Accelerating reacting flow simulations via vectorized chemical kinetic evaluation and integration

Nicholas Curtis

September 30, 2018

Introduction

As the combustion community has recognized the importance of detailed chemical kinetics for predictive reactive-flow simulations, chemical kinetic models have grown in size and complexity to describe current and next-generation fuels relevant to transportation and power generation. Several studies [1, 2, 3] have demonstrated that the chemical kinetic integration of even modestly sized chemical kinetic models can incur severe computation overhead in realistic reactive-flow simulations. Many techniques to reduce the cost of utilizing detailed chemical models in reactive-flow simulations exist [4]; in particular, a carefully derived analytical formulation of the Jacobian matrix can greatly increase sparsity [5] and drop the cost of Jacobian evaluation to linearly depend on the number of species in the model [4]. In addition, studies have shown [S1, S2][6] that Single-Instruction, Multiple-Data (SIMD) and the related Single-Instruction, Multiple-Thread (SIMT) processors can accelerate chemical kinetic simulations through vectorized evaluation.

This work details efforts to develop and validate open-source algorithms to accelerate integration of stiff chemical kinetic ordinary differential equations (ODEs) via vectorized execution on SIMD and SIMT devices. This includes codes to evaluate the chemical kinetic source terms and a sparse, analytical Jacobian, as well as ODE integration algorithms for the CPU, GPU and other accelerators. The developed algorithms will be applied to a detailed large eddy simulation of a bluff-body stabilized turbulent reacting flame, in order to demonstrate accuracy and computational efficiency.

1 An investigation of GPU-based stiff chemical kinetics integration methods

A fifth-order implicit Runge–Kutta method and two fourth-order exponential integration methods equipped with Krylov subspace approximations were implemented for the GPU and paired with the analytical chemical kinetic Jacobian software `pyJac`. The performance of each algorithm was evaluated by integrating thermochemical state data sampled from stochastic partially stirred reactor

simulations and compared with the commonly used CPU-based implicit integrator `CVODE`. We estimated that the implicit Runge–Kutta method running on a single Tesla C2075 GPU is equivalent to `CVODE` running on 12–38 Intel Xeon E5-4640 v2 CPU cores (Table 1) for integration of a single global integration time step of 10^{-6} s with hydrogen and methane kinetic models. In the stiffest case studied—the methane model with a global integration time step of 10^{-4} s—thread divergence and higher memory traffic significantly decreased GPU performance to the equivalent of `CVODE` running on approximately three CPU cores. The exponential integration algorithms performed more slowly than the implicit integrators on both the CPU and GPU. Thread divergence and memory traffic were identified as the main limiters of GPU integrator performance, and techniques to mitigate these issues were discussed. Use of a finite-difference Jacobian on the GPU—in place of the analytical Jacobian provided by `pyJac`—greatly decreased integrator performance (Figure 1) due to thread divergence, resulting in maximum slowdowns of 7.11–240.96 ×; in comparison, the corresponding slowdowns on the CPU were just 1.39–2.61 ×, underscoring the importance of use of an analytical Jacobian for efficient GPU integration. Finally, future research directions for working towards enabling realistic chemistry in reactive-flow simulations via GPU/SIMT accelerated stiff chemical kinetics integration were identified.

Δt	# equivalent CPU cores	
	Hydrogen	GRI-Mech 3.0
10^{-6} s	38	12
10^{-4} s	15	3

Table 1: Approximate number of CPU cores running `CVODES` required to match performance of implicit Runge–Kutta algorithm on a single GPU for various models and integration time-step sizes.

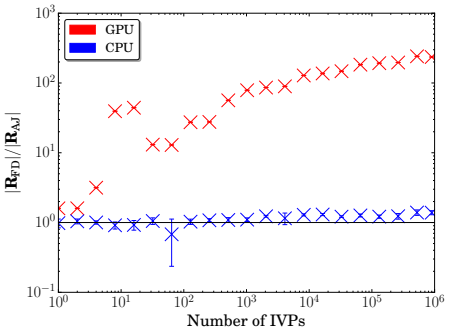
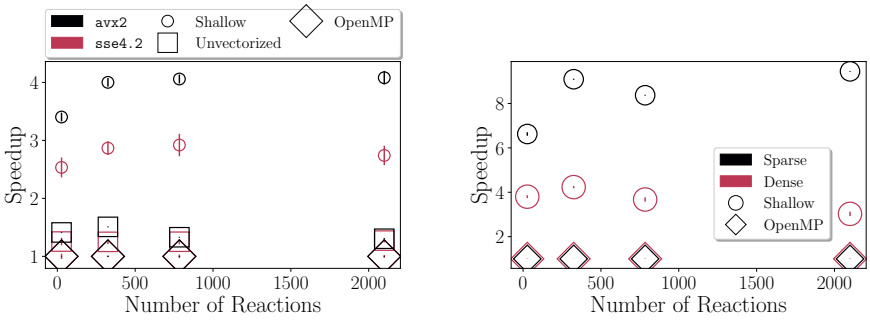


Figure 1: Ratio of the average finite-difference Jacobian based integrator runtime $|\mathbf{R}_{\mathrm{FD}}|$ to that of the analytical Jacobian runtime $|\mathbf{R}_{\mathrm{AJ}}|$ for the implicit Runge–Kutta (CPU/GPU) solvers using the hydrogen model with a global integration step-size of $\Delta t = 10^{-4}$ sec.

2 Using SIMD and SIMT vectorization to evaluate sparse chemical kinetic Jacobian matrices and thermochemical source terms

Accurately predicting key combustion phenomena in reactive-flow simulations, e.g., lean blow-out, extinction/ignition limits and pollutant formation, necessitates the use of detailed chemical kinetics. The large size and high lev-

els of numerical stiffness typically present in chemical kinetic models relevant to transportation/power-generation applications make the efficient evaluation/factorization of the chemical kinetic Jacobian and thermochemical source-terms critical to the performance of reactive-flow codes. Here we investigate the performance of vectorized evaluation of constant-pressure/volume thermochemical source-term and sparse/dense chemical kinetic Jacobians using single-instruction, multiple-data (SIMD) and single-instruction, multiple thread (SIMT) paradigms. These are implemented in pyJac, an open-source, reproducible code generation platform. Selected chemical kinetic models covering the range of sizes typically used in reactive-flow simulations were used for demonstration. A new formulation of the chemical kinetic governing equations was derived and verified, resulting in Jacobian sparsities of 28.6–92.0 % for the tested models. Speedups of 3.40–4.08 × were found for shallow-vectorized OpenCL source-rate evaluation compared with a parallel OpenMP code (Figure 2a) on an `avx2` central processing unit (CPU), increasing to 6.63–9.44 × and 3.03–4.23 × for sparse and dense chemical kinetic Jacobian evaluation (Figure 2b), respectively. Furthermore, the effect of data-ordering was investigated and a storage pattern specifically formulated for vectorized evaluation was proposed; as well, the effect of the constant pressure/volume assumptions and varying vector widths were studied on source-term evaluation performance. Speedups reached up to 17.60 × and 45.13 × for dense and sparse evaluation on the GPU, and up to 55.11 × and 245.63 × on the CPU over a first-order finite-difference Jacobian approach. Further, dense Jacobian evaluation was up to 19.56 × and 2.84 × times faster than a previous version of `pyJac` on a CPU and GPU, respectively. Finally, future directions for vectorized chemical kinetic evaluation and sparse linear-algebra techniques were discussed.



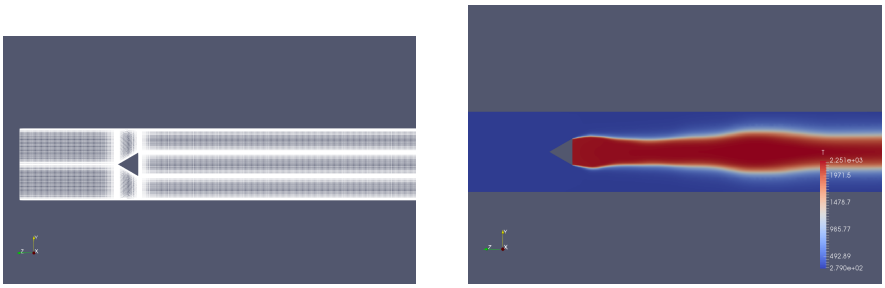
(a) Speedup of vectorized source-term evaluation over simple parallel evaluation on two different CPUs.

(b) Speedup of sparse and dense chemical kinetic Jacobian evaluation over simple parallel evaluation on a CPU with `avx2` instructions.

Figure 2: Acceleration of chemical kinetic source terms and sparse/dense analytical Jacobian on the CPU using `pyJac-V2`

3 Accelerating large-eddy simulations with SIMD-vectorized chemical kinetic integration

The Volvo bluff-body premixed turbulent reacting flame [7, 8] was modeled using the OpenFOAM computational fluid dynamics library. A large eddy simulation was conducted for a non-reacting case, with a 2 mm hexagonal mesh and 0.3 mm wall-normal distance (Figure 3a), resulting in roughly 2.37×10^6 total cells. The non-reacting velocity profiles were validated against experimentally measured mean and fluctuating velocities [7, 8]. A reacting stoichiometic methane case utilizing the GRI-Mech 3.0 model (Figure 3b) was then run on 96 cores of Intel® Xeon® E5-2690 v3 Processors (four nodes total) with the built-in fourth-order linearly-implicit Rosenbrock integration algorithm present in OpenFOAM (which includes a semi-analytical chemical kinetic Jacobian) to obtain a performance benchmark. Finally, an open-source, vectorized ODE integration algorithm for the CPU (currently under development)—consisting of a similar linearly-implicit method implemented in OpenCL [6], chemical kinetic source terms, and analytical Jacobian [S3]—will be applied to the problem. The developed algorithm will be compared to the built-in OpenFOAM solver to determine the accuracy, effective speedup and resulting impact on chemistry load-balancing. Future directions and improvements, including sparse linear-algebra and load-balancing algorithms suited for the vectorized solver will be detailed.



(a) 2 mm mesh used in bluff-body stabilized flame simulation.

(b) Snapshot of temperature field in reacting flow case using GRI-Mech 3.0 and stock OpenFOAM solvers.

Figure 3: The bluff-body stabilized premixed turbulent flame simulation.

Appendix A `pyJac`: analytical Jacobian generator for chemical kinetics

Accurate simulations of combustion phenomena require the use of detailed chemical kinetics in order to capture limit phenomena such as ignition and extinction as well as predict pollutant formation. However, the chemical kinetic models for hydrocarbon fuels of practical interest typically have large numbers of species and reactions and exhibit high levels of mathematical stiffness in the governing differential equations, particularly for larger fuel molecules. In order to integrate the stiff equations governing chemical kinetics, generally reactive-flow

simulations rely on implicit algorithms that require frequent Jacobian matrix evaluations. Some in situ and a posteriori computational diagnostics methods also require accurate Jacobian matrices, including computational singular perturbation and chemical explosive mode analysis. Typically, finite differences numerically approximate these, but for larger chemical kinetic models this poses significant computational demands since the number of chemical source term evaluations scales with the square of species count. Furthermore, existing analytical Jacobian tools do not optimize evaluations or support emerging SIMD processors such as GPUs. Here we introduce `pyJac`, a Python-based open-source program that generates analytical Jacobian matrices for use in chemical kinetics modeling and analysis. In addition to producing the necessary customized source code for evaluating reaction rates (including all modern reaction rate formulations), the chemical source terms, and the Jacobian matrix, `pyJac` uses an optimized evaluation order to minimize computational and memory operations. As a demonstration, we first establish the correctness of the Jacobian matrices for kinetic models of hydrogen, methane, ethylene, and isopentanol oxidation (number of species ranging 13–360) by showing agreement within 0.001 % of matrices obtained via automatic differentiation. We then demonstrate the performance achievable on CPUs and GPUs using `pyJac` via matrix evaluation timing comparisons; the routines produced by `pyJac` outperformed first-order finite differences by 3–7.5 times and the existing analytical Jacobian software `TChem` by 1.1–2.2 times on a single-threaded basis. It is noted that `TChem` is not thread-safe, while `pyJac` is easily parallelized, and hence can greatly outperform `TChem` on multicore CPUs. The Jacobian matrix generator we describe here will be useful for reducing the cost of integrating chemical source terms with implicit algorithms in particular and algorithms that require an accurate Jacobian matrix in general. Furthermore, the open-source release of the program and Python-based implementation will enable wide adoption.

Selected Bibliography

[S1] Nicholas J Curtis, Kyle E Niemeyer, and Chih-Jen Sung. An investigation of GPU-based stiff chemical kinetics integration methods. *Combustion and Flame*, 179:312–324, 2017.

[S2] Kyle E Niemeyer, N J Curtis, and C J Sung. `pyJac`: analytical Jacobian generator for chemical kinetics. *Comput. Phys. Comm.*, 215:188–203, Feb 2017.

[S3] Nicholas J. Curtis, Kyle E. Niemeyer, and Chih-Jen Sung. Using SIMD and SIMT vectorization to evaluate sparse chemical kinetic Jacobian matrices and thermochemical source terms. *Combustion and Flame*, 198:186 – 204, 2018.

References

[1] Chen Huang, Mingfa Yao, Xingcai Lu, and Zhen Huang. Study of dimethyl ether homogeneous charge compression ignition combustion process using a

multi-dimensional computational fluid dynamics model. *Int. J. Therm. Sci.*, 48(9):1814 – 1822, 2009.

[2] Francesco Bottone, Andreas Kronenburg, David Gosman, and Andrew Marquis. The numerical simulation of diesel spray combustion with LES-CMC. *Flow, Turbul. Combust.*, 89(4):651–673, 2012.

[3] Ahmed Abdul Moiz, Muhsin M Ameen, Seong-Young Lee, and Sibendu Som. Study of soot production for double injections of n-dodecane in CI engine-like conditions. *Combust. Flame*, 173:123 – 131, 2016.

[4] Tianfeng Lu and Chung K. Law. Toward accommodating realistic fuel chemistry in large-scale computations. *Prog. Energy Combust. Sci.*, 35(2):192 – 215, 2009.

[5] Douglas A Schwer, John E Tolsma, William H Green, and Paul I Barton. On upgrading the numerics in combustion chemistry codes. *Combust. Flame*, 128(3):270–291, 2002.

[6] Christopher P. Stone, Andrew T. Alferman, and Kyle E. Niemeyer. Accelerating finite-rate chemical kinetics with coprocessors: Comparing vectorization methods on GPUs, MICs, and CPUs. *Computer Physics Communications*, 226:18 – 29, 2018.

[7] A Sjunnesson, S Olovsson, and B Sjoblom. Validation rig - a tool for flame studies. In *International Symposium on Air Breathing Engines, 10th, Nottingham, England*, pages 385–393, 1991.

[8] A Sjunnesson, C Nelsson, and E Max. Lda measurements of velocities and turbulence in a bluff body stabilized flame. *Laser Anemometry*, 3:83–90, 1991.