SIAM J. Sci. Comput.
Vol. 38, No. 5, pp. S264–S281

# PoKiTT: EXPOSING TASK AND DATA PARALLELISM ON HETEROGENEOUS ARCHITECTURES FOR DETAILED CHEMICAL KINETICS, TRANSPORT, AND THERMODYNAMICS CALCULATIONS[*]

NATHAN YONKEE[†] AND JAMES C. SUTHERLAND[†]

**Abstract.** Simulating combustion provides detailed insights into the interplay between fluid flow, molecular diffusion, and chemical reactions. However, the breadth of state of the art simulations is restricted by both hardware capacity and software capability. In this paper we introduce PoKiTT, a library of detailed thermochemical calculations which runs on heterogeneous and parallel architectures including CPUs, GPUs, and Xeon Phi coprocessors. PoKiTT leverages both data parallelism, by using large computational domains, and task parallelism, by using directed acyclic graphs for scheduling. When compared against Cantera, PoKiTT's executions are up to 4× faster while using one CPU core. Running PoKiTT on 16 CPU cores improves execution speeds by up to 13×, and running on a GPU improves speeds by up to 40×. PoKiTT reduces the execution time, relative to one CPU core, for an illustrative simulation of methanol combustion by 9× using 16 CPU cores and 25× using a GPU.

**Key words.** kinetics, thermodynamics, diffusion, transport, GPU

**AMS subject classifications.** 80A32, 68U20, 68W10

**DOI.** 10.1137/15M1026237

## 1. Introduction.

Simulations of reacting flows improve understanding of complicated combustion systems and provide tools for improving designs of systems with important practical applications. Detailed chemical models are needed in order to capture the complex physics and chemistry involved in these flows, but require significant computational resources [4]. A recent direct numerical simulation (DNS), which fully resolved all turbulent scales of motion, investigated ignition of n-heptane using detailed chemistry [29]. The study was restricted to lean mixtures because the chemistry model used only 58 chemical species out of the original 561 [5]. Higher fidelity models require more computational resources and improved software design.

Recently, graphics processing units (GPUs) have demonstrated excellent performance in many CFD applications, including turbulent combustion [19]. GPUs are designed to maximize computational throughput, an attractive trait for high cost detailed chemistry simulations, but GPUs require new software design to perform optimally [8].

Software designs that expose and exploit both task- and data-parallelism provide tools to effectively utilize emerging architectures. Turbulent simulations inherently require large domains; these large domains provide sufficient parallelism for GPU execution [26, 14, 23, 28]. Indeed, even for solution of ODEs, sufficient parallelism has been exposed to effectively utilize GPUs [24, 18, 27]. Performance results from these studies are generally positive, although they show significant variation depending on the choice of fuel being studied. Fuels ranged in complexity from tens of species (see, e.g., [27]) to thousands (see, e.g., [24]). Studies performed on PDE systems provide

[*]Received by the editors June 16, 2015; accepted for publication (in revised form) March 28, 2016; published electronically October 27, 2016. This work was supported by National Science Foundation PetaApps award 0904631 and XPS award 1337145.

http://www.siam.org/journals/sisc/38-5/M102623.html

[†]Department of Chemical Engineering, The University of Utah, Salt Lake City, UT 84112 (nathan.yonkee@chemeng.utah.edu, james.sutherland@chemeng.utah.edu).

holistic results which demonstrated the potential of GPU acceleration; see, e.g., [26, 28]. These results represent a variety of potential applications for GPU acceleration, but they did not quantify GPU acceleration in isolation from the simulation as a whole.

Combustion simulations have long benefited from libraries that perform the detailed and complex calculations of thermodynamic, transport, and kinetics quantities. Historical packages include CHEMKIN [12, 13] and, more recently, Cantera [9]. Cantera is an open source project with interfaces for C++, Fortran, Python, and MATLAB. However, both Cantera and CHEMKIN are poorly suited for usage on multicore CPUs or GPUs.

This work stemmed from the need for efficient, portable computations of fundamental chemical properties in PDE solvers which are inherently designed for execution on parallel architectures. We leverage the preprocessing capabilities of Cantera during initialization but replace the calculations with efficient, vectorized, and portable implementations. The resulting software package is called PoKiTT (Portable Kinetics, Thermodynamics, and Transport). We perform vectorized computations using a domain specific language (DSL) called Nebo [6] that facilitates execution using multicore CPU, GPU, and Xeon Phi architectures. PoKiTT and Nebo use C++ code exclusively, and Nebo is embedded in C++ and leverages template metaprogramming to eliminate two-phase compilations that are common in other DSLs. PoKiTT provides PDE solvers with essential thermochemistry capabilities, including calculations of chemical source terms, mixture average transport properties, fluid temperature using Newton's method, thermodynamic state properties, and others. In addition to the data parallelism handled through Nebo, PoKiTT is designed with task-level parallelism such that the structure of calculations forms a directed acyclic graph [20]. These features relieve burden from application programmers, allowing them to focus on high-level design instead of low-level parallelism.

Section 2 presents the mathematical formulations used to calculate the quantities of interest. Then, section 3 describes aspects of algorithmic design, with a focus on parallelization. We present the results of this study in section 4 by comparing the performance of PoKiTT against Cantera when operating with one CPU core and then showing addition speedups for multicore and GPU execution. Section 4 demonstrates performance results on a benchmark simulation which solves 21 PDEs describing autoignition and flame propagation of methanol in a two-dimensional, partially premixed system. Finally, section 5 concludes by summarizing key findings and providing suggestions for future research.

**2. Mathematical formulation.** Models for all but the simplest calculations use parameters provided in a "chemical mechanism" which includes thermodynamic coefficients, molecular interaction parameters, and kinetics models. These models use the known (i.e., transported) thermodynamic state to estimate quantities such as heat capacity, enthalpy, transport coefficients, reaction rates, etc. PoKiTT adopts Cantera's input format and uses Cantera as a preprocessor to load the input and calculate model parameters, as discussed below.

**2.1. Thermodynamics.** Thermodynamic properties are typically expressed in terms of polynomials that are fit to individual species and have been standardized [17]. PoKiTT supports the 7-coefficient NASA model [17] to describe the enthalpy per unit mass of each species,

$$h_i = \hat{R}\left(a_{1,i}T + \frac{a_{2,i}T^2}{2} + \frac{a_{3,i}T^3}{3} + \frac{a_{4,i}T^4}{4} + \frac{a_{5,i}T^5}{5} + a_{6,i}\right), \tag{1}$$

where $\hat{R}$ is the specific gas constant, $T$ is the fluid temperature, and $a_{1,i}, \dots, a_{6,i}$ are model coefficients for species $i$. The heat capacity of a pure species is defined as

$$c_{p,i} = \frac{\partial h_i}{\partial T}, \tag{2}$$

which is easily found from (1). The Gibbs energy of a pure species is defined as

$$g_i = h_i - T s_i, \tag{3}$$

where $s_i$ is the entropy per unit mass of pure species $i$ computed using

$$s_i = R\left(a_{1,i} \ln T + a_{2,i} T + \frac{a_{3,i} T^2}{2} + \frac{a_{4,i} T^3}{3} + \frac{a_{5,i} T^4}{4} + a_{7,i}\right). \tag{4}$$

The system temperature can be calculated from species mass fractions ($Y_i$) and mixture enthalpy such that we satisfy

$$h = \sum_i^N Y_i h_i(T). \tag{5}$$

This is performed using Newton's method, taking advantage of (2). PoKiTT vectorizes the temperature calculation and, to avoid thread divergence on GPU, iterates the entire set of points within a warp until convergence is reached on all points.

**2.2. Transport.** Evaluating multicomponent transport properties is very expensive, but they play an important role in fluid flow. A standard, significantly quicker approximation estimates average properties using mixing rules, functions of the transport properties of each pure species, and the overall composition [11]. The following molecular interaction parameters are provided for each species: molecular geometry, polarizability, dipole moment, Lennard-Jones well depth, Lennard-Jones diameter, and rotational relaxation. Formulations are unique for each transport property, but they follow the same general procedure: first, fundamental laws of gases are used to generate data on pure species properties; second, these pure properties are fit to a polynomial in temperature; then, at each time step, the pure property polynomials are evaluated and the results applied to a mixing rule. Determination of the polynomial coefficients in the second step is performed once, during initialization, and PoKiTT retrieves the model coefficients from Cantera directly.

The viscosity of a pure species is calculated using the kinetic theory of gases [10]. Once a dataset of pure species viscosities is calculated, it is regressed to the form

$$\mu_i = \left(T^{1/2}\left(b_{0,i} + b_{1,i} \ln T + b_{2,i} (\ln T)^2 + b_{3,i} (\ln T)^3 + b_{4,i} (\ln T)^4\right)\right)^2, \tag{6}$$

where $\mu_i$ is the viscosity of pure species $i$ and $b_{1,i}, \dots, b_{4,i}$ are regression coefficients. The mixing rule for viscosity is a semiempirical formula as provided in [2],

$$\mu = \sum_k \frac{\mu_k X_k}{\sum_j \Phi_{k,j} X_j}, \tag{7}$$

where $\mu$ is the mixture viscosity used in Newton's law of viscosity, $X_k$ are mole species mole fractions, and the dimensionless quantity

$$\Phi_{k,j} = \frac{\left[1 + \sqrt{\frac{\mu_k}{\mu_j}\sqrt{\frac{M_j}{M_k}}}\right]^2}{\sqrt{8}\sqrt{1 + M_k/M_j}}. \tag{8}$$

Binary diffusion coefficients, also derived from kinetic theory of gases [2], are collected into a dataset which is then regressed to the following polynomial in temperature:

$$D_{i,j} = T^{3/2}\left(d_{i,j}^{(0)} + d_{i,j}^{(1)}\ln T + d_{i,j}^{(2)}(\ln T)^2 + d_{i,j}^{(3)}(\ln T)^3 + d_{i,j}^{(4)}(\ln T)^4\right), \tag{9}$$

where $D_{i,j}$ is the binary diffusion coefficient of species $i$ with respect to species $j$ and $d_{i,j}^{(0)}, \dots, d_{i,j}^{(4)}$ are regression coefficients. The mixing rule to calculate the effective diffusivity of species $k$ is derived in [2] as

$$D_i^{\text{eff}} = \left(\sum_{j\neq i}^{n_s}\frac{X_j}{D_{i,j}} + \frac{X_i}{1-Y_i}\sum_{j\neq i}^{n_s}\frac{Y_j}{D_{i,j}}\right)^{-1}, \tag{10}$$

where $n_s$ is the number of species. These coefficients are used in Fick's law of diffusion to calculate the diffusive mass flux of $i$ relative to the mass average velocity,

$$\mathbf{J}_i = -\rho D_i^{\text{eff}}\nabla Y_i, \tag{11}$$

where $J_i$ is the mass flux of species $i$, and $\rho$ is the fluid density. The sum of the all diffusive mass flux is zero by definition, but this constraint is not enforced by the mixing rule approximation. Instead, we account for net residual flux by subtracting some flux from each species in proportion to the magnitude of its flux.

The thermal conductivity of dilute monatomic gases is well known [2], although this formulation provides a good approximation for polyatomic gases as well. Once the pure thermal conductivities have been calculated, they are regressed to a polynomial,

$$\lambda_i = T^{1/2}\left(c_{0,i} + c_{1,i}\ln T + c_{2,i}(\ln T)^2 + c_{3,i}(\ln T)^3 + c_{4,i}(\ln T)^4\right), \tag{12}$$

where $\lambda_i$ is the thermal conductivity of pure species $i$ and $c_{1,i}, \dots, c_{4,i}$ are regression coefficients. The mixture thermal conductivity is evaluated using a combination averaging mixing rule [16],

$$\lambda = \frac{1}{2}\left(\sum_k X_k\lambda_k + \frac{1}{\sum_k X_k/\lambda_k}\right). \tag{13}$$

Heat fluxes in diffusive systems are calculated as a combination of Fourier's law of heat conduction and molecular diffusion,

$$\mathbf{q} = -\lambda\nabla T + \sum_i^{n_s} h_i\mathbf{J}_i, \tag{14}$$

where $q$ is the net heat flux and $h_i$ and $J_i$ are defined in (1) and (11).

For all of the transport properties listed, PoKiTT extracts the regressed polynomial coefficients, evaluates pure properties, and applies the mixing rule. This results in simple and efficient code to quickly approximate multicomponent transport properties.

**2.3. Chemical kinetics.** Chemical mechanisms consist of many elementary reactions, each of which contributes to the net source of the reactants and products. All elementary reactions have at most two reactants and/or two products, and each stoichiometric coefficient is an integer (e.g., $A + B \rightleftharpoons 2C$). The net source of species $i$ is the sum of all reactions in which it participates,

$$\omega_i = \frac{W_i}{\rho} \sum_{j=1}^{n_r} \left(\nu''_{ij} - \nu'_{ij}\right) \Omega_j, \tag{15}$$

where $\omega_i$ is net source of species $i$, $W_i$ is the molecular weight of species $i$, $\rho$ is the mixture density, $\nu''_{ij}$ and $\nu'_{ij}$ are the forward and reverse stoichiometric coefficients of species $i$ in reaction $j$, respectively, and $\Omega_j$ is the rate of progress of reaction $j$. The rate of progress for an elementary reaction without pressure dependence is given by the difference between forward reaction rates

$$\Omega_{j,f} = k_{j,f} \prod_{k=1}^{n_s} C_k^{\nu'_{kj}} \tag{16}$$

and reverse reaction rates

$$\Omega_{j,r} = k_{j,r} \prod_{k=1}^{n_s} C_k^{\nu''_{kj}}, \tag{17}$$

resulting in a net value,

$$\Omega_j = \Omega_{j,f} - \Omega_{j,r}, \tag{18}$$

where $k_{j,f}$ and $k_{j,r}$ are the forward and reverse Arrhenius rate constants and $C_k$ is the molar concentration of species $k$. The forward Arrhenius rate constant is evaluated as

$$k_{j,f} = A_j T^{\beta_j} \exp\left(\frac{-E_j}{RT}\right), \tag{19}$$

where $A_j$ is the pre-exponential factor, $\beta_j$ is the temperature exponent, $E_j$ is the activation energy, and $R$ is the universal gas constant; $A_j$, $\beta_j$, and $E_j$ are given in the mechanism. The reverse rate constant is defined such that the reaction is driven towards a state of equilibrium (i.e., net rate is zero). For reversible reactions the reverse rate constant is evaluated as

$$k_{j,r} = k_{j,f} \exp\left(\ln(C_t) \sum_{j=1}^{n_r} \left(\nu''_{ij} - \nu'_{ij}\right) - \frac{\Delta G}{RT}\right), \tag{20}$$

where $C_t$ is the concentration of the mixture and $\Delta G$ is the net Gibbs energy,

$$\Delta G = \sum_{i=0}^{n_s} \nu''_i g_i - \nu'_i g_i, \tag{21}$$

where $g_i$ is calculated using (3).

Reaction rate laws which account for the effects of pressure use either third body or falloff formulations of $k_{j,f}$, but are otherwise identical to simple Arrhenius rate laws. For a third body reaction, (19) includes a factor to account for the concentration of every species,

$$k_{j,f} = M_j k_{j,f}, \tag{22}$$

where $M_j$ is a weighted sum over all species,

$$M_j = \sum_{i=1}^{n_s} m_{j,i} C_i, \tag{23}$$

where $m_{j,i}$ is the weighting factor for species $i$ in reaction $j$; values of $m_{j,i}$ are specified in the mechanism. Because all quantities in the following discussion of falloff reactions are for a specific reaction, $j$, we drop this subscript. We first calculate a reduced pressure,

$$P_r = \frac{k_0 M}{k_\infty}, \tag{24}$$

where $k_0$ and $k_\infty$ are evaluated using (19) but with separate parameters: $k_0$ is evaluated using a low pressure parameter set, while $k_\infty$ is evaluated using a high pressure set. The forward rate constant used in (16) is calculated as a function of the reduced pressure,

$$k_f = k_\infty \frac{P_r}{1 + P_r} F, \tag{25}$$

where $F$ is another falloff model. The simplest model is the Lindemann falloff reaction which uses $F = 1$. PoKiTT also supports Troe falloff reactions which use a costly model for $F$, involving multiple logarithm evaluations. The formulation for Troe kinetics is involved and not directly relevant to this discussion; we refer the reader to [30] for a full description. All of the variants described here are fully supported in PoKiTT.

## 3. Algorithm design.

PoKiTT groups individual thermochemistry computations into "tasks" which are composed into a directed acyclic graph (DAG) according to their data dependencies at runtime. Task decomposition provides dynamic runtime scheduling, prevents task duplication, and facilitates creating and using highly optimized tasks. Dynamic scheduling orders executions using the concept of a DAG, as discussed in [20]. Tasks identify which value they compute and what data dependencies they have; e.g., the task for viscosity requires values for temperature and composition. The DAG, or task graph, uses graph nodes to represent tasks, and a node's out-edges represent its data dependencies. The execution graph is the transpose of the dependency graph; the task at the "top" of the dependency graph is the final task to be executed; i.e., it is the quantity we wish to calculate. Tasks at the "bottom" of the dependency graph are nodes without data dependencies, such as the solution variables from the previous timestep/iteration for transient calculations.

One noteworthy advantage of PoKiTT's DAG representation is that it avoids redundant calculations and naturally handles complex interdependencies. Cantera, on the other hand, uses a static schedule at each cell and unavoidably duplicates some calculations because it does not have knowledge of the entire calculation structure (as PoKiTT does by virtue of the DAG). PoKiTT automatically links repetitious evaluations into a single node in the DAG, thereby eliminating repetitions, and then reuses values as needed.

Task concurrency is automatically exposed by the edges connecting each task, and this facilitates automated runtime concurrency [20]. Furthermore, graphs containing heterogeneous nodes (i.e., a mixture of tasks to be executed on either GPU or CPU) perform data transfers asynchronously between host and device while simultaneously backfilling tasks to hide transfer costs. PoKiTT supports all of these features.

A key aspect of DAG representation is the task scheduling algorithm. The scheduler manages reclamation (reuse once a field is no longer needed in the calculation), memory transfer (migrating fields between CPU and GPU), and execution of tasks once dependencies are satisfied. This allows overlap of computation with host/device memory transfers in heterogeneous calculations.

**3.1. Parallelism.** The actual evaluation of a task involves one to hundreds of field assignments. A field is a simple abstraction used to represent a three-dimensional array for a particular quantity. Nebo [6], a DSL, simplifies field assignments by lifting operators over fields and supporting stencil operations on structured grids. Nebo overloads the C++ `<<=` operator for assignment operations generated by the template metaprogram engine within Nebo. Other DSLs written specifically for GPU programming (see, e.g., [1, 3]) aimed to improve performance or simplify writing code, but Nebo is different because it generates both CUDA code and parallelized C++ code.

Each Nebo assignment corresponds to an inlined statement (consisting of nested loops on a grid variable, or a single loop on a variable not associated with a grid) on CPU or a kernel call on GPU. Thus, application developers can control the granularity of loops/kernels by the complexity of a Nebo assignment statement. There are some inherent limitations on the degree of loop fusion/inlining that can be accomplished because many of the loops are over species to accumulate contributions (see the summations in section 2), and the number of species present in the system is determined at runtime. While one could overcome some of these limitations by autogenerating code specific to a given chemical mechanism, PoKiTT does not support this at present. Quantities related to individual species such as mass fractions, heat capacities, reaction rates, etc. (see section 2), are held individually (i.e., a struct of fields rather than a field of structs), and we have not considered effects of inverting this structure primarily because this is not presently supported in Nebo. High performance, massively parallel combustion simulation codes such as Sandia's S3D solver arrange memory in the same way.

Nebo supports discrete interpolation, gradient, divergence, and filtering operators for fields on structured grids, as well as particle-cell interpolants. Since Nebo fields wrap existing memory blocks, and Nebo also supports the concept of "point-fields" that are not associated with a mesh, it can also interface with unstructured codes as well—it just does not presently support stencil operations on those fields. The quantities calculated by PoKiTT, however, involve only pointwise calculations. Therefore, PoKiTT can be used in conjunction with both structured and unstructured application codes. Furthermore, because it only involved pointwise calculations, PoKiTT only requires on-node information; in a domain-decomposed distributed parallel application, PoKiTT does not need access to MPI communication, etc.

Nebo provides data parallelism on multicore and GPU architectures using identical syntax in the source code. Any programmer using Nebo can run CUDA code by simply using the overloaded `<<=` operator; one does not need any knowledge about CUDA. Nebo uses the POSIX threads (Pthreads) interface to generate and manage its multicore thread pool. For multicore applications, the domain is split into chunks such that one chunk is assigned to each core. The GPU backend is different compared against the multicore implementation because a GPU operates using many more threads which are much lighter. Nebo performs decomposition on GPUs by

Table 1
*The number of species ($n_s$) and number of reactions ($n_r$) for the mechanisms considered here.*

| Mechanism | Reference | $n_s$ | $n_r$ |
|---|---|---|---|
| H2 | [22] | 10 | 40 |
| MeOH | [15] | 21 | 93 |
| GRI | [25] | 53 | 325 |

dividing the domain into individual cells; then Nebo assigns one thread to each cell.

Some additional details of how PoKiTT decomposes a particular problem will be provided in section 4.3, where an example application is considered.

**4. Performance.** CPU timings were collected using a single workstation with two Intel Xeon E52680 Sandy Bridge processors (2.7 GHz, 8 cores per socket) and DDR3 1600 memory. The GPU node contained a single NVIDIA Tesla K20 GPU which contains 2496 CUDA cores operating at 706 MHz and GDDR5 memory operating at 2.6 GHz. The Xeon CPUs were released 6 months prior to the K20, and the CPUs had a combined MSRP nearly identical to the K20. The Xeon CPUs have a combined thermal design power (TDP, a conservative estimate of power consumption) of 260 watts, and the GPU's TDP is 225 watts. We used gcc v4.9.1 with the flag `-O3`, and CUDA code was compiled with nvcc 6.0 and flag `-march=native`.

We benchmarked the performance of PoKiTT using a two-dimensional grid initialized with arbitrary linear profiles of temperature and mass fractions with a constant pressure of one atmosphere. Newton's method in the temperature solve used an initial guess of a separate linear temperature profile such that it differed from the true solution by +25K at the highest temperature and –25K at the lowest temperature. We executed each task 20 times, then discarded the top and bottom quartiles, and then recorded the arithmetic mean of the remaining 10 executions. Using this procedure, we collected timings for $h_i$, $c_p$, $T$, $D_i^{\mathrm{eff}}$, $\mu$, $\lambda$, and $\omega_i$ as defined in section 2, and $c_p$ is a mixture average derived using (2) and (5). We used three mechanisms, as summarized in Table 1. We chose GRI 3.0 due to its widespread use and applicability; we chose hydrogen (H2) because it is the simplest fuel for detailed chemistry; and we chose methanol (MeOH) because its size was between the other two. To avoid considerations of host/device transfers, all computations on the GPU system were run entirely on the device, and all necessary memory was initialized on the device.

In section 4.1, we compare execution times using one core against Cantera to demonstrate speedups from task-based execution. Then, in section 4.2, we present speedups achieved by leveraging data parallelism with multiple CPUs and a single GPU. Finally, in section 4.3, we present multicore and GPU speedups for a simulation of methanol combustion involving 21 PDEs and 93 elementary chemical reactions.

**4.1. Serial performance.** Table 2 summarizes execution times and corresponding speedups of PoKiTT over Cantera for each task on a mesh of 2048 × 1024 cells using the methanol mechanism. We observed the largest speedups for enthalpy, heat capacity, and viscosity and use these as examples for explanation.

PoKiTT uses a few optimizations to improve performance relative to Cantera. For example, the calculation of $h_i$ in (1) was rewritten as

$$h_i = (Ra_{6,i}) + \sum_{j=1}^{5} a'_{j,i} T^j, \tag{26}$$

where the new polynomial coefficients,

$$a'_{j,i} = \frac{Ra_{j,i}}{j}, \tag{27}$$

are precomputed at startup and cached. Caching the values of $a'_{j,i}$ reduces operation count, and we applied Horner's rule for further reduction. Cantera also implements Horner's rule, but it repeats the calculation of $a'_{j,i}$ at every grid cell. The large speedup in the viscosity evaluation is due to lumping constants in (8) and caching intermediate values during the summation in (7). We cached every term involving molecular weights in (8), including the denominator. The square root of the ratio of viscosities had to be calculated once per grid loop, but the result was cached and reused in the species loop. These optimizations are significant because operations would be repeated at every cell, which corresponds to avoiding millions of operations using this grid size.

Table 2
*Timings for Cantera and PoKiTT (single-core) using the MeOH mechanism on a grid of* $2048 \times 1024$.

| Quantity | Cantera [s] | PoKiTT [s] | Speedup |
|---|---|---|---|
| $h_i$ | 1.45 | 0.46 | 3.1637 |
| $c_p$ | 0.76 | 0.20 | 3.8913 |
| $T$ | 1.79 | 1.34 | 1.3342 |
| $\lambda$ | 0.92 | 0.63 | 1.4687 |
| $\mu$ | 19.1 | 8.04 | 2.3711 |
| $D_i^{\text{eff}}$ | 9.87 | 7.00 | 1.4109 |
| $\omega_i$ | 19.6 | 17.1 | 1.1410 |

Cantera requires one to set the thermodynamic state of the system. This is done by aggregating species mass fractions into a buffer, passing that through to Cantera, collecting the results in another buffer (in the case of a result such as reaction rates), and transferring those back to the appropriate fields. PoKiTT avoids this intermediate buffer usage by operating directly on the appropriate fields, which are each stored independently. We have not directly considered the effect of storage options on species ("struct of fields" versus "field of structs"). While one may argue that the "field of structs" variable layout may be more appropriate for PoKiTT calculations, in a PDE solver the "struct of fields" layout is more appropriate due to the stencil operations. Given the results shown in Table 2, it appears that any cache effects due to the "struct of fields" layout in PoKiTT do not lead to degraded performance relative to Cantera.

Results from previous studies [24, 18, 27] imply that performance may be related to the choice of chemical mechanism or the size of the domain. We chose two tasks, viscosity, $\mu$, and source terms, $\omega_i$, to demonstrate the relation between performance and mechanism choice or problem size. Figure 1 shows speedups for the viscosity evaluation on a range of grid sizes for all three mechanisms. There is a small, yet distinct, performance loss for very small problem sizes. Performing field operations through Nebo adds a small amount of overhead, but the additional cost is negligible compared to the net performance gain. Performance on GRI appears to be slightly worse than the others, but again it is nearly indistinguishable and negligible compared to the overall improvement. Figure 2 shows speedups for the reaction rate evaluation on the same grids as Figure 1. Again, we see the slight performance loss for small problems, but this graph shows a distinct difference between hydrogen and the other mechanisms. The source term evaluation is unique because the complexity of the calculations changes between mechanisms in addition to the number of reactions, which complicates a scaling analysis. The hydrogen mechanism does not contain many of the falloff reactions (see section 2.3) that are present in the other two mechanisms. We see performance benefits for hydrogen, but not the others, because our optimizations were insignificant when compared with the cost of multiple exponential and logarithm evaluations required for falloff reactions.

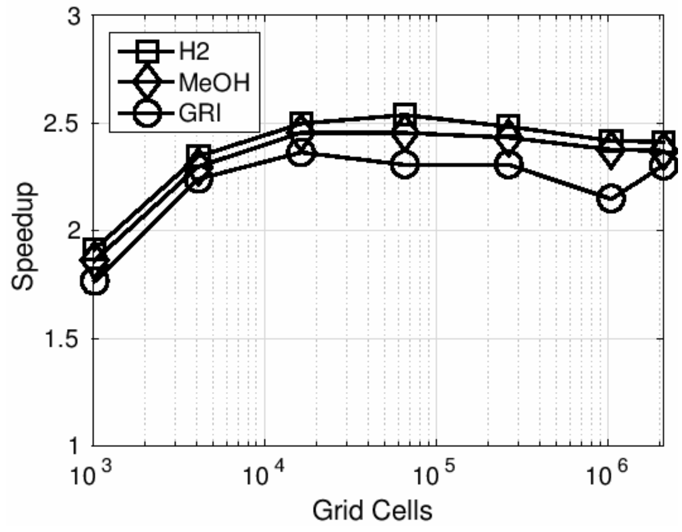

Fig. 1. *Serial PoKiTT speedup relative to Cantera for viscosity evaluation on the $H2$, MeOH, and GRI mechanisms across a range of problem sizes.*

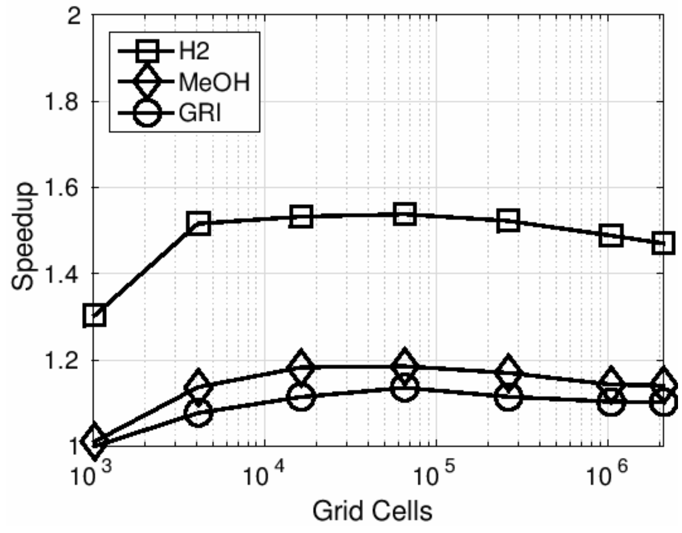

Fig. 2. *Serial PoKiTT speedup relative to Cantera for kinetics evaluation on the $H2$, MeOH, and GRI mechanisms across a range of problem sizes.*

Detailed results for other quantities are not included, but they follow the same trend as the viscosity evaluation.

**4.2. Parallel performance.** We now compare PoKiTT's serial execution timings with parallel timings using multiple CPU cores and a GPU. Table 3 lists the execution times and speedups achieved on the parallel implementations for the same test case discussed in section 4.1. We observed consistent, significant speedups when running on either parallel architecture. The speedup for multicore CPU evaluations varies between 8× and 13×, while speedups on the GPU vary between 13× and 43×. The ratio of GPU speedup over multicore speedup for each calculation was approximately 2.4×, except $h_i$ and $\omega_i$, for which the ratio was larger. We speculate that the high GPU speedup of the $h_i$ kernel is because it depends on a single field, temperature, while the other quantities in Table 3 have multiple fields as dependencies. Therefore, $h_i$ has a higher computational intensity (computation relative to memory reads), and this results in low memory pressure and may favor architectures with a higher computational throughput. The evaluation of the reaction rates $\omega_i$ involves multiple exponential function evaluations and thus also has a high computational intensity relative to the other terms in Table 3.

Table 3
*Parallel performance results for PoKiTT on CPU and GPU using the MeOH mechanism on a grid of* $2048 \times 1024$.

| Quantity | 1-core [s] | 16-core [s] | Speedup | GPU [s] | Speedup |
|---|---|---|---|---|---|
| $h_i$ | 0.46 | 0.034 | 13.5 | 0.0106 | 43 |
| $c_p$ | 0.20 | 0.024 | 8.0 | 0.0109 | 18 |
| $T$ | 1.34 | 0.173 | 7.8 | 0.1033 | 13 |
| $\lambda$ | 0.63 | 0.065 | 9.6 | 0.0276 | 23 |
| $\mu$ | 8.04 | 0.777 | 10.4 | 0.3519 | 23 |
| $D_i^{\text{eff}}$ | 7.00 | 0.521 | 13.4 | 0.2642 | 26 |
| $\omega_i$ | 17.1 | 1.436 | 11.9 | 0.4022 | 43 |

Next, we investigated the impact of mechanism size on observed speedups. We chose four tasks as representative examples: reaction rates ($\omega_i$), diffusivities ($D_i^{\text{eff}}$), viscosity ($\mu$), and temperature ($T$). We chose these due to their different computational characteristics (see section 2). Figure 3 shows results for the multicore implementation, and Figure 4 shows results for the GPU implementation. These figures show that our results are mostly independent of mechanism choice, although the temperature solve appears to perform better on larger mechanisms.

Next, we consider the effect of domain size on parallel performance for calculations of $\omega_i$, $\mu$, and $T$. We did not include $D_i^{\text{eff}}$ because its results are visually indistinguishable from $\mu$. Figures 5 and 6 show results for the 16-core and GPU implementations, respectively, for domain sizes ranging from $64 \times 64$ to $2048 \times 1024$ cells. There is a strong correlation between domain size and parallel performance for both CPU and GPU implementations. The speedups between each task vary in magnitude, but each calculation's correlation with grid size is consistent. The CPU saturates at smaller problem sizes ($512^2$) than the GPU ($1024^2$). We expect the GPU to perform best at larger grid sizes because GPUs can operate on thousands of threads simultaneously [21], which requires thousands of cells in total to saturate the threads of a GPU, compared with 16 threads for multicore CPU execution.

**4.3. Triple flame.** Sections 4.1 and 4.2 established PoKiTT's performance by examining individual terms. To assess PoKiTT's performance on an application involving combinations of these terms, we simulated a two-dimensional reaction diffusion system. This involves solution of one PDE for enthalpy transport and $n_s - 1$ PDEs

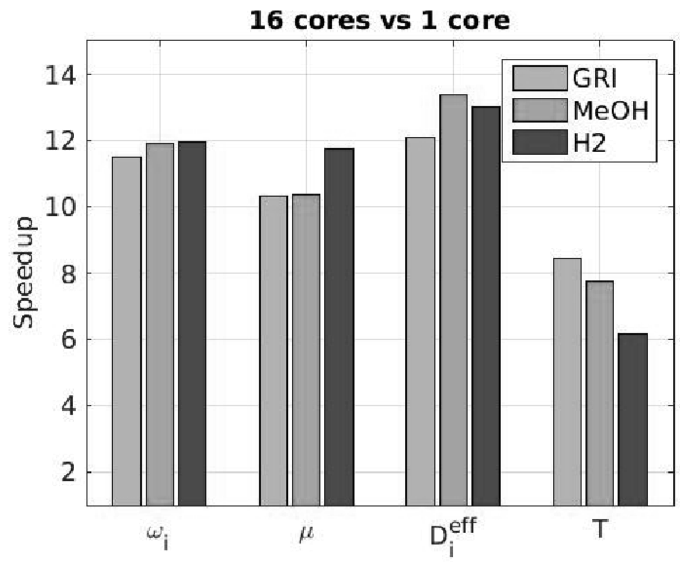

FIG. 3. 16-*core speedups across all three mechanisms for a variety of calculations.*

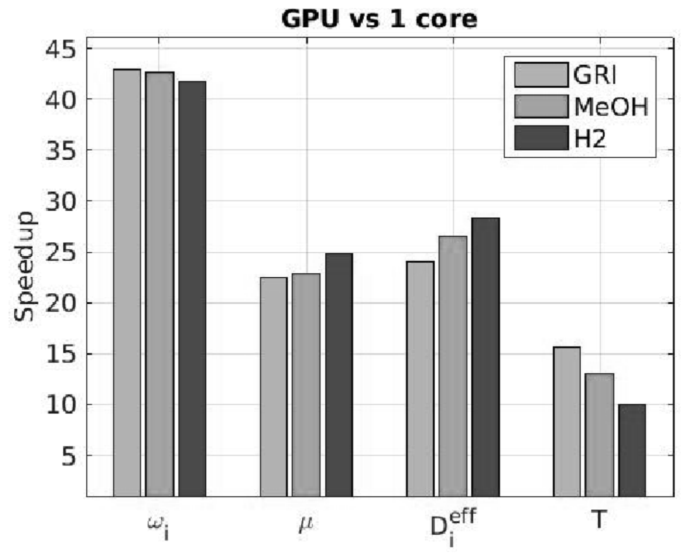

FIG. 4. *GPU speedups across all three mechanisms for a variety of calculations.*

for transporting species mass fractions,

$$\frac{\partial \rho h}{\partial t} = -\nabla \cdot \mathbf{q}, \tag{28}$$

$$\frac{\partial \rho y_i}{\partial t} = -\nabla \cdot \mathbf{J}_i + \omega_i, \tag{29}$$

with $\mathbf{J}_i$, $\mathbf{q}$, and $\omega_i$ given by (11), (14), and (15), respectively. Density of the fluid is constant, and conservation of overall mass is enforced using the one nontransported species as a balance.

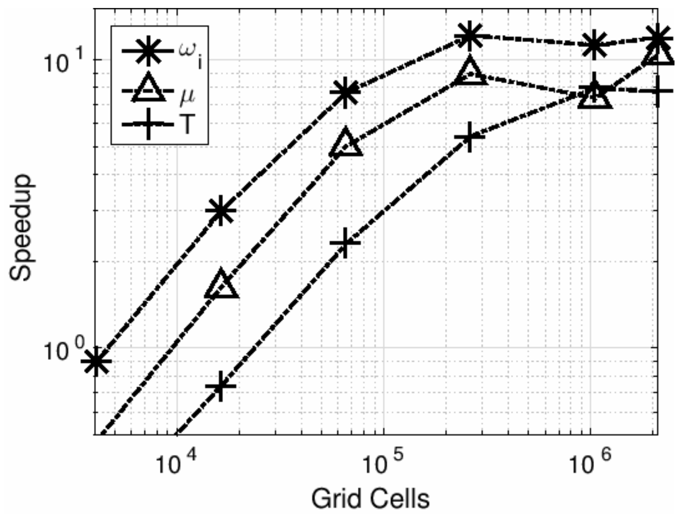

FIG. 5. *16-core speedups over serial execution across a range of domain sizes for source terms, viscosity, and temperature.*

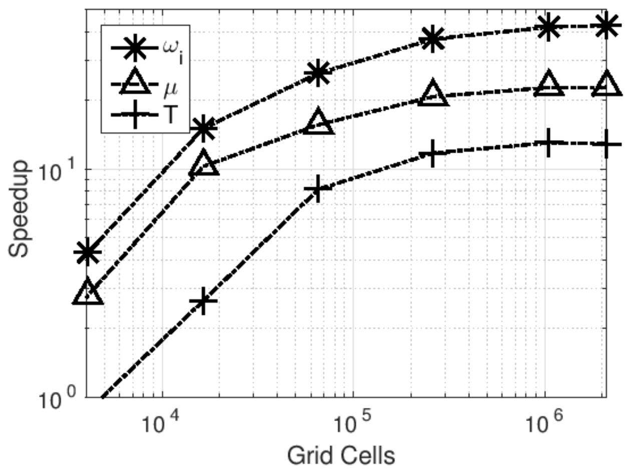

FIG. 6. *GPU speedups over serial execution across a range of domain sizes for source terms, viscosity, and temperature.*

These equations are discretized on a uniform, structured mesh using a second-order finite volume technique. PoKiTT was used to calculate the reaction rates, temperature, and diffusion coefficients for the heat and mass diffusion fluxes, which were then used to construct the right-hand sides of the above PDEs and advanced using an explicit time integrator.

Methanol was the fuel and air the oxidizer; the initial condition for the mixture's composition was a linear profile of the mixture fraction, with a fuel rich zone on the left, a fuel lean zone on the right, and a stoichiometric mixture in the center.

With methanol as the fuel, we must solve 21 highly nonlinear PDEs involving 93 elementary reactions (see Table 1).

The temperature was initialized as a sharp Gaussian edge at the bottom of the domain, and the initial pressure is 1 atm throughout the domain. Density is initialized using the ideal gas law, $\rho = \frac{pM}{RT}$, where $M$ is the mixture molecular weight. Figure 7 shows the autoignition event of the mixture on the left and the resulting structure on the right. After ignition, the flame front propagates as a premixed flame in the center, while diffusion flames form on each side.

Figure 8 shows the DAG for this simulation but using a mechanism with seven species [7] to simplify the graph. Here, “D” nodes correspond to $D_i^{\text{eff}}$ and “r” nodes correspond to $\omega_i$. We maximize task parallelism using a priority scheduler: long chains have high priorities, while short chains have low priorities; we backfill (when using a multithreaded scheduler) low priority tasks once resources become available. The DAG shows that “r” nodes are in a chain different from “D” nodes which would allow them to be evaluated concurrently. From Table 3 we expect these tasks will occupy a significant portion of execution time and could be evaluated simultaneously by executing one task on the CPU and the other on a GPU.

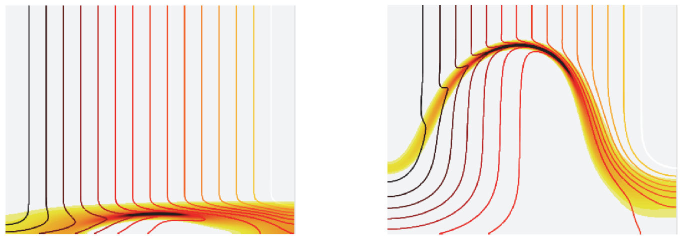

Fig. 7. *Autoignition of methanol in air shown on the left and the resulting triple flame on the right. Mixture fraction is shown using contour lines, black is fuel rich, and red is stoichiometric. Reaction zone is shown in pseudocolor overlaying the contour lines. Color is available online only.*

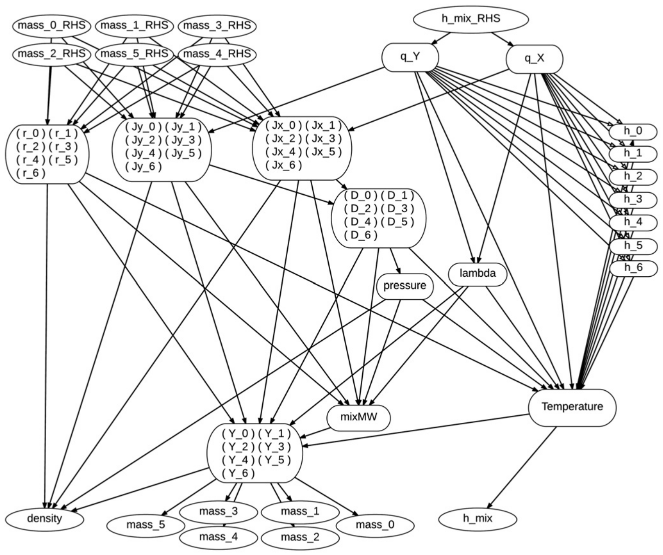
mass_0_RHS
mass_1_RHS
mass_3_RHS
mass_2_RHS
mass_5_RHS
mass_4_RHS
h_mix_RHS
q_Y
q_X
( r_0 ) ( r_1 ) ( r_2 ) ( r_3 ) ( r_4 ) ( r_5 ) ( r_6 )
( Jy_0 ) ( Jy_1 ) ( Jy_2 ) ( Jy_3 ) ( Jy_4 ) ( Jy_5 ) ( Jy_6 )
( Jx_0 ) ( Jx_1 ) ( Jx_2 ) ( Jx_3 ) ( Jx_4 ) ( Jx_5 ) ( Jx_6 )
h_0
h_1
h_2
h_3
h_4
h_5
h_6
( D_0 ) ( D_1 ) ( D_2 ) ( D_3 ) ( D_4 ) ( D_5 ) ( D_6 )
pressure
lambda
mixMW
Temperature
( Y_0 ) ( Y_1 ) ( Y_2 ) ( Y_3 ) ( Y_4 ) ( Y_5 ) ( Y_6 )
density
mass_5
mass_3
mass_4
mass_1
mass_2
mass_0
h_mix

Fig. 8. *DAG for the reaction diffusion problem using a mechanism with seven species.*

The following results were run using a mesh with $512 \times 512$ cells and using a forward Euler integrator. This size was chosen because it was the largest which could fit entirely within the GPU global memory for this mechanism. Note that for larger domains we typically use MPI to distribute the workload over multiple nodes, where PoKiTT would be deployed on each node.

Figure 9 shows speedups over serial execution for the multicore and GPU implementations. The simulation ran between 20–30 times faster using a GPU and 10 times faster using 16 CPU cores. These speedups are similar to speedups of the individual tasks for diffusion coefficients and for chemical source terms shown in Figures 5 and 6.

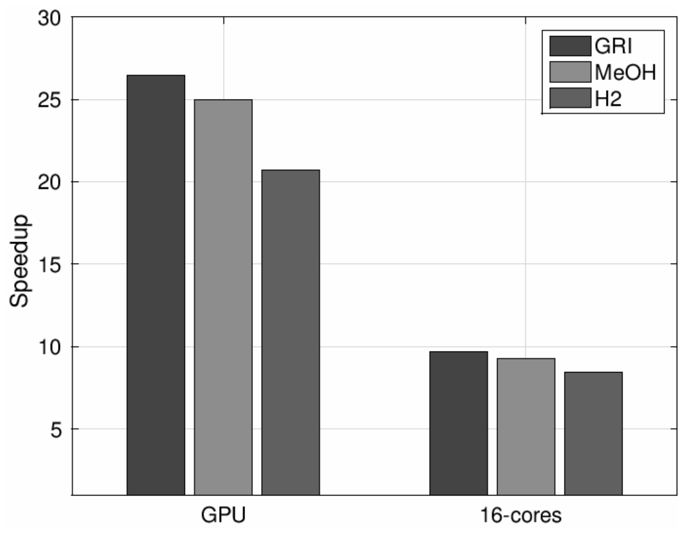

FIG. 9. *Speedup of the triple flame simulation using* 16 *cores or the GPU compared with single-core execution.*

Figure 9 also indicates that the GPU speedup improves as the mechanism size increases, which is surprising when we consider the results in Figure 4. This prompted us to compare individual task execution times and compare between mechanisms and architectures. Figure 10 shows the fractional time spent in the largest tasks for the GRI mechanism, with CPU execution on the left and GPU execution on the right. We grouped tasks into four categories: $D_i^{\text{eff}}$, $\omega_i$, stencil operations, and everything else. For the multicore execution, calculating $D_i^{\text{eff}}$ and $\omega_i$ consumed the vast majority of the time. When compared to the GPU execution, the time spent evaluating source terms is reduced significantly (see Table 3) and therefore reduces the overall simulation time significantly.

Figure 11 shows the time spent on each task using the H2 mechanism, with CPU timings on the left and GPU timings on the right. The most significant difference between Figures 10 and 11 is in the GPU time distribution. While the source terms occupy a similar portion of time, the $D_i^{\text{eff}}$ calculation occupies less than a fifth of total time, significantly less than for GRI. Calculation of the $D_i^{\text{eff}}$ kernel scales as $n_s^2$, thus becoming relatively more expensive when more species are involved. The $\omega_i$ calculation scales with both the $n_s$ and $n_r$, and $n_r$ typically increases superlinearly with $n_s$ (see Table 1), which explains why evaluation of $\omega_i$ consumes an increasing portion of time on large mechanisms. Figure 4 shows that speedup on every task is insensitive to mechanism choice; therefore, we infer that the speedup differences are related to the scaling behaviors of the tasks involved.

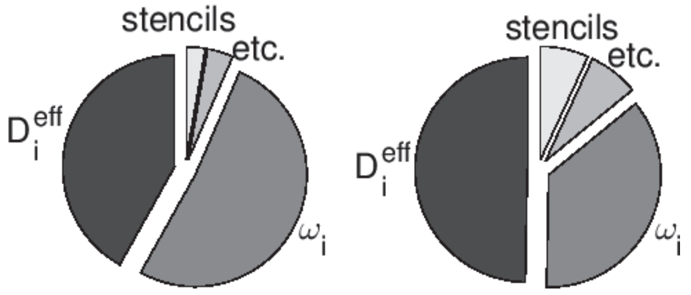

FIG. 10. *Relative time spent per task using the GRI mechanism for* 16*-core CPU execution (left) and GPU execution (right).*

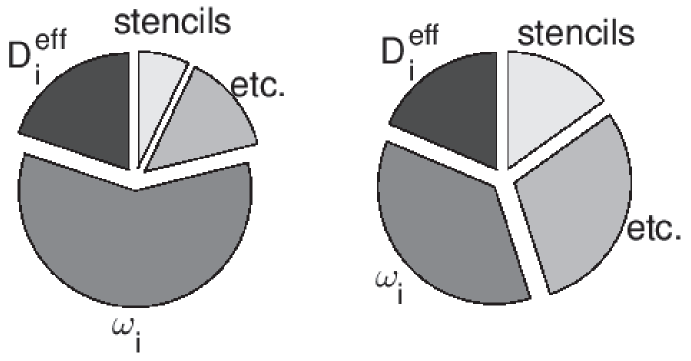

FIG. 11. *Relative time spent per task using the H*2 *mechanism for* 16*-core CPU execution (left) and GPU execution (right).*

**5. Conclusions.** We presented a new software package, PoKiTT, which offers a suite of thermochemistry calculations to model detailed chemistry in PDE solvers. PoKiTT uses the DSL Nebo to automatically generate high performance code for GPUs and multicore CPUs, and it leverages DAG approaches to eliminate redundant calculations and facilitate task-level parallelism. When compared against the general purpose software package Cantera, PoKiTT provides speedups of up to 4× for serial executions primarily by streamlining execution to minimize the number of operations for each cell. PoKiTT employs data-parallelism for multicore and GPU configurations, and it layers task level parallelism on top of that to allow multiple tasks to be executed concurrently. Comparing the parallel implementations with serial execution, we saw speedups between 8× to 13× using 16 CPU cores and speedups between 20× to 40× using a GPU. We simulated a partially premixed methanol flame using PoKiTT; overall execution time was reduced by approximately 9× on 16 CPU cores and 25× on a GPU relative to execution time using one CPU core.

## REFERENCES

[1] M. BAUER, S. TREICHLER, AND A. AIKEN, *Singe*, in Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '14), 2014, pp. 119–130, http://dx.doi.org/10.1145/2555243.2555258.

[2] B. R. Bird, W. E. Stewart, and E. N. Lightfoot, *Transport Phenomena*, Wiley, New York, 2007.
[3] D. D. J. Chandar, J. Sitaraman, and D. Mavriplis, *CU++: An object oriented framework for computational fluid dynamics applications using graphics processing units*, J. Supercomput., 67 (2013), pp. 47–68, http://dx.doi.org/10.1007/s11227-013-0985-9.
[4] J. H. Chen, *Petascale direct numerical simulation of turbulent combustion - Fundamental insights towards predictive models*, P. Combust. Inst., 33 (2011), pp. 99–123, http://dx.doi.org/10.1016/j.proci.2010.09.012.
[5] H. J. Curran, P. Gaffuri, W. J. Pitz, and C. K. Westbrook, *A comprehensive modeling study of iso-octane oxidation*, Combust. Flame, 129 (2002), pp. 253–280, http://dx.doi.org/10.1016/S0010-2180(01)00373-X .
[6] C. Earl, M. Might, A. Bagusetty, and J. C. Sutherland, *Nebo: An efficient, parallel, and portable domain-specific language for numerically solving partial differential equations*, J. Syst. Softw., 2016, http://dx.doi.org/10.1016/j.jss.2016.01.023.
[7] D. Fernández-Galisteo, A. L. Sánchez, A. Liñán, and F. A. Williams, *One-step reduced kinetics for lean hydrogen-air deflagration*, Combust. Flame, 156 (2009), pp. 985–996, http://dx.doi.org/10.1016/j.combustflame.2008.10.009.
[8] M. Garland and D. B. Kirk, *Understanding throughput-oriented architectures*, Comm. ACM, 53 (2010), pp. 58–66, http://dx.doi.org/10.1145/1839676.1839694.
[9] D. G. Goodwin, H. K. Moffat, and R. L. Speth, *Cantera: An Object-Oriented Software Toolkit for Chemical Kinetics, Thermodynamics, and Transport Processes*, Version 2.1.2, 2014.
[10] J. O. Hirschfelder, C. F. Curtiss, R. B. Bird, and M. G. Mayer, *Molecular Theory of Gases and Liquids*, Wiley, New York, 1954.
[11] R. J. Kee, G. Dixon-Lewis, and J. A. Miller, *Transport a Software Package for the Evaluation of Gas-Phase, Multicomponent Transport Properties*, manuscript, 1986.
[12] R. J. Kee, J. A. Miller, and T. H. Jefferson, *CHEMKIN: A General-Purpose, Problem-Independent, Transportable, FORTRAN Chemical Kinetics Code Package*, Technical report, Sandia National Laboratories, Livermore, CA, 1980.
[13] R. J. Kee, F. M. Rupley, and J. A. Miller, *Chemkin*-II: *A Fortran Chemical Kinetics Package for the Analysis of Gas-Phase Chemical Kinetics*, Technical report, Sandia National Laboratories, Livermore, CA, 1989.
[14] H. P. Le, J.-L. Cambier, and L. K. Cole, *GPU-based flow simulation with detailed chemical kinetics*, Comput. Phys. Commun., 184 (2013), pp. 596–606, http://dx.doi.org/10.1016/j.cpc.2012.10.013.
[15] J. Li, Z. Zhao, A. Kazakov, M. Chaos, F. L. Dryer, and J. J. Scire, *A comprehensive kinetic mechanism for CO, CH2O, and CH3OH combustion*, Int. J. Chem. Kinet., 39 (2007), pp. 109–136, http://dx.doi.org/10.1002/kin.20218.
[16] S. Mathura, P. Tondona, and S. Saxena, *Thermal conductivity of binary, ternary and quaternary mixtures of rare gases*, Molecular Phys., 12 (1967), pp. 569–579, http://dx.doi.org/10.1080/00268976700100731.
[17] B. J. Mcbride, S. Gordon, and M. A. Reno, *Coefficients for Calculating Thermodynamic and Transport Properties of Individual Species*, Technical Report NASA-TM-4513, 1993, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19940013151_1994013151.pdf.
[18] K. E. Niemeyer and C.-J. Sung, *Accelerating moderately stiff chemical kinetics in reactive-flow simulations using GPUs*, J. Comput. Phys., 256 (2013), pp. 854–871, http://dx.doi.org/10.1016/j.jcp.2013.09.025.
[19] K. E. Niemeyer and C.-J. Sung, *Recent progress and challenges in exploiting graphics processors in computational fluid dynamics*, J. Supercomput., 67 (2014), pp. 528–564.
[20] P. K. Notz, R. P. Pawlowski, and J. C. Sutherland, *Graph-based software design for managing complexity and enabling concurrency in multiphysics PDE software*, ACM Trans. Math. Software, 39 (2012), pp. 1–21, http://dx.doi.org/10.1145/2382585.2382586.
[21] NVIDIA, *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK*110/210, Technical report, NVIDIA, 2014.
[22] M. Ó Conaire, H. J. Curran, J. M. Simmie, W. J. Pitz, and C. K. Westbrook, *A comprehensive modeling study of hydrogen oxidation*, Int. J. Chem. Kinet., 36 (2004), pp. 603–622, http://dx.doi.org/10.1002/kin.20036.
[23] R. Sankaran, *GPU-accelerated software library for unsteady flamelet modeling of turbulent combustion with complex chemical kinetics*, in Proceedings of the 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, 2013, pp. 5599–5604, http://dx.doi.org/10.2514/6.2013-372.
[24] Y. Shi, W. H. Green, Jr., H.-W. Wong, and O. O. Oluwole, *Redesigning combustion mod-*

*eling algorithms for the Graphics Processing Unit (GPU): Chemical kinetic rate evaluation and ordinary differential equation integration*, Combust. Flame, 158 (2011), pp. 836–847, http://dx.doi.org/10.1016/j.combustflame.2011.01.024.

[25] G. P. Smith, D. M. Golden, M. Frenklach, N. W. Moriarty, B. Eiteneer, M. Goldenberg, C. T. Bowman, R. K. Hanson, S. Song, W. C. Gardiner, Jr., V. V. Lissianski, and Z. Qin, *GRI-Mech* 3.0, http://www.me.berkeley.edu/gri_mech/.

[26] K. Spafford, J. Meredith, J. Vetter, J. Chen, R. Grout, and R. Sankaran, *Accelerating S3D: A GPGPU case study*, in Euro-Par 2009–Parallel Processing Workshops, Springer, Berlin, Heidelberg, 2010, pp. 122–131.

[27] C. P. Stone, R. L. Davis, and B. Sekar, *Techniques for solving stiff chemical kinetics on GPUs*, in Proceedings of the 51st AIAA Aerospace Sciences Meeting, 2013.

[28] B. D. Taylor, D. A. Schwer, and A. Corrigan, *Implementation of Thermochemistry and Chemical Kinetics in a GPU-based CFD Code*, in Proceedings of the 53rd AIAA Aerospace Sciences Meeting, 2015.

[29] C. S. Yoo, T. Lu, J. H. Chen, and C. K. Law, *Direct numerical simulations of ignition of a lean n-heptane/air mixture with temperature inhomogeneities at constant volume: Parametric study*, Combust. Flame, 158 (2011), pp. 1727–1741, http://dx.doi.org/10.1016/j.combustflame.2011.01.025.

[30] P. Zhang and C. K. Law, *A fitting formula for pressure and temperature dependence of unimolecular reaction rate constants*, in Proceedings of the 6th U.S. National Combustion Meeting, 2009, http://dx.doi.org/10.1002/kin.20451.