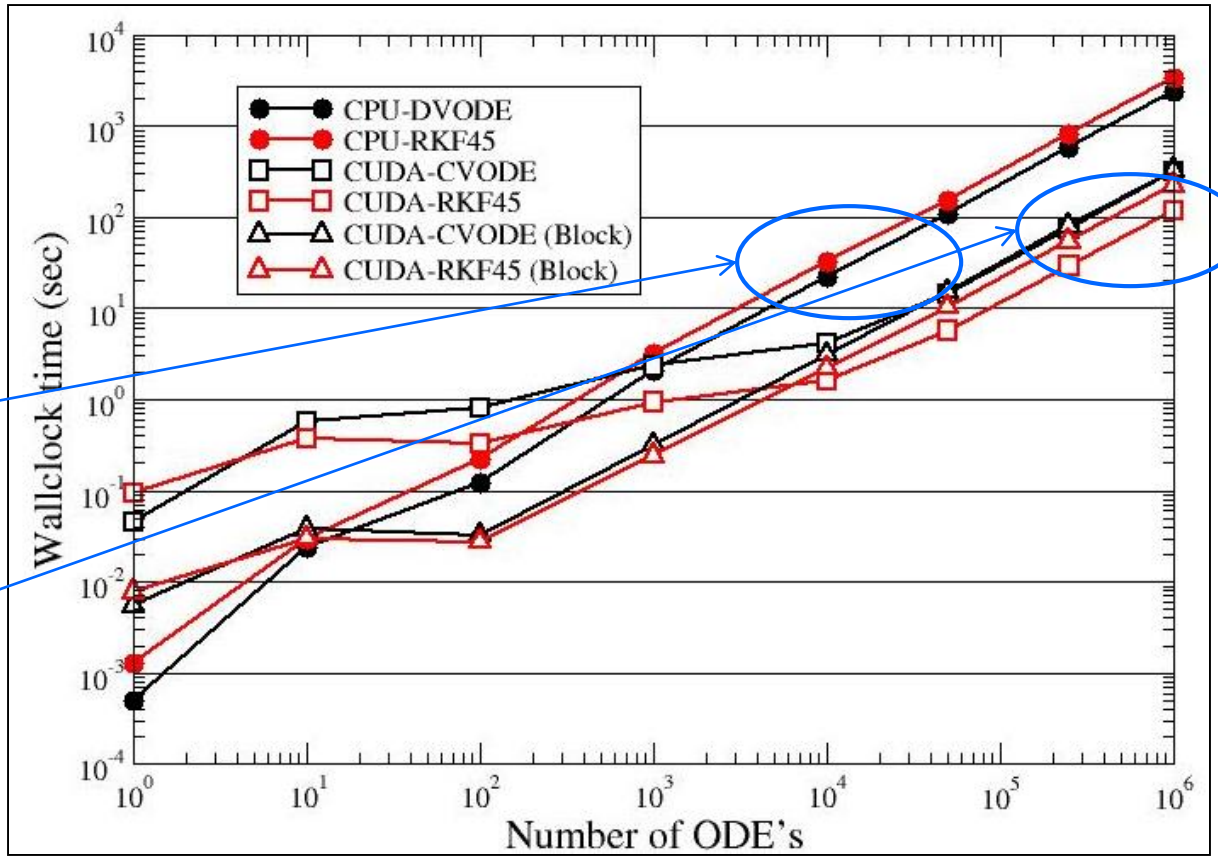
- 19 species C_2H_4 mechanism
  - 167 reversible reactions
  - 10 quasi-steady species
  - Low stiffness

RKF Solver Performance

RKF ~ 2x slower than VODE on host

RKF ~ 2-3x faster than VODE on GPU

Overhead less than 0.05%



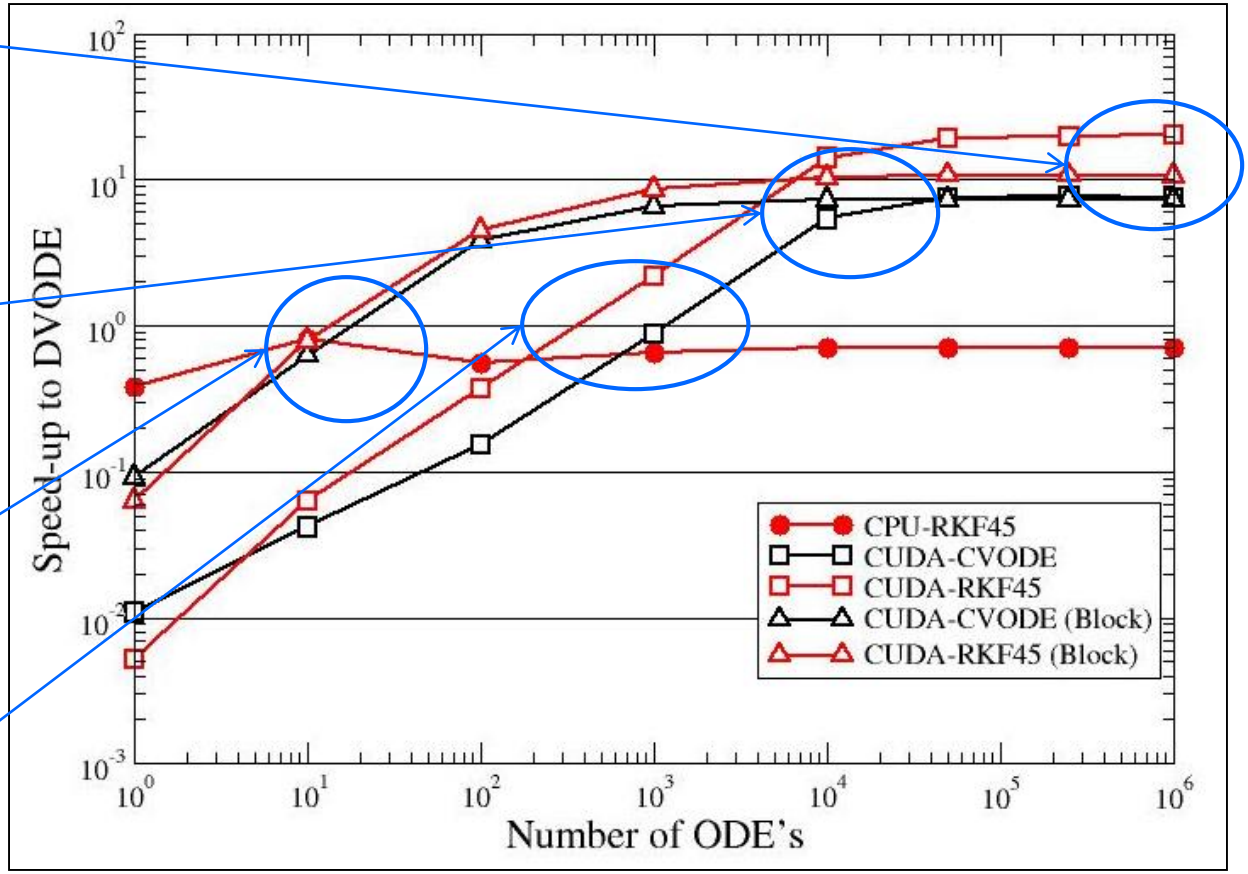
RKF Solver Performance

One-thread RKF:
20.2x max speed-up
One-block RKF:
10.7x max speed-up

One-thread VODE:
7.7x max speed-up
One-block VODE:
7.3x max speed-up

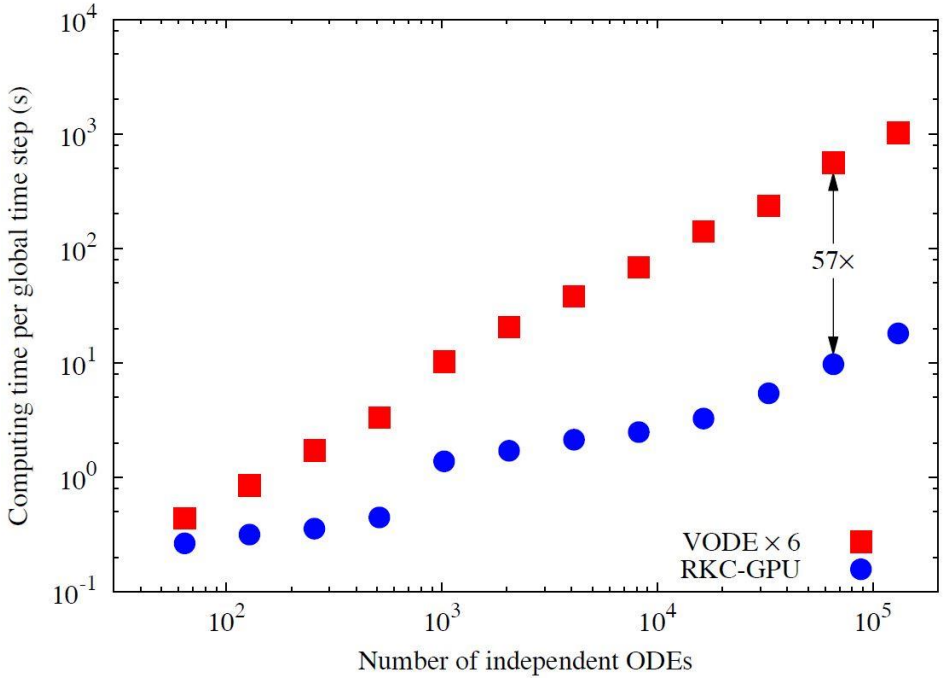
One-block breaks even at M ~ 10

One-thread breaks even at M ~ 1000



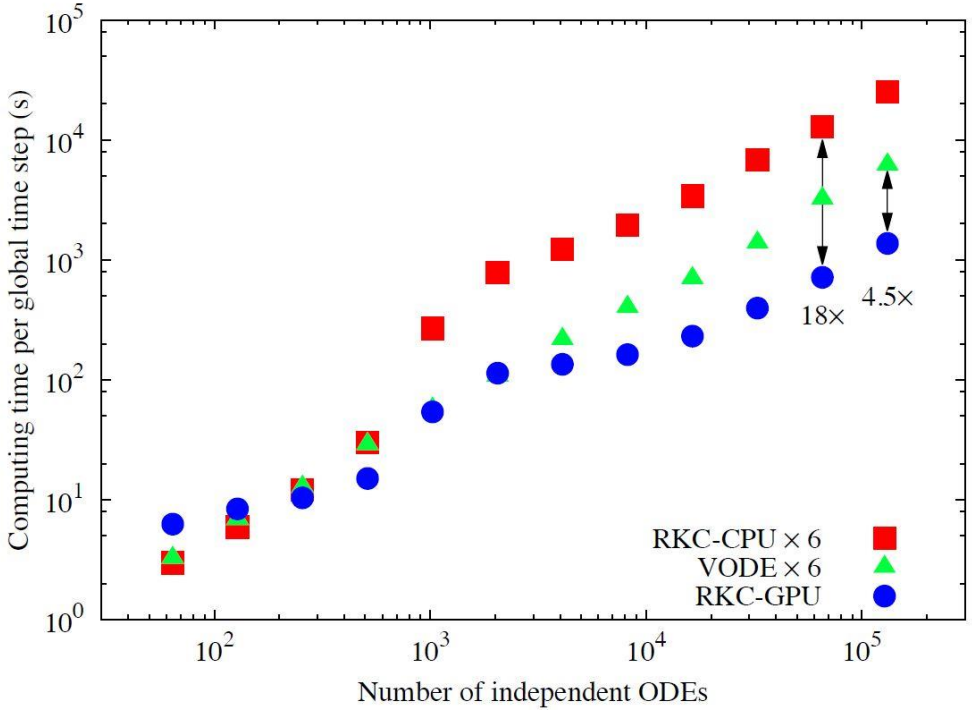
RKC Solver Performance

- RKC vs. VODE with *moderately stiff* natural gas mechanism (GRI Mech v3)
  - 53 species, 634 reactions
  - Reltol = 10^{-6}; Abstol = 10^{-10}
  - $\delta t = 10^{-6}$ sec
  - Homogeneous ignition test problem
- Host: Xeon X5650 2.67GHz
- GPU: c2075
- 57x speed-up over CPU VODE



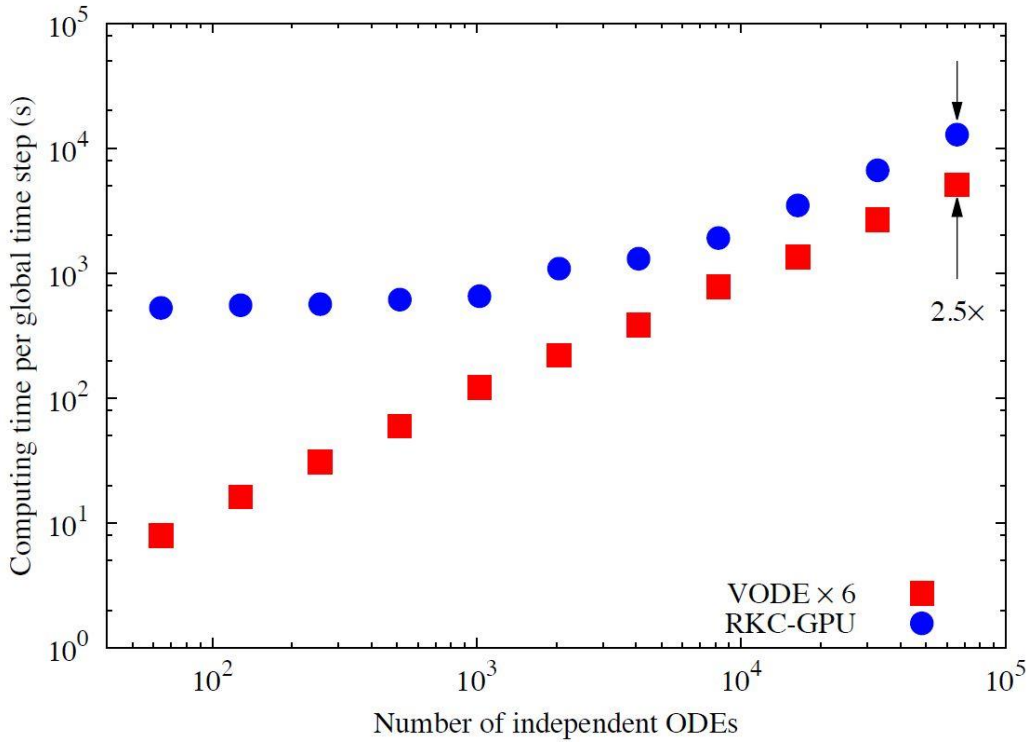
RKC Solver Performance

- C_2H_4-Air mechanism:
  - 111 species, 1566 reactions.
  - Reltol = 10^{-6}; Abstol = 10^{-10}
  - $\delta t = 10^{-6}$ sec
- 27x speed-up over CPU VODE … <u>still very effective</u>.



RKC Solver Performance

- Increase the stiffness by increasing the global time-step:
  - $\delta t = 10^{-4}$ sec
- RKC 2.5 slower than VODE for very stiff problem.



Impact of Ordering

- ODE performance is based on “linear” database
  - Neighboring grid points have similar initial conditions
  - Leads to better SIMD vectorization
- Randomize the database … amplifies the variation in the number of steps per ODE within each warp.
- RKF strongly impacted.
- VODE minimally impacted: already running poorly

	Randomization slow-down in one-thread implementation
VODE	**12%**
RKF	**71%**

Slow-down due to increased divergence.

Performance Summary

- VODE achieved ~8x speed-up on GPU with small C_2H_4 mechanism.
  - Complex algorithm has too much divergence.

- Fixed-order, explicit schemes performed better on device.
  - RKF > 20x speed-up over serial host VODE performance with 19-species ethylene.
  - RKC > 55x speed-up with modestly stiff CH_4 mechanism.
  - RKC performance diminished as stiffness is increased – still need implicit solver for very stiff problems.

ODE Solver Recipe

1. What's best on the host isn't always best on the GPU:
   a) Explicit RK beats implicit VODE on many problems if the global δt is small.
   b) Very stiff problems still need implicit solver.
2. One-thread-per-ODE mapping provides effective speed-up assuming 10k's of concurrent ODEs.
   a) Straightforward translation from single-threaded host code.
   b) One-block-per-ODE could be useful if ODEs vary widely or if there are only 100's of ODEs.
3. Be careful with ordering:
   a) Variable number of steps can severely degrade performance.
   b) Group ODEs with similar conditions in same warp.

Acknowledgments

- Funding for RKF and VODE GPU implementations provided by the DoD HPCMP's User Productivity Enhancement, Technology Transfer, and Training (PETTT) Program (Contract No: GS04T09DBC0017) under project PP-CFD-KY02-115-P3.

More Information

- Stone and Davis, “Techniques for solving stiff chemical kinetics on GPUs,” AIAA J. of Propulsion and Power, 2013.
- Niemeyer and Sung, “Accelerating moderately stiff chemical kinetics in reactive-flow simulations using GPUs,” J. Computational Physics, vol. 256, pages 854-871, 2014.

Comments?