12 S3D-Legion

An Exascale Software for Direct Numerical Simulation of Turbulent Combustion with Complex Multicomponent Chemistry

Sean Treichler, Michael Bauer, Ankit Bhagatwala, Giulio Borghesi, Ramanan Sankaran, Hemanth Kolla, Patrick S. McCormick, Elliott Slaughter, Wonchan Lee, Alex Aiken, and Jacqueline Chen

CONTENTS

12.1 INTRODUCTION

New methods are needed to explore novel fuels and design better engines that can substantially increase combustion efficiency, extend the longevity of finite fossil fuel reserves, and reduce carbon dioxide (CO_2) and other emissions. Government mandates to reduce petroleum use by 25% by 2020 and greenhouse gas emissions by 80% by 2050 are also exerting pressure on industry and will require significant retooling of all aspects of energy use in the United States. Achieving these aggressive goals requires the automotive industry to significantly shorten its design cycle. The transportation sector alone accounts for two-thirds of the nation's petroleum use and one-quarter of the nation's greenhouse gas emissions. Compounding these challenges, fuels are also evolving, adding another layer of complexity and further highlighting the need for rapid development cycles. We believe the optimal path to fast design cycles is through predictive modeling and simulation, enabled by recent advances in supercomputing.

In the past quarter century, the continual growth of high-performance computing (HPC) has had a major impact on the progress of science and engineering. It has accelerated the pace of research and development in many important fields including combustion science. For example, it has enabled new fundamental insights into turbulent combustion, a key subprocess that requires accurate modeling for the design of practical combustion devices. Turbulent combustion poses daunting modeling and computational challenges. The challenges arise due to (1) the large number of chemical species needed to describe a practical fuel (more than 100 species), (2) the large dynamic range of length and time scales (10^{-7}–10^{-2} m and 10^{-9}–10^{-2} seconds in an internal combustion engine), and (3) the coupling of highly nonlinear Arrhenius chemistry and large turbulent fluctuations of species concentrations with pressure and temperature.

12.1.1 Direct Numerical Simulation

To cope with these challenges, combustion scientists rely on a diverse set of simulation infrastructures and modeling techniques. The multiscale nature of combustion demands a hierarchy of computational fluid dynamics (CFD) methods ranging from fine-grain first principles direct numerical simulation (DNS) to coarse-grain large-eddy simulations (LES) and Reynolds averaged simulations (RANS). In particular, statistics from DNS simulations are used to bootstrap coarse-grain simulations and therefore DNS requires the most accuracy. In DNS, the instantaneous governing reactive Navier–Stokes, energy, and species continuity equations are solved without averaging or filtering; all relevant flow and combustion scales are resolved on the grid with accurate numerical methods.

Although DNS yields very high-fidelity results, it also makes simulations computationally expensive, especially because the Reynolds number scaling between the largest and smallest turbulent scales is proportional to $N^{9/4}$, where N is the number of grid points and practical devices operate at high Reynolds numbers for efficiency. A single simulation often takes longer than a month to complete, can consume millions of node hours on today's petascale supercomputers, and generates upward of a petabyte of raw data. To date, these costs have constrained scientists to perform simulations that are reduced in one or more ways:

- A reduction in the size of the physical volume being simulated decreases the number of grid points required, but limits the Reynolds number and the size of the turbulent structures that can be studied. It also limits the statistical sample size needed for model development and validation.
- Existing studies have focused on simpler fuel chemistries such as dimethyl ether (DME) [1]. Future fuels of interest for internal combustion engines are generally much more complicated. For example, the reduced primary reference fuel (PRF) mixture [2] we consider in this work requires over 4× more computation and storage resources than DME, even after reduction techniques have been applied [3].

- In studies that unavoidably require examining large structures with complicated chemistry, experimental diagnostics have been performed with reduced spatial dimensionality, using at most two-dimensional (2D) planar information in a probe volume [4,5]. Unfortunately, turbulence is intrinsically three-dimensional (3D), so these studies are unable to accurately capture the mixing effects of combustion in a turbulent environment.

12.1.2 Programming for HPC and Exascale

Simulation codes themselves are not steady-state either. A new DNS experiment often requires new functionality to be incorporated into an existing code while maintaining good performance characteristics. Within existing programming models, this process requires a good understanding of the underlying science, the intricacies of the existing source code, and the performance characteristics of the machine. Introducing new features mandates the optimization of both the new code as well any old code impacted by the change. Under such conditions, the programming effort and cognitive load placed on programmers explodes as the number of interacting features grows. The resulting complexity coupled with the time necessary to implement, test, and optimize code changes for new features can easily exceed the running time of the simulation itself.

The evolving architectures of modern supercomputers compound the problem of programmer productivity. As machines integrate heterogeneous processors such as graphics processing units (GPUs) and deep memory hierarchies, the programmer becomes responsible for deciding how to effectively *map* applications onto a target architecture. The process of mapping requires programmers to decide how computations are assigned to processors and how data are placed in caches and memories. Existing programming models force the programmer to implement these mapping decisions directly in the source code. This entangles the functionality of the code with its mapping in a way that inhibits either aspect of the code from being modified without a complete understanding of the other. Even worse, achieving performance portability across different architectures requires implementing two or more different mappings in the same source code, which is commonly done with a preponderance of conditional compilation directives and/or complete forks of the source tree.

12.1.3 Contributions

In this work, we report a dramatic improvement on the state of the art by presenting the first 3D DNS of the PRF mechanism that is able to resolve micrometer-scale turbulent flame structures. The simulation was run on two heterogeneous supercomputers, Piz Daint at CSCS [6] and Titan at ORNL [7], obtaining over 80% of the maximum achievable performance. Our code is based on a port of S3D [8] to the Legion programming model [9]. The port is significantly simplified by the ability of Legion to interoperate with existing message passing interface (MPI) applications—everything outside of the main simulation loop is left in its original MPI Fortran form. We leverage the novel mapping interface provided by the Legion runtime to easily tune our code for the PRF mechanism and map our code onto different architectures. The addition of the new PRF chemistry to our code was made effortless by the singe domain-specific language (DSL) compiler [10], which generates optimized Legion tasks for the main computational kernels.

We also present the DNS of a temporally evolving n-dodecane fuel jet surrounded on either side by air. For this simulation, the stencil tasks of the S3D-Legion code were enhanced to support stretched grids and to allow one-sided stencils at the domain boundaries. In addition, nonreflecting outflow boundary conditions (BCs) [11] were added as additional tasks. The BC implementation is based on the algorithm proposed by Sutherland and Kennedy [12] and can be readily extended to BCs for inflows and walls. Although this algorithm favors programmability over performance, most of the BC-related tasks operate on a small subset of the grid and their impact on the simulation time is negligible.

We believe that this work demonstrates a general approach and framework in which exascale computers can be productively programmed via the construction of performance-portable programs. By decoupling the specification of programs from their mapping using Legion, we make it easier to modify, tune, and port applications. The use of domain-specific compilers such as Singe, in conjunction with Legion, enables fast task implementations to be generated without requiring programmer input. These innovations greatly reduce both the time and effort required by human programmers to implement, maintain, and augment simulations. Furthermore, this approach often yields higher performance than hand-tuned codes. In our case, the use of Legion with a DSL compiler has made the 3D turbulent simulation of a complex PRF chemical mechanism, which is representative of other large hydrocarbon fuels, feasible for the first time.

12.2 S3D

S3D [8] is a massively parallel direct numerical solver to simulate turbulent combustion in canonical configurations and thereby gain fundamental insights into the physical and chemical interactions in turbulent reacting flows.

S3D solves the governing equations for reacting flows, namely the conservation equations for mass, momentum, total energy, and species written as

$$\frac{\partial}{\partial t}\rho = -\nabla_\beta \cdot (\rho u_\beta) \tag{12.1}$$

$$\frac{\partial}{\partial t}(\rho u_\alpha) = -\nabla_\beta \cdot (\rho u_\alpha u_\beta) + \nabla_\beta \cdot \boldsymbol{\tau}_{\beta\alpha} - \nabla_\alpha p \tag{12.2}$$

$$\frac{\partial}{\partial t}(\rho e) = -\nabla_\beta \cdot \left[u_\beta(\rho e_0 + p)\right] + \nabla_\beta \cdot (\tau_{\beta\alpha} \cdot u_\alpha) - \nabla_\beta \cdot \mathbf{q}_\beta \tag{12.3}$$

$$\frac{\partial}{\partial t}(\rho Y_i) = -\nabla_\beta \cdot (\rho Y_i u_\beta) - \nabla_\beta \cdot (\rho Y_i V_{\beta i}) + W_i \dot{\omega}_i \tag{12.4}$$

where the indices α and β indicate spatial dimensions (with repeated indices denoting summation) and i indicates the species index. Additional terms include the mass density (ρ), the mass fraction and molecular weights of each species (Y_i, W_i), the fluid velocity (u), the total energy (e), and pressure (p).

Species-specific heat capacities and enthalpies are needed to compute the thermodynamic relations between the internal energy and temperature. S3D uses NASA thermodynamic polynomials with seven coefficients and two temperature ranges for computing the detailed species thermodynamic properties.

$\dot{\omega}_i$ is the molar production rate of species due to chemical reaction and is computed using a detailed chemical kinetics model of the chemistry being simulated [13]. The complexity of these models depends on the number of species and reactions, which varies considerably between mechanisms, as shown in Table 12.1. For all but the simplest mechanisms, the computation of these production rates is the most important and computationally intensive kernel in S3D.

The next major computational kernels are in the evaluation of the diffusive transport terms: the stress tensor ($\boldsymbol{\tau}$), thermal diffusive flux (q), and species mass diffusion velocities (V_i). These quantities are computed from the known conserved variables using constitutive relations that require continuum models for molecular transport properties such as viscosity, thermal conductivity, multicomponent species diffusivity, and thermal diffusivity. S3D uses detailed multi-component and mixture-averaged models for the transport properties [14] that are also dependent on density, temperature, and local mixture composition.

TABLE 12.1
Summary of S3D Chemical Mechanisms

Mechanism	Reactions	Species	Unique	QSSA	Stiff
H2	15	9	9	–	–
DME	175	39	30	9	22
Dodecane	268	53	35	18	22
Heptane	283	68	52	16	27
PRF	861	171	116	55	93

*DME, dimethyl ether; PRF, primary reference fuel; QSSA, quasi-steady-state approximation.

The final significant computation kernel is used in the gradient operators (∇_β), which use a 9-point centered stencil to compute an explicit 8th order finite difference approximation of the partial derivative in direction β. Although not captured in the equations above, the solution is filtered periodically using an explicit 10th order finite difference filter on an 11-point stencil for numerical stability [15]. Navier-Stokes characteristic boundary conditions (NSCBC) [11,12] were used to prescribe the outflow BCs for the temporal jet simulation.

The solution is advanced through time integration using an explicit fourth order six-stage Runge–Kutta scheme [16] with built-in error estimates. A single time step, therefore, consists of six evaluations of the right-hand-side function (rhsf), each followed by three scaled accumulations for integration.

12.2.1 Fortran+MPI Implementation

S3D is a complex piece of software that has evolved over the course of 30 years, during which time it has been worked on by a number of different scientists and engineers. The original version of S3D consists of approximately 200 K lines of Fortran and uses MPI [17] for communication between threads.

Parallelism is achieved by running a separate MPI rank on each core and trusting the Fortran compiler to vectorize loops to take advantage of the wider datapaths in today's central processing units (CPUs). This approach works well at smaller node counts and the simplicity of the source code is appreciated by domain scientists when they wish to add features to the code base.

However, the limitations of this initial approach become evident at scale (see Figure 12.1), as well as on machines with heterogeneous computational resources where there is no way for the pure Fortran version to target the accelerators.

12.2.2 Hybrid OpenACC+MPI Implementation

A second version of S3D targets heterogeneous systems by combining MPI with OpenACC [18] directives in a hybrid implementation [19]. This version was ported from the Fortran+MPI version by a team of scientists and engineers from Cray, ORNL, NREL, and NVIDIA. By leveraging both CPUs and GPUs, the MPI/OpenACC version of S3D is roughly two times faster than the Fortran/MPI version for the heptane mechanism with 52 species, between a third and a half of the complexity of the PRF mechanism used in this study. This implementation runs with a single MPI rank per node and uses OpenMP to recover the parallelism available from the multiple CPU cores on each node.

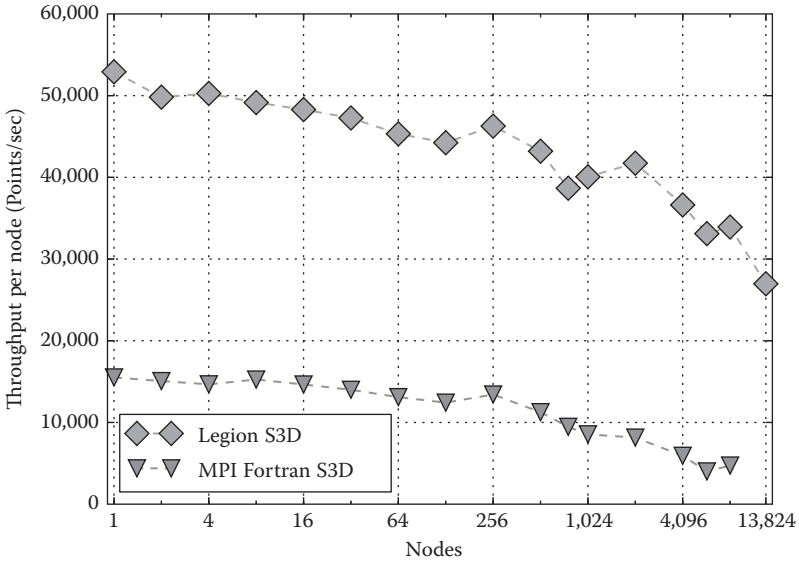

FIGURE 12.1 Weak scaling of PRF on Titan.

The effort to port the original Fortran version to the hybrid OpenACC model was significant. Over 2500 lines of directives were added to the code, each representing a decision by a person familiar with both the application and the then-current hardware of whether a given loop body should be offloaded to the GPU and which arrays must be copied from CPU memory to GPU memory or vice-versa. Because the directives apply to lexical constructs (specifically Fortran `do` loops), a large number of loops were manually fused (or occasionally split) and entire function bodies were inlined into loops when the compiler could not (or would not) perform the necessary inlining automatically.

The OpenACC implementation of S3D assumes a system with a single GPU per node. Systems with multiple GPUs per node are already common (e.g., Keeneland [20]), and although the OpenACC 2.0 standard includes support for multiple accelerators, additional directives would be required for all offloaded loops. Furthermore, when newer GPUs are introduced with different performance characteristics, the original decisions of which loops to fuse or split would need to be revisited. Any changes would be structural modifications to the source code, requiring conditional compilation directives to maintain support for versions targeted at different machines.

12.2.3 Legion Implementation

The most recent version of S3D is an implementation in the Legion programming model [21]. The Legion version runs inside of the original Fortran+MPI version, offloading the main simulation loop but using the existing Fortran code for everything else. The changes to the Fortran source consist of a few calls to initialize the Legion runtime and the replacement of the Fortran `integrates` call with one call into Legion code, all controlled by a single build option.

The Legion code consists of approximately 25,000 lines of C++ and CUDA source, representing several person–months of effort. All of the mechanism-specific codes are generated by the Singe DSL compiler (discussed in detail in Section 12.4). Approximately 1000 lines of the C++ source are in a custom *mapper* class that allows the same Legion application code to run efficiently on systems with different types and ratios of CPUs and GPUs, different memory hierarchies, and different interconnect networks.

12.3 LEGION

Legion is a task-based parallel programming model designed for high-performance computing on systems with heterogeneous computational resources and deep memory hierarchies [21]. Like other dynamically scheduled task-based execution models [22–24], Legion maintains a *task graph* in which the nodes are *tasks* to be executed and edges are ordering constraints between the tasks. The Legion runtime executes the tasks in the graph, guaranteeing that a given task does not start until all of its predecessors in the graph have been completed. The Legion programming model differs from other task-based models in the combination of two significant features that are visible to the programmer.

First, the programmer does not directly construct the task graph. The programmer instead declares what data each task uses and how each task uses it (e.g., reading or writing). The Legion runtime uses this information to infer the necessary ordering constraints. Legion provides a *logical region* abstraction that allows precise specification of the data being used by a task. Logical regions may be partitioned into *subregions*, and the runtime tracks subregion relationships, allowing it to correctly and efficiently detect when two tasks may access the same data. Importantly, logical regions provide a data model that decouples the description of data from how it is placed and laid out in the memory hierarchy.

Second, the Legion application code does not specify on what processors tasks run or where in the memory hierarchy logical regions are placed, neither does the runtime attempt to automatically make these decisions. Instead, these decisions are made in a separate *mapper* object that encapsulates all machine-specific decisions about program execution. The mapper is part of the application code and implements an interface that responds to queries from the Legion runtime. When a task t is ready to be mapped, the runtime provides information about the current locations of *physical instances* of logical regions needed by t. Using this information, the mapper responds with the processor on which t should run and where t's physical instances should be placed. If the requested locations of physical instances differ from the current locations, the runtime automatically adds the necessary data movement tasks and dependencies to the task graph to ensure that t executes only after its data have arrived.

A key property of Legion programs is that changing the mapping cannot change the program's input–output behavior; the Legion runtime understands the semantics of task dependencies and guarantees that every task executes on the right data, regardless of the mapping. Thus, Legion programs truly constitute a machine-independent specification of an application. This property of Legion is crucial to achieving high productivity: unlike other task-based models and bulk synchronous models such as MPI, task and data placement decisions are not baked directly into a Legion program. Mapping decisions that impact performance are completely isolated within mapper objects. In a Legion application, it is easy to change the mapping to tune or port for new machines without compromising correctness.

Like other dynamic runtime systems, Legion incurs runtime overhead to compute and manage the task graph. The Legion runtime is highly optimized to keep this overhead off of the critical path of execution by performing much of the dynamic analysis in parallel with the rest of the application [25]. However, any dynamic runtime has a minimum granularity of useful work that it can efficiently support. For Legion, we find that the analysis overhead can be hidden if tasks take at least 500 μsec to execute on average. Many of S3D's operations are below this threshold, and exploiting fine-grain vector parallelism within Legion tasks requires an alternative static scheduling approach.

12.4 SINGE

As mentioned in Section 12.2.3, the mechanism-specific kernels (the reaction rate and transport coefficient calculations) are both the most complicated and the most performance-critical kernels. Tuning these kernels is essential for improving time to solution, but their size and complexity exceed the

optimization capabilities of general purpose compilers. Hand optimization can yield significant benefits for an individual mechanism, but the immense time investment must be repeated for each new mechanism.

Our solution is Singe, a domain-specific language compiler for combustion chemistry kernels [10]. Singe takes as input a standard description of a chemical mechanism along with tables describing the transport and thermodynamic properties of chemical species. From these specifications, Singe synthesizes high-performance kernels for use in combustion simulations. Singe leverages domain-specific optimizations, such as the quasi-steady-state approximation (QSSA) [26], which groups similar reactions into a single computation, and special handling of *stiff* species [27], permitting larger time steps.* It also applies advanced compiler optimizations such as warp specialization for GPUs [28]. Singe then emits highly specialized variants of kernels for different architectures. Code is specialized differently for the Fermi and Kepler GPU architectures, as well as for CPUs with different cache sizes and vector instruction sets (e.g., streaming SIMD extensions [SSE] vs. advanced vector extensions [AVX]. SIMD is single instruction, multiple data [SIMD]). All of these kernel variants are registered with the Legion runtime and made available to the mapper, allowing it the runtime option of mapping tasks to a CPU or a GPU.

The performance of kernels generated by Singe is excellent. In cases where the OpenACC compiler generated GPU code for these kernels, the Singe versions are two to five times faster, with the largest speedups coming from sophisticated mechanisms, such as PRF, that have complicated computations and huge working sets. However, the productivity improvement is even more dramatic. Singe comes equipped with its own autotuning framework [10], which can optimize all the required kernels for a new chemical mechanism in less than 24 hours without any human intervention or direction. Instead of spending days or weeks hand-optimizing individual kernels that are thousands of lines long, adding support for a new mechanism is now an automated process that frees combustion scientists to engage in other useful work.

12.5 SIMULATION SETUP

The S3D-Legion code was used to simulate two separate scientific problems that are described below. The first scientific problem was the simulation of reactivity controlled compression ignition (RCCI) that was performed using all periodic boundaries and therefore did not require any special BC. The second scientific problem simulated using the S3D-Legion is the temporal evolution of a nonpremixed jet mixing layer, which required the application of outflow BCs and nonuniform mesh spacing through algebraic stretching. The setup of the simulation experiments for these two problems is presented below.

12.5.1 RCCI Simulation

The first simulation case is part of a larger parametric study of the impact of turbulence on combustion under RCCI conditions [29]. In RCCI, a mixture of fuels is used—iso-octane (i-C_8H_{18}) and n-heptane (n-C_7H_{16}) in the case of PRF—to obtain reliable ignition while operating in a regime that minimizes both particulate and NO_x emissions and still provides very high (i.e., diesel-like) thermal efficiency. Combustion phasing (ignition timing with respect to piston motion) and subsequent heat release rate are strongly sensitive to turbulence—the uneven mixing due to turbulence causes behavior to deviate significantly from that of a perfectly homogeneous mixture. Thus, it is essential to be able to couple turbulent simulations with detailed chemistry.

The physical volume of the DNS was chosen to be a cube 3 mm on a side, with the expectation that a grid spacing of 2.6 μm would be sufficient to resolve all the flame structures that were generated.

* The number of QSSA and stiff species in PRF and the other chemical mechanisms we study are listed in Table 12.1.

This results in a grid size of 1152^3 (just over 1.5 billion grid points) with 123 conserved variables per grid point (including two additional transported variables for model assessment discussed later in this section). With guidance from 1D flame simulations on the size of PRF flame structures under these conditions, this choice was conservative; we gave ourselves enough margin to refine the grid spacing to as small as 2.1 μm (using nearly 3 billion grid points) if necessary. Our run needed to simulate 3 ms of physical time using 2.5 ns time steps, requiring 1.2 million time steps to observe interesting combustion effects.

As described earlier, the flame structures that demand such high spatial resolution do not appear until well into the simulation at a time when the first ignition kernel undergoes thermal explosion and forms a flame, so the simulation was initially started with a grid size of 576^3. The factor of 2 increase in grid spacing also permitted a factor of 2 increase to 5 ns per time step for the first 2.7 ms of the run, improving the time to solution. The computational resource usage is improved by the 2× reduction in grid points in each dimensions and the reduced number of time steps required for an overall reduction factor of 16 for the majority of our run.

S3D is formulated as a constant-volume simulation, and models compression of the cylinder with an additional source term in each of the governing equations that adds mass in a way that does not cause spurious pressure fluctuations [30,31]. To maximize the number of turbulent eddies in the simulation domain, the simulation was set up with periodic BCs, which can cause the fraction of volume associated with heat release to be higher than under realistic engine conditions. To address this issue, the mass injection strategy was extended to not only account for volume changes due to piston motion, but to also replicate the pressure history of a fired engine. This required the introduction of a global reduction (to compute the average rate of change of pressure over the entire volume) to be incorporated into an application whose communication pattern previously consisted entirely of nearest-neighbor exchanges.

These changes were initially implemented in the Fortran+MPI version of S3D by domain scientists. As they all impact the main simulation loop, they had to be ported to the Legion application code. The existing Fortran code served as an excellent reference, as it succinctly described the necessary functionality without being obfuscated by directives and/or manual code transformations for optimization reasons. Because Legion tasks are also unencumbered by mapping considerations, the porting was a simple matter of translating Fortran code into the corresponding C++. The porting (and verification) effort required less than a programmer-week of work. Once the desired functionality was achieved, adding mapping policies for the new tasks required less than 10 lines of code in the Legion S3D mapper and was tuned in less than an hour.

For this simulation, we also explored the use of the OpenACC version of S3D for performance comparisons. This required the OpenACC code to be updated for both the PRF mechanisms and the additional features described above. We estimated that at least another month of implementation work as well as an unknown amount of debugging and tuning would be required to reach a production version of the OpenACC code. Our previous performance results demonstrated that the Legion version outperforms the OpenACC version by 2–3× on the smaller DME and heptane mechanisms [9]. This led us to abandon the effort and collect performance data for the MPI and Legion version of S3D only. We performed the RCCI production simulation entirely using the Legion version of S3D.

12.5.2 Temporal Jet Simulation

The second simulation case addressed ignition and flame propagation in a temporally evolving air/n-dodecane jet at pressure (25 bar) and temperature ($T_{air} = 960$ K) conditions that are relevant to low-temperature diesel engine combustion. The scope of this simulation was to investigate the effect of low-temperature chemistry on the ignition of the jet and to better understand how burning kernels develop into flames propagating in a non-homogeneous mixture.

The physical volume of the DNS was a cuboid of 3.6 mm × 14 mm × 3 mm in the x, y, and z directions. The computational grid was uniform and periodic in the x and z directions and stretched along the inhomogeneous y direction, with a fine mesh region that spanned a distance equal to 4.0 mm at the beginning of the simulation to 6.6 mm at the end of the simulation. The grid spacing in the periodic directions and in the fine mesh region was 2.9 micron. The grid size at the end of the simulation was $1216 \times 2400 \times 1024$ (3 billion grid nodes). The chemical mechanism used in the simulation was the 35-species reduced n-dodecane one listed in Table 12.1 and the number of conserved variables per grid point was 42. The variable count includes two auxiliary transported variables, a mixture fraction and a cumulative scalar dissipation rate, which were used to better understand the physics of the simulated problem. The run needed to simulate 1 ms of physical time using 4 ns time steps to observe mixture ignition and the propagation of burning flames throughout the simulation domain.

The code used for the simulation described in 12.5.1 supported neither stretched grids nor open BCs. To overcome these limitations, we rewrote the existing stencil tasks to support stretched grids and one-sided stencils at the domain boundaries. We also implemented the necessary tasks to simulate problems in which nonreflecting outflow BCs [11] are used. Our BCs implementation is based on the algorithm proposed by Sutherland and Kennedy [12] and can be readily extended to other types of BCs, such as walls and inlets. Most of the boundary-related tasks operate on a small subsets of the grid associated with each computational node and their impact on the simulation time is negligible (less than 1%).

The jet was initialized as a laminar jet of half-width equal to 0.25 mm with superposed turbulent fluctuations. Chemical reactions were not activated until $t = 0.28$ ms, when the jet had become fully turbulent. Early during the flow development, the jet width remained small and a computational grid with a fine mesh region of 4.0 mm was used to save computational resources. We used the full three billion nodes grid only after $t = 0.65$ ms, shortly before the jet extended beyond the refined region of the starting computational grid.

Initially, the simulation was run on Titan using the MPI version of S3D because at that time support for BCs and nonuniform grids was not implemented yet within the Legion version of the code. We eventually adopted the modified Legion version when the simulation reached $t = 0.45$ ms. The per node problem size was 64^3 grid points for the smallest computational grid used during the simulation and $64 \times 96 \times 64$ grid points for the largest one. Both these problem sizes allowed an all-GPU mapping of the simulation tasks, which turned out to be the optimal mapping strategy on Titan for problems using the n-dodecane chemistry. Due to the lack of a ready-to-use n-dodecane mechanism for the OpenACC version of S3D, no attempt was made in running this DNS simulation with that code.

12.6 COMBUSTION RESULTS

12.6.1 RCCI Simulation Results

Figure 12.2 shows the temporal evolution of normalized domain average species mass fractions of several key species along with temperature and heat release for the 3D simulation. It can be seen that n-heptane (n-C_7H_{16}) is consumed significantly earlier in the cycle than iso-octane. After the low-temperature heat release stage marked by the first peak in the heat release rate (HRR in Figure 12.2) profile, the original n-heptane is almost entirely consumed, most of it having decomposed to CH_2O, C_2H_4, and other smaller molecules. Consumption of CH_2O and production of OH appear to coincide with the consumption of all remaining iso-octane (i-C_8H_{18}). This period also coincides with the oxidation of CO. The generation of intermediate species such as CH_2O and CO occurs primar-

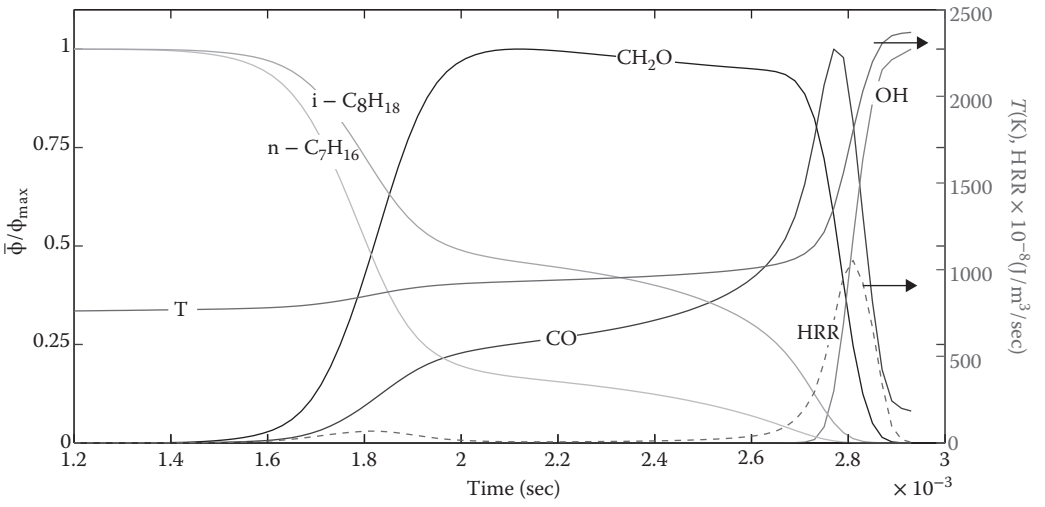

FIGURE 12.2 Temporal evolution of normalized species concentrations (left axis) and temperature and heat release rate (right axis) for the three-dimensional reactivity controlled compression ignition simulation. The overbar ($\bar{.}$) denotes an average over the domain. The dashed line represents heat release rate.

ily through the breakdown of n-heptane. Because most of the heat released later in the simulation (i.e., the high-temperature heat release) is driven by the oxidation of these intermediates, it follows that combustion is driven by the staged consumption and oxidation of n-heptane and its intermediates followed by a rapid decomposition and oxidation of iso-octane.

Compression ignition configurations generally ignite by generating a series of ignition fronts. One of the most interesting aspects of the RCCI configuration is the appearance of flame fronts in conjunction with spontaneous ignition fronts. Ignition fronts are completely reaction driven: individual locations in the domain react and spontaneously ignite independently of their neighbors, down a gradient of ignition delay imposed by spatial variations in reactivity, temperature, or composition. In contrast, a flame front is diffusion driven: individual locations in the domain react and propagate only when heat and reactants diffuse into them from neighboring locations. Visually, flame fronts appear as thin, spindly structures, whereas ignition fronts appear as thick, blob-like structures. Figure 12.3 shows the overall heat release rate in the simulation volume at the time corresponding to approximately 50% of the total heat release. Figure 12.4 shows slices of the heat release rate in the simulation domain taken at one of the midplanes. The three images correspond to time instances at which 30%, 50%, and 80% of the total heat release has taken place. Most of the heat release occurs through the thin flame fronts.

One of the methods for quantifying whether the mode of combustion in a given location is through ignition or flames is to compute a reaction-diffusion balance across the burning surface. Figure 12.5 shows a volume rendering of these surfaces at a time instance corresponding to approximately 50% of the total combustion heat release. It can be seen that diffusion and reaction track each other quite closely in space, though they are not collocated. Moreover, their magnitudes are roughly equal throughout the domain. This suggests that most of the combustion in the simulation occurs through the flame propagation mode. Figure 12.6 shows profiles of the diffusion rate and reaction rate of one of the key intermediate species, OH, at the same time instant. The profile on the left shows that the peak magnitudes of both reaction and diffusion are comparable, which is a key characteristic of premixed flames. The profile on the right shows a *spontaneous ignition front*, which is dominated by reaction and hence the magnitude of the diffusion rate is negligible compared to that of the reaction rate.

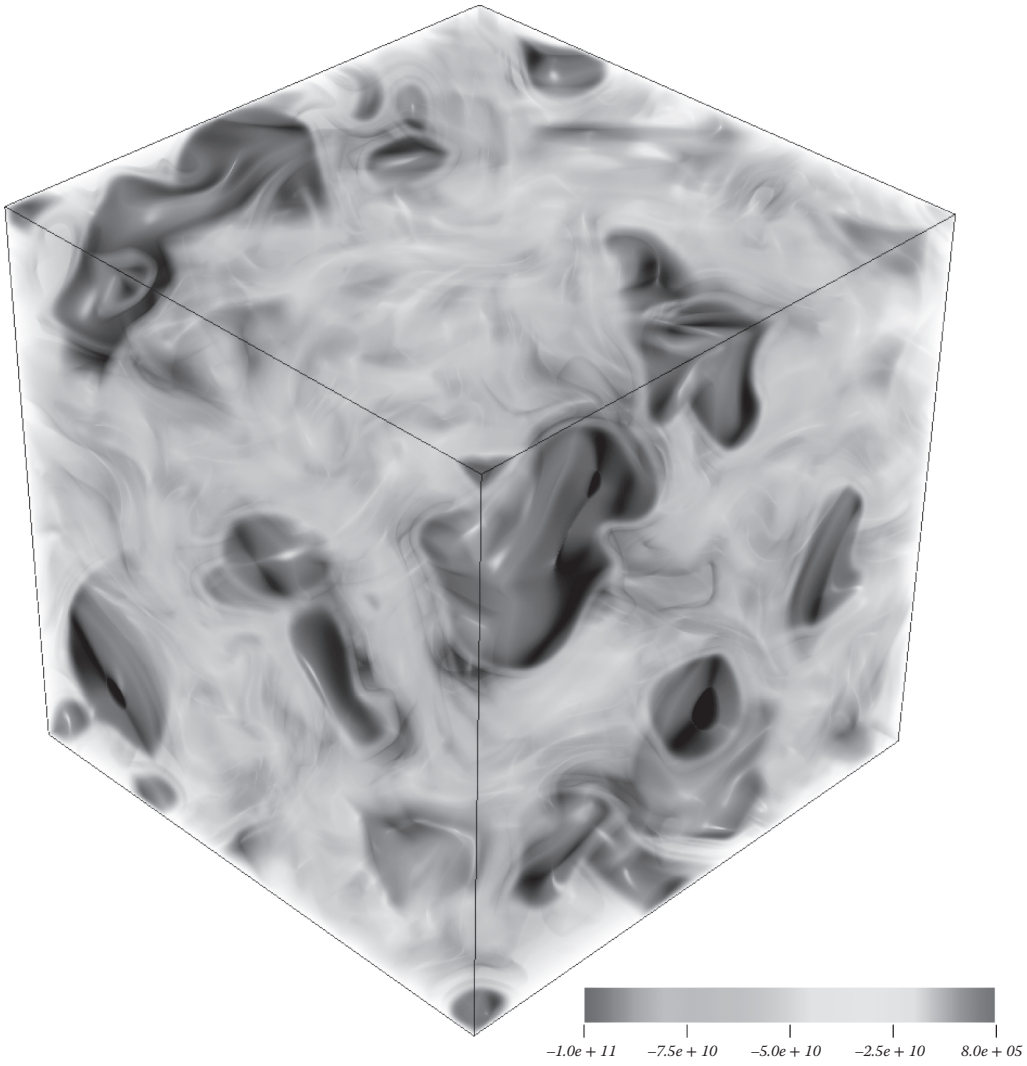

FIGURE 12.3 (See color insert.) Volume rendering of the heat release rate at the time corresponding to 50% of total heat release. Values are in J/m^3/sec.

The PRF mixture used in this study had a large fraction of n-heptane. Although both flames and ignition fronts are found to coexist in this simulation, more than 80% of the combustion heat release occurs through the flame propagation mode. This observation appears to be unique to fuel blends that contain high reactivity fuels, such as n-heptane. This has broad implications for modeling of RCCI combustion and this 3D dataset provides a unique benchmark for the development and validation of models applicable to an engine operating in the mixed combustion modes under RCCI conditions. Optical engine experiments provide complementary statistics but have insufficient resolution to discern flames from spontaneous ignition fronts. This is the first study of the details of the underlying flame and ignition structure with fuel blending. The results suggest that by conducting future DNS simulations, in conjunction with experiments and LES, it will be possible to find the optimal fuel blend for different engine operating conditions.

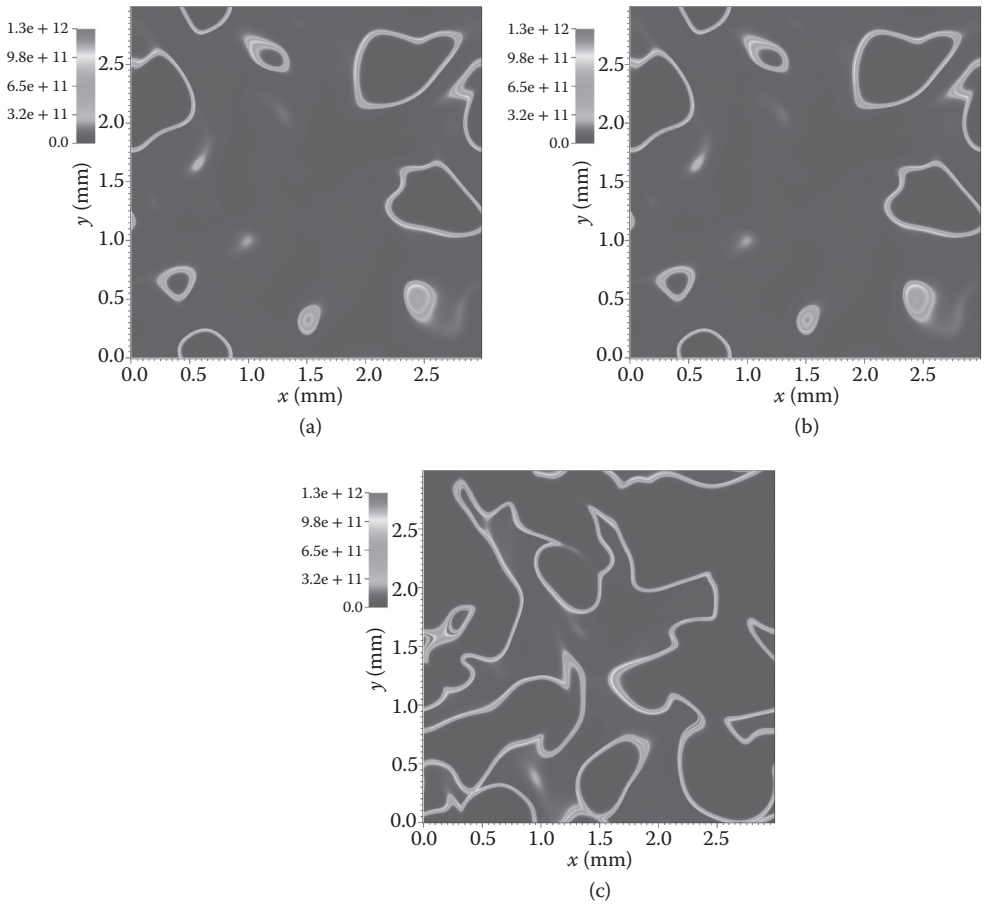

FIGURE 12.4 Slices taken at the midplane of the simulation volume showing contours of heat release rate at time instances corresponding to 30% (a), 50% (b), and 80% (c) of the total heat release. Values are in J/m^3/sec.

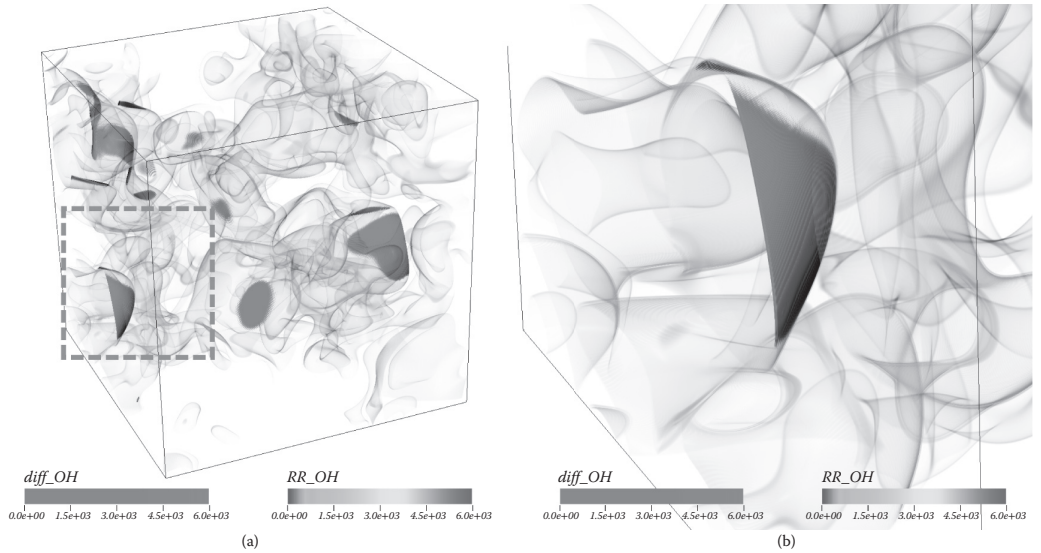

FIGURE 12.5 (See color insert.) Contours of diffusion rate and reaction rate of the OH radical for the full simulation volume (a) and a zoomed in view of the same in the boxed region near a flame kernel (b).

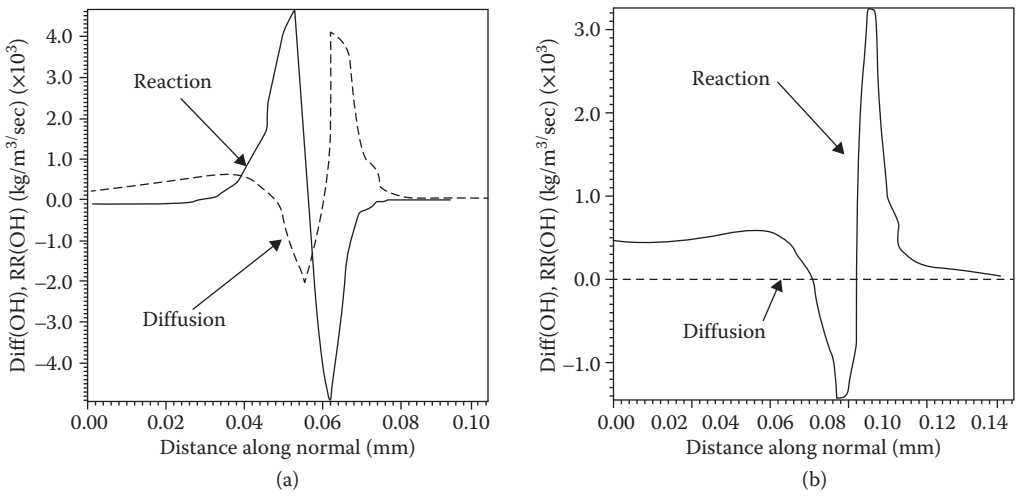

FIGURE 12.6 Comparison of the diffusion rate and reaction rate of the OH radical along a flame surface (a) and an ignition front (b).

12.6.2 Temporal Jet Simulation Results

Understanding the physics of ignition is very important for achieving high efficiency and low pollutant emissions in modern diesel engines. Generally speaking, ignition in a nonpremixed flow is the result of a competition over time between heat production due to chemical reactions and heat dissipation due to transport processes. Under turbulent conditions, the probability is small that two flow elements will be subjected to the same reacting and mixing histories and, as such, we should expect that different flow regions with the same composition will ignite at different times. Our simulation reveals this fact to hold true for both low- and high-temperature ignition.

Characterizing the topology of the low- and high-temperature ignition kernels is very important to optimize the mixture preparation phase in compression-ignition engines. Figure 12.7 shows the scatterplots of temperature versus mixture fraction (ξ) at selected instants of time during the ignition transient. It can be seen that the turbulent jet ignites first in those regions where $\xi = 0.06$ and that ignition is followed by the rapid propagation of the burning kernels toward regions with fuel-rich compositions. During this phase, most of the heat release rate is due to low-temperature reactions and therefore the maximum increase in gas temperature remains below 400 K. This increase in temperature, albeit modest, is sufficient to initiate the second phase of the ignition transient by triggering the high-temperature reactions pathway of n-dodecane, which in turn initiates the hot flame ignition process. High-temperature regions are seen developing first around $\xi = 0.16$: their subsequent ignition results in several reaction fronts that propagate toward the flow regions with leaner mixture compositions, leading eventually to the establishment of fully burning conditions throughout the entire flow.

In this study, low-temperature reactions play a key role in triggering autoignition by raising the gas temperature just enough to activate the high-temperature reactions pathways. As their name suggests, low-temperature reactions are non-negligible only when the ambient gas temperature is below a fuel-dependent threshold, which is approximately $T_g = 1000$ K for n-dodecane. Diesel engines used to be operated above this threshold value to maximize their efficiencies: as such, low-temperature combustion phenomena received little or no attention in most of the existing DNS studies on turbulent autoignition. The situation has changed in recent years due to the emergence of the low-temperature diesel combustion concept as a promising engine technology for abating pollutants emissions while maintaining the high efficiency typical of traditional diesel engines. Unfortunately, investigating tur-

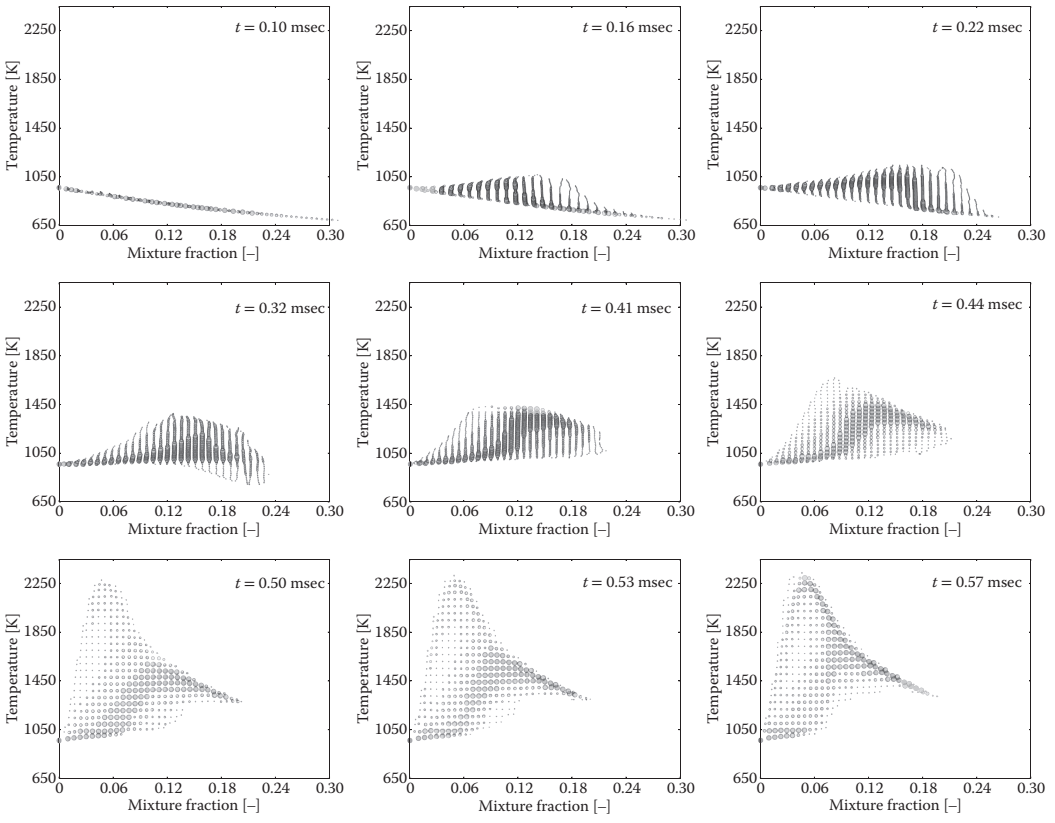

FIGURE 12.7 Scatterplots of temperature versus mixture fraction at selected instants of time during the ignition and transient. Shading and size of the symbols are indicative of the number of samples in the computational domain with that composition and temperature.

bulent flow autoignition with DNS at conditions representative of low-temperature diesel combustion has proven to be an extreme challenge due to the combination of long induction times, small length scales, and large sizes of the chemical mechanisms required to capture the low-temperature reaction pathways. This study constitutes a pioneering work that will shed light on the combustion physics behind the next generations of clean, efficient engines.

12.7 PERFORMANCE

This section describes the performance characteristics of the RCCI science case described in Section 12.5.1. Portions of the RCCI simulations were run on two different Cray machines: Titan at the Oak Ridge Leadership Computing Facility (currently second on the Top500 list [32]) and Piz Daint at the Swiss National Supercomputing Center (sixth in the Top500). Both systems rely on a Kepler K20X GPU per node for the bulk of their computational power. Titan [7] has 18,688 nodes, pairing the GPU with a 16-core Opteron CPU and a Gemini interconnect. Piz Daint [6] is a newer, but smaller, machine, consisting of 5272 nodes each with an 8-core Sandy Bridge-EP CPU and a Cray Aries interconnect.

12.7.1 Performance Bottlenecks

With roughly 90% of the available floating point operations per second (FLOPS) on the GPU, the first priority is to make sure the GPU is fully utilized. Figure 12.8 breaks down the GPU utilization on Titan by kernel, showing that the GPU is kept busy over 95% of the time. Due to all the internode communication for the gradient operations, every time step requires the movement of a large amount of data over the PCI-Express bus that links the CPU and GPU: 1.6 GB is moved from CPU to GPU and 4.0 GB is moved back. The bus is bidirectional and can transfer approximately 6 GB/sec in each direction; the CPU to GPU channel operates at 15% capacity, whereas the return link operates at 38% capacity. The Legion runtime automatically overlaps data transfers with kernels running on the GPU, requiring no effort from the programmer.

Although ensuring that the GPU is nearly always busy is important, it is equally important that the individual tasks make efficient use of the GPU. The roofline analysis used by Rossinelli et al. [33] remains an excellent tool for this purpose. We measured the achievable memory bandwidth of the K20X to be 159.6 GB/sec and the peak computational throughput to be 1121.9 GFLOPS. The computational intensity of the top kernels from Figure 12.8 and their performance are shown in comparison to the roofline in Figure 12.9. The two key kernels (`getrates` and `stencil`) are both memory bound. The `getrates` kernel is benefiting slightly from L1 cache hits for its register spills, which is why it appears slightly above the roofline. The Singe-generated `diffusion` and `viscosity` kernels are able to move their working sets into CUDA shared memory, eliminating the external memory bottleneck (their arithmetic intensities would be reduced to 1.4 and 2.8 FLOPS/B without this optimization), but are still unable to fully utilize the arithmetic potential of the GPU because they saturate the available shared memory bandwidth. Finally, the **`temp`** and `ydiffflux` kernels must iterate over all 116 species of the PRF mechanism, resulting in less efficient global memory access patterns and reduced memory system performance. By considering each kernel's

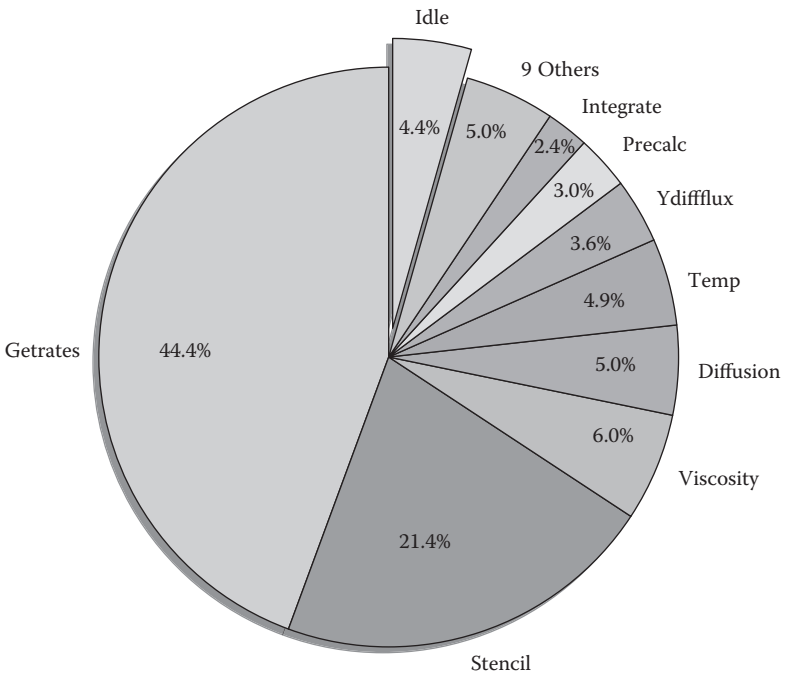

FIGURE 12.8 GPU usage by task on Titan.

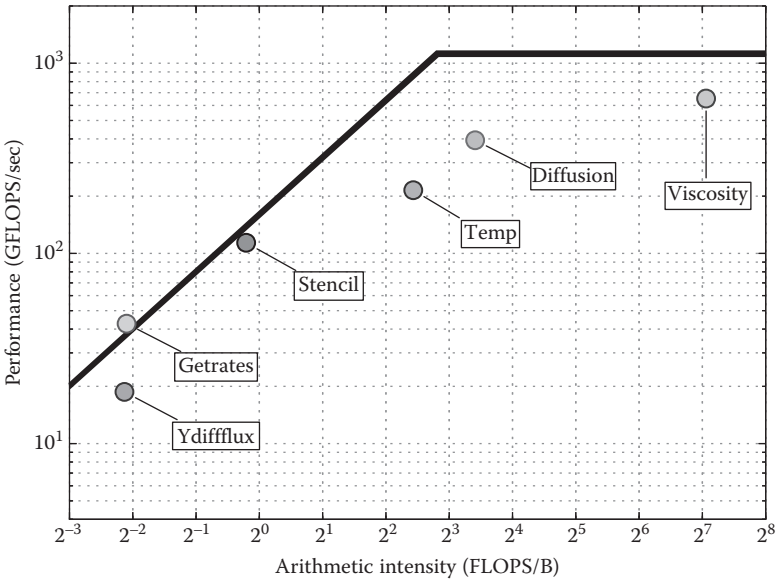

FIGURE 12.9 Roofline analysis of key GPU kernels.

distance from the roofline and weighting by the average fraction of a time step spent in a kernel, we determined that we are obtaining 80.4% of the achievable performance of the GPU.

Prior work on CFD [33] has shown that cache blocking on CPUs can be used to significantly improve the computational intensities of important kernels. Unfortunately, these techniques are not available to us for two reasons. First, although the caches and shared memories on the Kepler K20X are similar in capacity to those of modern CPUs, they service an order of magnitude more concurrent threads that considerably reduces the space available to each thread. Second, the introduction of complex chemistry to the system dramatically increases the working set of each thread. The computation of the right hand side for a single grid point in the PRF mechanism requires approximately 2000 double-precision intermediate temporaries. The L1 cache on a modern CPU would only be able to fit a handful of grid points, and even all the combined L2 caches in a CPU socket could support only a few hundred—not nearly enough to compensate for the edge effects (e.g., warming up prefetch engines and fetching stencil neighbors) in a cache-blocked formulation. Additionally, the chemistry, transport, and enthalpy calculations make heavy use of constants for polynomial coefficients. For PRF, there are over 34,000 of these constants, increasing the working set by a further 272 KB.

The viscosity and diffusion kernels achieve the best computational intensities because Singe is able to fit their working sets into on-chip memories. To handle large working sets, Singe uses a technique called warp specialization to extract task parallelism from a computation and distribute it across the warps of a thread block [28]. For smaller mechanisms such as DME and heptane, Singe has been able to use the same technique for the chemistry kernel [10]. However, for the PRF mechanism, the chemistry kernel working set consists of all the forward and backward reaction rates (1722 doubles per grid point), which exceeds the available on-chip memory and the resulting code is limited by external memory bandwidth due to register spills.

Our discussion has focused on the Kepler GPU because nearly all tasks related to the main computation are placed there by the Legion S3D mapper. Only a few tasks related to inter-node communication or interfacing with the Fortran application are performed on the CPU. By changing only a few lines of code in the mapper, we explored many alternative mappings that use the computational resources of the CPUs as well. Although we have found mapping strategies that use the CPUs to be effective on other machines with other chemical mechanisms [9], the increased size of the PRF mech-

anism results in the need to move much more intermediate data between the CPU and GPU. In all of the proposed mappings for the PRF mechanism, the PCI-Express bus became the bottleneck and GPU utilization dropped. Although the CPU utilization increased, overall performance was reduced.

12.7.2 Weak Scaling

We next measured weak scaling performance on both Titan and Piz Daint. We held the per node problem size constant at 48^3 grid points, the size on which we had based our simulation plan. The original Fortran+MPI version of S3D was run as a reference. The throughput achieved per node (higher is better) is shown for Titan and Piz Daint in Figures 12.1 and 12.10 respectively. Perfect weak scaling would be a flat line.

In addition to being significantly faster than the Fortran baseline (3.95× faster on Piz Daint and 3.93× faster on Titan for a single node), the Legion version demonstrates greatly improved scalability. Whereas the Fortran version drops below 50% efficiency beyond 2048 nodes on Titan,* the Legion version is above 64% even out to 8192 nodes, where the performance difference has grown to a factor of 7.2×. The asynchronous, task-based nature of the Legion programming model allows the Legion runtime to dynamically discover considerable task parallelism and use it to hide communication latency better than the MPI version that relies on manual overlap of communication and computation. The improvement from the Gemini to the Aries interconnect is evident in the Piz Daint data, with both the Fortran and Legion versions achieving parallel efficiencies of 82% at 4096 nodes. Note that the Legion version must reduce network overhead by a factor of 4 compared to the Fortran version to achieve the equivalent parallel efficiency.

12.7.3 Strong Scaling

In an effort to further improve our time to solution, we explored the strong scaling properties of the Legion implementation. Using the smaller 576^3 grid size that we hoped to use for the majority of

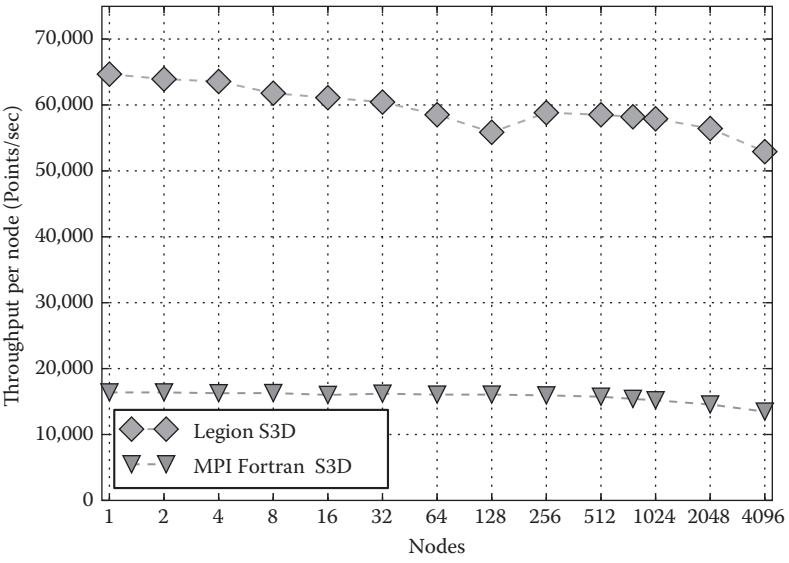

FIGURE 12.10 Weak scaling of PRF on Piz Daint.

* All runs on Titan made use of a *reranking* script that attempts to optimize the assignment of MPI ranks to nodes to match S3D's specific communication pattern to the underlying network topology [34].

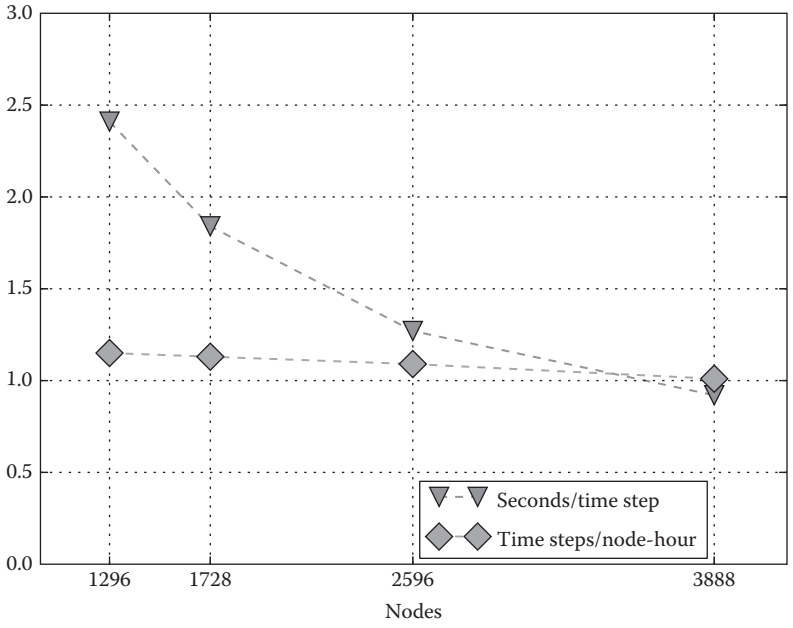

FIGURE 12.11 Strong scaling: PRF on Piz Daint for problem size of 576^3 points.

the simulation, we ran at node counts of 1296, 1728, 2596, and 3888. (The S3D code requires that the grid size divide evenly among the nodes, limiting the number of allowable node counts.) The results were excellent. Figure 12.11 shows the wall clock time per time step as well as the throughput per node (in timesteps/node-hour) as a measure of efficiency. Based on these results, we chose to run nearly all of the simulation of the 576^3 grid with 3888 nodes instead of the originally planned 1728 nodes, improving our simulation speed by a factor of 2 for only an 11% loss in efficiency.

12.8 CONCLUSIONS

We have presented the first 3-D DNS simulation of a PRF chemical mechanism at scale. Our results represent a substantial improvement in both the kind of combustion chemistry that can be studied as well as the time required to perform simulations. We reduced overall time to solution (including both development time and actual run time) over the previous state of the art, making it feasible to perform these computations in a reasonable amount of time and within existing computational budgets. Our approach also demonstrates that it is computationally tractable to conduct simulations of realistic chemical mechanisms such as PRF that had previously been possible only in reduced capacities, thus advancing the state of the art for computational combustion science both quantitatively and qualitatively.

Although our results have a significant impact within the domain of turbulent combustion with complex chemistry, our approach is more generally applicable. We have shown that by raising the level of programming abstraction using a task-based programming model, we can both reduce programming time and improve performance. Specifically, we have demonstrated the value of decoupling the specification of an application from how it is mapped onto a target architecture. By isolating correctness concerns from performance issues, we can develop performance-portable codes capable of being easily modified, tuned, and ported with a minimum amount of programmer effort.

In this work we have shown how the Legion programming model provides one approach to decoupling the specification of a program from how it is mapped. In particular, Legion isolates the specification of a program in the form of tasks and logical regions from how mapping decisions are made through the mapping interface. This property allowed us to easily port our version of S3D to several different architectures with minimal programmer effort and overhead. We further built on top of the Legion abstractions by creating the even higher-level Singe DSL compiler capable of emitting high-performance Legion tasks for a myriad array of chemical mechanisms and target architectures. The result is a cohesive framework for combustion chemistry coupled with compressible reacting fluid dynamics that addresses currently pressing problems while ensuring our code remains adaptable to upcoming machine architectures, and as yet unknown domain-specific variations, all the while achieving much improved performance over existing techniques.

The challenges we encountered are not unique to computational combustion science. For many current applications, time to solution is already dominated by programming effort, even for conceptually small extensions to existing codes. This situation will only become worse as both applications and machines continue to grow in complexity. Consequently, there will be an increasing need to develop performance-portable codes that can be easily reconfigured for new experiments and quickly adapted to new machine architectures. Under these circumstances, providing high-productivity computing environments such as Legion will be imperative to ensure that machines are efficiently utilized and codes achieve high performance. Our work demonstrates that this goal is not just a dream, but an achievable reality for actual production codes.

ACKNOWLEDGMENTS

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy (DOE) under Contract No. DE-AC05-00OR22725, as well as computing resources at the Swiss National Supercomputing Centre (CSCS). We thank the operations and user support staff at OLCF and CSCS for their tireless assistance. We thank Dr. Jack C. Wells of ORNL and Dr. Thomas C. Schulthess of CSCS for their support. This research was supported in part by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the US DOE through the ExaCT Combustion Co-Design Center.

REFERENCES

1. G. Bansal, A. Mascarenhas, and J. H. Chen. Direct numerical simulation of autoignition in stratified dimethyl-ether (dme)/air turbulent mixtures. *Combustion and Flame*, 162:688–702, 2015.
2. P. Gaffuri et al. A comprehensive modeling study of iso-octane oxidation. *Combustion and Flame*, 129:253–280, 2002.
3. T. Lu and C. K. Law. A directed relation graph method for mechanism reduction. *Proceedings of the Combustion Institute*, 30(1):1333–1341, 2005.
4. M. B. Loung et al. A dns study of the ignition of lean prf/air mixtures with temperature inhomogeneities under high pressure and intermediate temperature. *Combustion Flame,* 162:717–726, 2015.
5. M. B. Luong et al. Direct numerical simulations of the ignition of lean primary reference fuel/air mixtures under hcci condition. *Combustion and Flame*, 160:2038–2047, 2013.
6. Piz Daint & Piz Dora - CSCS. http://www.cscs.ch/computers/piz_daint_piz_dora/, 2013.
7. Introducing Titan—The World's #1 Open Science Supercomputer. https://www.olcf.ornl.gov/titan/, 2012.
8. J. H. Chen et al. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science and Discovery*, 2:015001, 2009.

9. M. Bauer, S. Treichler, E. Slaughter, and A. Aiken. Structure slicing: Extending logical regions with fields. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New Orleans, LA, 845–856. Piscataway, NJ: IEEE Press, 2014.
10. M. Bauer et al. Singe: Leveraging warp specialization for high performance on GPUs. In *Symposium on Principles and Practice of Parallel Programming*, February 2014.
11. T. J. Poinsot and S. K. Lele. Boundary-conditions for direct simulations of compressible viscous flows. *Journal of Computational Physics*, 101:104–129, 1 July 1992.
12. J. C. Sutherland and C. A. Kennedy. Improved boundary conditions for viscous, reacting, compressible ows. *Journal of Computational Physics*, 191:502–524, 2 November 2003. A.
13. R. Kee et al. *CHEMKIN-III: A Fortran Chemical Kinetics Package for the Analysis of Gas Phase Chemical and Plasma Kinetics*. Report number UC-405 SAND96-8216, Livermore, CA: Sandia National Laboratories. 1996.
14. R. B. Bird et al. *Transport Phenomena*. Hoboken, NJ: John Wiley & Sons, 1960.
15. C. A. Kennedy and M. H. Carpenter. Several new numerical methods for compressible shear- layer simulations. *Applied Numerical Mathematics*, 14(4):397–433, June 1994.
16. M. H. Carpenter et al. Fourth-order Runge-Kutta schemes for fluid mechanics applications. *Journal of Scientific Computing*, 25:157–194, October 2005.
17. M. Snir et al. *MPI-The Complete Reference*. Cambridge, MA: MIT Presss, 1998.
18. OpenACC Standard. http://www.openacc-standard.org.
19. J. Levesque et al. Hybridizing S3D into an exascale application using OpenACC: An approach for moving to multi-petaops and beyond. In *SC'12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 15:1–15:11, Salt Lake City, UT: IEEE, 2012.
20. J. Vetter et al. Keeneland: Bringing heterogeneous GPU computing to the computational science community. *Computing in Science Engineering*, 13:90–95, 2011.
21. M. Bauer et al. Legion: Expressing locality and independence with logical regions. In *Supercomputing Conference (SC)*, 2012.
22. The Open Community Runtime Interface. https://xstackwiki.modelado.org/images/1/13/Ocr-v0.9-spec.pdf, 2014.
23. C. Augonnet et al. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23:187–198, February 2011.
24. Q. Meng et al. Investigating applications portability with the uintah dag-based runtime system on petascale supercomputers. In *Supercomputing Conference (SC)*, pp. 96:1–96:12, New York, NY: ACM, 2013.
25. S. Treichler et al. Realm: An event-based low-level runtime for distributed memory architectures. In Parallel Architectures and Compilation Techniques (PACT), 2014.
26. T. Lu and C. K. Law. Toward accommodating realistic fuel chemistry in large-scale computations. *Progress in Energy and Combustion Science,* 35:192–215.
27. T. Lu et al. Dynamic stiffness removal for direct numerical simulations. *Combustion and Flame*, 156(8):1542–1551, 2009.
28. M. Bauer et al. CudaDMA: optimizing GPU memory bandwidth via warp specialization. In *SC '11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, Seattle, WA, 2011.
29. S. L. Kokjohn et al. Fuel reactivity controlled compression ignition (rcci): A pathway to controlled high-efficiency clean combustion. *International Journal of Engine Research*, 12:209–226, 2011.
30. Bhagatwala et al. Direct numerical simulations of hcci/saci with ethanol. *Combustion Flame*, 161:1826–1841, 2014.
31. P. Domingo and L. Vervisch. Triple flames and partially premixed combustion in autoignition of non-premixed turbulent mixtures. In *Symposium (International) on Combustion*, Vol. 26, pp. 233–240. Elsevier, 1996.
32. November 2014—top500 supercomputer sites. http://top500.org/lists/2014/11/, 2014.
33. D. Rossinelli et al. 11 pop/s simulations of cloud cavitation collapse. In *Supercomputing Conference (SC)*, pp. 3:1–3:13, New York, NY: ACM, 2013.
34. R. Sankaran et al. Genetic algorithm based task reordering to improve the performance of batch scheduled massively parallel scientific applications. *Concurrency and Computation: Practice and Experience*, 27(17):4763–4783, December 2015.