# Terascale direct numerical simulations of turbulent combustion using S3D



2009 Comput. Sci. Disc. 2 015001

(http://iopscience.iop.org/1749-4699/2/1/015001)

COMPUTATIONAL SCIENCE&DISCOVERY

# Terascale direct numerical simulations of turbulent combustion using S3D

**J H Chen[1], A Choudhary[2], B de Supinski[3], M DeVries[4], E R Hawkes[5], S Klasky[4], W K Liao[2], K L Ma[6], J Mellor-Crummey[7], N Podhorszki[4], R Sankaran[4], S Shende[8] and C S Yoo[1]**

[1] Sandia National Laboratories, Livermore, CA, USA
[2] Northwestern University, Evanston, IL, USA
[3] Lawrence Livermore National Laboratories, Livermore, CA, USA
[4] Oak Ridge National Laboratories, TN, USA
[5] University of New South Wales, Sydney, Australia
[6] University of California at Davis, CA, USA
[7] Rice University, Houston, TX, USA
[8] University of Oregon, Eugene, OR, USA
E-mail: jhchen@sandia.gov

Received 11 August 2008, in final form 5 December 2008
Published 23 January 2009
*Computational Science & Discovery* **2** (2009) 015001 (31pp)
doi:10.1088/1749-4699/2/1/015001

**Abstract.** Computational science is paramount to the understanding of underlying processes in internal combustion engines of the future that will utilize non-petroleum-based alternative fuels, including carbon-neutral biofuels, and burn in new combustion regimes that will attain high efficiency while minimizing emissions of particulates and nitrogen oxides. Next-generation engines will likely operate at higher pressures, with greater amounts of dilution and utilize alternative fuels that exhibit a wide range of chemical and physical properties. Therefore, there is a significant role for high-fidelity simulations, direct numerical simulations (DNS), specifically designed to capture key turbulence-chemistry interactions in these relatively uncharted combustion regimes, and in particular, that can discriminate the effects of differences in fuel properties. In DNS, all of the relevant turbulence and flame scales are resolved numerically using high-order accurate numerical algorithms. As a consequence terascale DNS are computationally intensive, require massive amounts of computing power and generate tens of terabytes of data. Recent results from terascale DNS of turbulent flames are presented here, illustrating its role in elucidating flame stabilization mechanisms in a lifted turbulent hydrogen/air jet flame in a hot air coflow, and the flame structure of a fuel-lean turbulent premixed jet flame. Computing at this scale requires close collaborations between computer and combustion scientists to provide optimized scaleable algorithms and software for terascale simulations, efficient collective parallel I/O, tools for volume visualization of multiscale, multivariate data and automating the combustion workflow. The enabling computer science, applied to combustion science, is also required in many other terascale



1749-4699/09/015001+31$30.00

physics and engineering simulations. In particular, performance monitoring is used to identify the performance of key kernels in the DNS code, S3D and especially memory intensive loops in the code. Through the careful application of loop transformations, data reuse in cache is exploited thereby reducing memory bandwidth needs, and hence, improving S3D's nodal performance. To enhance collective parallel I/O in S3D, an MPI-I/O caching design is used to construct a two-stage write-behind method for improving the performance of write-only operations. The simulations generate tens of terabytes of data requiring analysis. Interactive exploration of the simulation data is enabled by multivariate time-varying volume visualization. The visualization highlights spatial and temporal correlations between multiple reactive scalar fields using an intuitive user interface based on parallel coordinates and time histogram. Finally, an automated combustion workflow is designed using Kepler to manage large-scale data movement, data morphing, and archival and to provide a graphical display of run-time diagnostics.

## Contents

## 1. Introduction

Next-generation alternative-fuel internal combustion engines will operate in non-conventional, mixed-mode, turbulent combustion under previously uncharted turbulence-chemistry regimes. Compared to current engines, combustion in next-generation engines are likely to be characterized by higher pressures, lower temperatures, higher levels of dilution, and may operate overall fuel-lean near combustion limits. Combustion processes in these environments, combined with new physical and chemical fuel properties associated with non-petroleum-based fuels, result in complicated interactions that are poorly understood at a fundamental level. These unknown phenomena in tandem with new fuel parameters pose new demands for innovative simulations, particularly given the difficulties in obtaining detailed measurements in these adverse high-pressure environments. Therefore, there is an urgent demand for high-fidelity direct simulation approaches that capture these 'turbulence-chemistry' interactions, and in particular, capture and discriminate the effects of differences in fuel properties. DNS can uniquely isolate and reveal fundamental causal relationships among turbulence, turbulent mixing and reaction that are required for physical understanding and to develop predictive models used in engineering computational fluid dynamics to design and optimize practical devices.

The rapid growth in computational capabilities has presented both opportunities and challenges for DNS of turbulent combustion. The advent of terascale computing power made it possible to glean fundamental physical insight into fine-grained 'turbulence-chemistry interactions in simple laboratory scale, canonical turbulent flames with detailed chemistry using three-dimensional (3D) DNS. In DNS, the instantaneous governing equations are solved without averaging or filtering; therefore, all relevant continuum scales are resolved on the grid without any turbulence closure models using accurate high-order numerical methods for computational efficiency. Such simulations are costly, requiring several million CPU hours on a terascale computer, up to several billion grid points, and generating tens of terabytes of data. It is anticipated that petascale computing will provide access to directly simulating a larger dynamic range of scales and the transport of greater numbers of species representative of hydrocarbon fuels of practical interest.

In the present case study, we present recent results from terascale DNS of turbulent combustion that are enabled by collaboration between computer scientists and combustion scientists. First, we present a description of the governing compressible reactive flow equations, the solution approach in the DNS solver, S3D and the domain decomposition. Next, we demonstrate how the nodal performance of the DNS code, S3D, is improved by careful performance analysis on a Department of Energy (DOE) Office of Science supercomputer, the Cray XT Jaguar, followed by restructuring key data intensive kernels in S3D to reduce its memory bandwidth requirements. Next, the issue of effective parallel I/O at scale (O(10 000) processors) using MPI-I/O parallel shared-file I/O is addressed through a file-caching layer in the MPI-I/O library to address the lock problem. With the optimized code and collective MPI-I/O, recent terascale simulations of an autoigniting lifted hydrogen/air turbulent jet flame and a turbulent premixed methane–air Bunsen flame are described. These examples illustrate the role of DNS in elucidating how combustion is stabilized in a hot coflow—with similarities to stabilization of a lifted diesel jet flame—and understanding how the turbulent burning rate and flame structure of a premixed flame encountering intense turbulence is affected. These DNS produce tens of terabytes of raw data that subsequently need to analyzed and visualized to glean physical insight or to validate models. Next, we describe how multivariate volume visualization of spatially and temporally varying data is used to understand the correlation of dozens of reactive scalar fields together with the flow field. An intuitive visualization interface is described along with future requirements for *in situ* visualization as data sizes continue to grow. Finally, to reduce the manpower costs of managing the large and complex DNS data, workflow automation is adopted to manage the data movement, storage and monitoring of production runs.

## 2. S3D formulation and numerical methods

S3D is a massively parallel DNS solver developed at Sandia National Laboratories [1]. S3D solves the full compressible Navier–Stokes, total energy, species and mass continuity equations coupled with detailed chemistry. These governing equations are outlined in section 2.1. The governing equations are supplemented with additional constitutive relationships, such as the ideal gas equation of state, and models for reaction rates, molecular transport and thermodynamic properties.

### *2.1. Governing equations*

The equations governing reacting flows may be written in conservative form as

$$\frac{\partial \rho}{\partial t} = -\nabla_\beta \cdot (\rho \mathbf{u}_\beta), \tag{1}$$

$$\frac{\partial (\rho \mathbf{u}_\alpha)}{\partial t} = -\nabla_\beta \cdot (\rho \mathbf{u}_\alpha \mathbf{u}_\beta) + \nabla_\beta \cdot \boldsymbol{\tau}_{\beta\alpha} - \nabla_\alpha p + \rho \sum_{i=1}^{N_\mathrm{s}} Y_i \mathbf{f}_{\alpha i}, \tag{2}$$

$$\frac{\partial (\rho e_0)}{\partial t} = -\nabla_\beta \cdot \left[\mathbf{u}_\beta (\rho e_0 + p)\right] + \nabla_\beta \cdot (\tau_{\beta\alpha} \cdot \mathbf{u}_\alpha) - \nabla_\beta \cdot \mathbf{q}_\beta + \rho \sum_{i=1}^{N_\mathrm{s}} Y_i \mathbf{f}_{\alpha i} \cdot (\mathbf{V}_{\alpha i} + \mathbf{u}_\alpha), \tag{3}$$

$$\frac{\partial (\rho Y_i)}{\partial t} = -\nabla_\beta \cdot (\rho Y_i \mathbf{u}_\beta) - \nabla_\beta \cdot (\rho Y_i \mathbf{V}_{\beta i}) + W_i \dot{\omega}_i, \tag{4}$$

where $\nabla_\beta$ is the gradient operator in direction $\beta$, $Y_i$ is the mass fraction of species $i$, $W_i$ is the molecular weight of species $i$, $\tau_{\beta\alpha}$ is the stress tensor, $\mathbf{f}_{\alpha i}$ is the body force on species $i$ in direction $\alpha$, $\mathbf{q}_\beta$ is the heat flux vector, $\mathbf{V}_{\beta j}$ is the species mass diffusion velocity, $\dot{\omega}_i$ is the molar production rate of species $i$ and $e_0$ is the specific total energy (internal energy plus kinetic energy),

$$e_0 = \frac{\mathbf{u}_\alpha \cdot \mathbf{u}_\alpha}{2} - \frac{p}{\rho} + h \tag{5}$$

and $h$ is the total enthalpy (sensible plus chemical). Throughout this document, $\alpha$, $\beta$, $\gamma$ will indicate spatial indices and $i$, $j$ will indicate species indices unless stated otherwise. Repeated spatial indices imply summation. For example, in Cartesian coordinates,

$$\nabla_\beta \cdot (\rho \mathbf{u}_\beta) = \frac{\partial (\rho u)}{\partial x} + \frac{\partial (\rho v)}{\partial y} + \frac{\partial (\rho w)}{\partial z},$$

where $u$, $v$ and $w$ are the velocity components in the $x$-, $y$- and $z$-directions, respectively. Only $(N_\mathrm{s} - 1)$ species equations are solved because the sum of the $N_\mathrm{s}$ species equations yields the continuity equation. The mass fraction of the last species is determined from the constraint

$$\sum_{i=1}^{N_\mathrm{s}} Y_i = 1. \tag{6}$$

Assuming an ideal gas mixture, the equation of state is given as

$$p = \frac{\rho R_\mathrm{u} T}{W}, \tag{7}$$

where $R_\mathrm{u}$ is the universal gas constant and $W$ is the mixture molecular weight given by

$$W = \left( \sum_{i=1}^{N_\mathrm{s}} Y_i / W_i \right)^{-1} = \sum_{i=1}^{N_\mathrm{s}} X_i W_i. \tag{8}$$

The species mass fractions ($Y_i$) and mole fractions ($X_i$) are related by

$$\frac{Y_i}{X_i} = \frac{W_i}{W}. \tag{9}$$

Relevant thermodynamic relationships between enthalpy and temperature for an ideal gas mixture include

$$h = \sum_{i=1}^{N_\mathrm{s}} Y_i h_i, \quad h_i = h_i^0 + \int_{T_0}^{T} c_{p,i} \, \mathrm{d}T,$$

$$c_p = \sum_{i=1}^{N_\mathrm{s}} c_{p,i} Y_i, \quad c_p - c_v = R_u / W,$$

where $h_i$ is the enthalpy of species $i$, $h_i^0$ is the enthalpy of formation of species $i$ at temperature $T_0$, and $c_p$ and $c_v$ are the isobaric and isochoric heat capacities, respectively.

### *2.2. Constitutive relationships*

The stress tensor, species diffusion velocities and heat flux vector in equations (2)–(4) are given by [2–4]

$$\tau_{\beta\alpha} = \tau_{\alpha\beta} = \mu\left[\nabla_\alpha \mathbf{u}_\beta + \nabla_\beta \mathbf{u}_\alpha\right] - \delta_{\alpha\beta}\left(\tfrac{2}{3}\mu - \kappa\right)\nabla_\gamma \cdot \mathbf{u}_\gamma, \tag{10}$$

$$\mathbf{V}_{\alpha i} = \frac{1}{X_i}\sum_{j=1}^{N_s}\frac{Y_j}{X_j}D_{ij}\mathbf{d}_{\alpha j} - \frac{D_i^T}{\rho Y_i}\nabla_\alpha(\ln T), \tag{11}$$

$$\mathbf{q}_\alpha = -\lambda\nabla_\alpha T + \sum_{i=1}^{N_s} h_i \mathbf{J}_{\alpha i} - \sum_{i=1}^{N_s}\frac{p}{\rho Y_i}D_i^T \mathbf{d}_{\alpha j}, \tag{12}$$

where $\mu$ is the mixture viscosity, $\kappa$ is the bulk viscosity, $D_{ij}$ are the *multicomponent* diffusion coefficients, $D_i^T$ is the thermal diffusion coefficient for species $i$, $\lambda$ is the thermal conductivity, $\mathbf{J}_{\alpha i} = \rho Y_i \mathbf{V}_{\alpha i}$ is the species diffusive flux and $\mathbf{d}_{\alpha i}$ is the diffusion driving force for species $i$ in direction $\alpha$, given by [2–4]

$$\mathbf{d}_{\alpha i} = \underbrace{\nabla_\alpha X_i}_{1} + \underbrace{(X_i - Y_i)\nabla_\alpha(\ln p)}_{2} + \underbrace{\frac{\rho Y_i}{p}\left[\mathbf{f}_{\alpha i} - \sum_{j=1}^{N_s} Y_j \mathbf{f}_{\alpha j}\right]}_{3}. \tag{13}$$

The driving force vector involves thermodynamic forces generated by gradients in concentration (term 1), gradients in pressure (term 2) also called 'barodiffusion', and due to any body force such as an electrical or gravitational field (term 3). Equation (13) allows for the possibility that the force on each species, $\mathbf{f}_{\alpha i}$, is different, though in the case of a gravitational field, $\mathbf{f}_{\alpha j} = \mathbf{g}_\alpha$, and term 3 is identically zero. In the following sections, we will consider the fluxes given in equations (10)–(12) in more detail.

### *2.3. Stress tensor*

For monatomic gases, $\kappa$ is identically zero, and it is often neglected for polyatomic gases as well [2–4]. It will be neglected in all discussion herein. This simplifies (10) to

$$\tau_{\beta\alpha} = \tau_{\alpha\beta} = \mu\left(\nabla_\alpha \mathbf{u}_\beta + \nabla_\beta \mathbf{u}_\alpha - \tfrac{2}{3}\delta_{\alpha\beta}\nabla_\gamma \cdot \mathbf{u}_\gamma\right), \tag{14}$$

which is the form that will be used for this work.

### *2.4. Mass diffusion flux*

It should be noted that all diagonal components of the multicomponent diffusion coefficient matrix ($D_{ii}$) are identically zero [2]. Also, the diffusive fluxes and driving forces for all species must sum to zero,

$$\sum_{i=1}^{N_s}\mathbf{J}_{\alpha i} = \sum_{i=1}^{N_s}\rho Y_i \mathbf{V}_{\alpha i} = 0, \qquad \sum_{i=1}^{N_s}\mathbf{d}_{\alpha i} = 0. \tag{15}$$

Equation (11) is often approximated as [2–5]

$$\mathbf{V}_{\alpha i} = -\frac{D_i^{\text{mix}}}{X_i}\mathbf{d}_{\alpha i} - \frac{D_i^T}{\rho Y_i}\nabla_\alpha \ln T, \tag{16}$$

where $D_i^{\text{mix}}$ are 'mixture-averaged' diffusion coefficients given in terms of the binary diffusion coefficients ($\mathcal{D}_{ij}$) and the mixture composition as

$$D_i^{\text{mix}} = \frac{1 - X_i}{\sum_{j \neq i} X_j / \mathcal{D}_{ij}}, \tag{17}$$

where the binary coefficient matrix is symmetric ($\mathcal{D}_{ij} = \mathcal{D}_{ji}$), and the diagonal elements are zero ($\mathcal{D}_{ij} = 0$). If we assume that body forces act in the same manner on all species and baro-diffusion (term 2 in (13) is negligible, then (13) becomes $\mathbf{d}_{\alpha i} = \nabla_\alpha X_i$. If we further neglect the Soret effect, (the second term in (11) and (16), then (16) reduces to

$$\mathbf{V}_{\alpha i} = -\frac{D_i^{\text{mix}}}{X_i} \nabla_\alpha X_i, \tag{18}$$

which, using (9), can be expressed in terms of mass fractions as

$$\begin{aligned} \mathbf{V}_{\alpha i} &= -\frac{D_i^{\text{mix}}}{Y_i} \left[ \nabla_\alpha Y_i + \frac{Y_i}{W} \nabla_\alpha W \right] \\ &= -\frac{D_i^{\text{mix}}}{Y_i} \left[ \nabla_\alpha Y_i - Y_i W \sum_{j=1}^{N_s} \frac{\nabla_\alpha Y_j}{W_j} \right]. \end{aligned} \tag{19}$$

Studies on the effects of thermal diffusion suggest that the Soret effect is much more important for premixed flames than for non-premixed flames, the Dufour effect is of little importance in either premixed or non-premixed flames [6].

### *2.5. Heat flux*

The heat flux vector is comprised of three components representing the diffusion of heat due to temperature gradients, the diffusion of heat due to mass diffusion and the Dufour effect [2, 4–7]. In most combustion simulations, the Dufour effect is neglected, and (12) may be written as

$$\mathbf{q}_\alpha = -\lambda \nabla_\alpha T + \sum_{i=1}^{N_s} h_i \mathbf{J}_{\alpha i}. \tag{20}$$

All treatment here will be restricted to adiabatic systems. Although it is certainly true that radiation and other heat-loss mechanisms are important in many combustion applications, this complication will not be considered here.

### *2.6. Numerical methods*

S3D is based on a high-order accurate, non-dissipative numerical scheme. The governing equations are solved on a structured 3D Cartesian mesh. The solution is advanced in time through a six-stage fourth-order explicit Runge–Kutta method [8]. The solution is spatially discretized using an eighth-order central differencing scheme and a tenth-order filter is used to remove any spurious high-frequency fluctuations in the solution [9]. The spatial differencing and filtering require nine and eleven point stencils, respectively. CHEMKIN and TRANSPORT software libraries [10, 11] were linked with S3D to evaluate reaction rates, thermodynamic and mixture-averaged transport properties. Navier–Stokes characteristic boundary conditions (NSCBC) were used to prescribe the boundary conditions. Non-reflecting characteristic inflow/outflow boundary conditions [12, 13] are typically prescribed in stationary DNS configurations.

S3D is parallelized using a 3D domain decomposition and MPI communication. Each MPI process is in charge of a piece of the 3D domain. All MPI processes have the same number of grid points and the

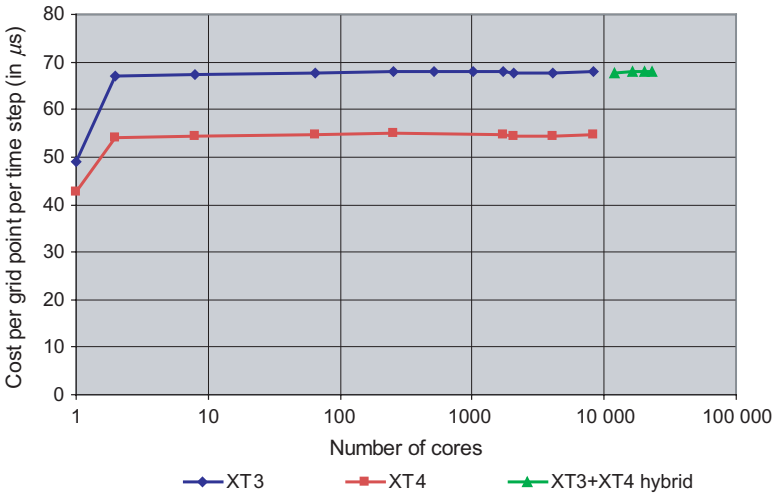


**Figure 1.** Weak scaling performance of S3D on a Cray XT3 + XT4 hybrid at NCCS.

same computational load. Inter-processor communication is only between nearest neighbors in a 3D topology. The messages for a typical problem are approximately 80 kB. A ghost-zone is constructed at the processor boundaries by non-blocking MPI sends and receives among the nearest neighbors in the 3D processor topology. All-to-all communications are only required for monitoring and synchronization ahead of I/O.

S3D shows good parallel performance on several architectures and can make effective use of a large fraction of the Office of Science leadership computing platforms [14]. The rapid growth of computational capabilities has presented significant opportunities for DNS of turbulent combustion. Increases in computational power allow for increased grid size and/or a larger number of time steps in the simulation. Both of these are favorable to the scientific goals by helping achieve higher Reynolds numbers, a larger sample of turbulent structures for better statistical measures and a longer simulation time to obtain a more complete temporal development of the turbulent flame. Increases in computational power also allow for the solution of a larger number of species equations, thereby allowing the simulation of fuels with higher chemical complexity. Recently, S3D has been used to perform several hero simulations of combustion problems ranging from premixed flames (200 million grid points, 18 variables) [15], non-premixed flames (500 million grid points, 16 variables) [16], to lifted jet flames (1 billion grid points, 14 variables) [17].

## 3. NCCS Cray XT platform

The NCCS Cray XT (Jaguar) in 2007 is a hybrid platform that consists of a mix of Cray XT3 and Cray XT4 nodes. The system currently has a total of 11 706 nodes, with 11 508 configured as compute nodes. Each compute node has a 2.6 GHz dual core AMD Opteron processor with 4 GB of memory. The remaining 198 nodes are used as service nodes. Each service node has a 2.6 GHz dual core AMD Opteron processor and 8 GB of memory. Of Jaguar's compute nodes, 6214 are XT3 nodes, which have a peak memory bandwidth of 6.4 GB $s^{-1}$. The remaining 5294 compute nodes are XT4 nodes, which are populated with 667 MHz DDR2 memory that provides a peak memory bandwidth 10.6 GB $s^{-1}$.

## 4. S3D performance

S3D uses almost no collective communication; almost all communication is nearest neighbor point-to-point non-blocking operations with the opportunity to overlap message communication. Thus, S3D exhibits excellent weak scaling performance.[9] Figure 1 shows results from running a model problem that uses a 50 × 50 × 50 grid per core on Jaguar. Runs exclusively on XT4 nodes consistently take approximately 55 ms per grid point

[9] Weak scaling is when the problem size is directly proportional to the number of processors.

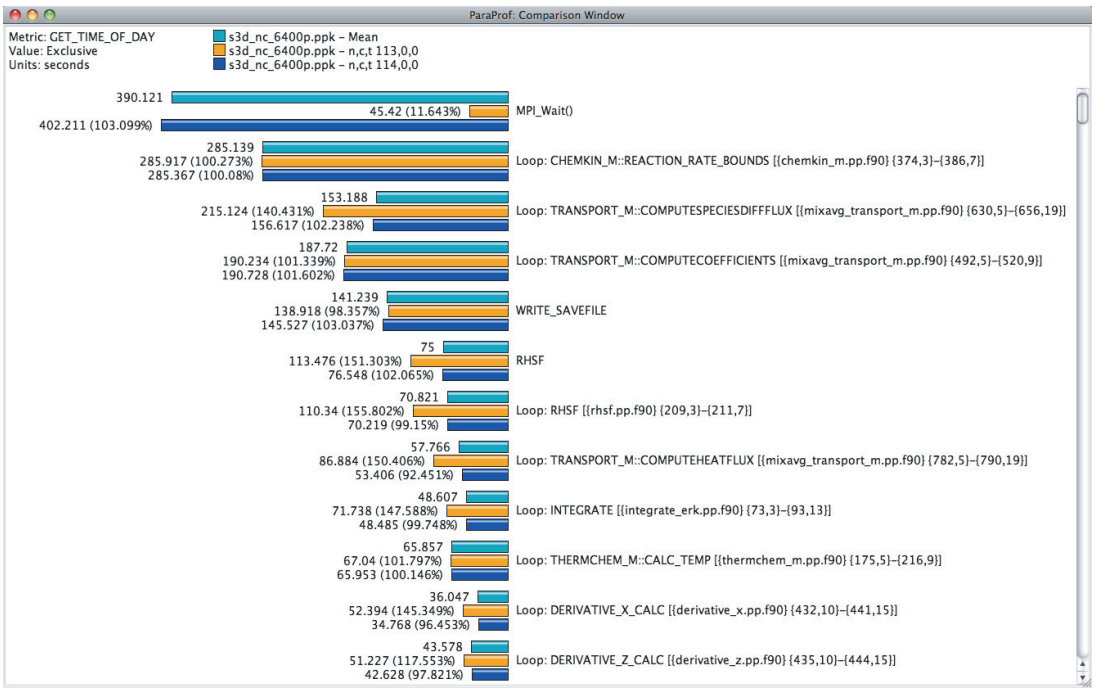


**Figure 2.** Distribution of exclusive time among S3D's procedures and loops for selected processes in a 6400 core execution.

per time step as the computation is scaled from 2 to 8192 cores, as shown by the red curve in the figure. While runs exclusively on XT3 nodes yield similar flat weak scaling, the runs take slightly longer—approximately 68 ms per grid point per time step—due to the lower memory bandwidth of the XT3 nodes. The blue curve in figure 1 shows the performance of the XT3 nodes.

With Jaguar's current configuration, runs on more than 8192 cores must use a combination of XT4 and XT3 nodes. S3D also exhibits outstanding weak scaling performance on such heterogeneous configurations, as shown by the green curve in figure 1. However, performance is dominated by the memory bandwidth limitations of the XT3 nodes when a constant problem size per core is used. Thus, the cost per grid point per time step from 12 000 to 22 800 cores is approximately 68 ms, matching the computation rate on the XT3 cores alone.

We performed a detailed performance analysis of runs on heterogeneous allocations using TAU [18]. Figure 2 shows a quantitative breakdown of execution time among loops and procedures for two representative processes, along with the average breakdown, in a 6400 core execution. The processes exhibit two general equivalence classes of performance. One class, as represented by the blue bars, spends substantially longer in `MPI_Wait`, than the other class, as represented by the orange bars. Analysis of the metadata associated with the execution confirmed that the processes in the first class were mapped to XT4 nodes, whereas those in the second were mapped onto XT3 nodes. Close examination of the figure shows that some of the activities, such as the second (`REACTION_RATE_BOUNDS`), take nearly identical time in both classes, whereas for others, such as the third (`COMPUTESPECIESDIFFFLUX`), the processes on XT3 nodes take noticeably longer. Further study of these data showed that CPU-bound computations take approximately the same time on both XT3 and XT4 nodes, whereas memory-intensive loops take longer on the XT3 nodes. Overall, these results show that overall application performance on the hybrid system is limited by the memory bandwidth of the XT3 nodes. The slower performance of the XT3 nodes on memory-intensive loop nests causes the XT3 nodes to arrive late at communication events, which is reflected by the longer waiting time on XT4 nodes.

As part of the 2007 Joule metric, S3D will run a large problem on the full Jaguar installation. The overall goal is to run a very large problem that demonstrates the addition of new chemistry capabilities to S3D. However, completion of that problem will require outstanding single node and scaling performance. Although S3D already performs very well on the combined machine, the clear performance limitation of the XT3 nodes on overall performance suggests a straighforward mechanism to improve overall performance.

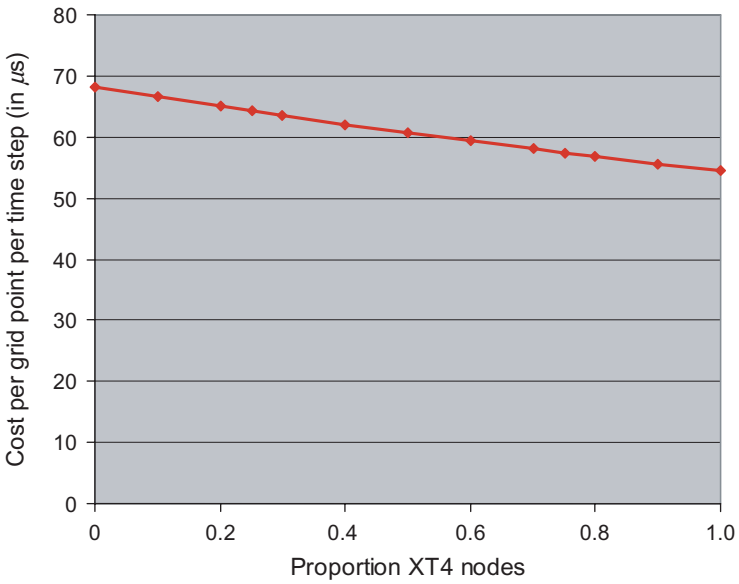


**Figure 3.** Predicted average cost per grid point when balancing computational load between Cray XT3 and XT4 nodes on a hybrid system.

Specifically, we can run a smaller problem size on the XT3 nodes than on the XT4 nodes, compensating for the approximately 24% performance degradation caused by the lower memory bandwidth of the XT3 nodes. Overall, we conservatively estimate that a $50 \times 50 \times 40$ grid size on the XT3 nodes will take no longer than the $50 \times 50 \times 50$ problem takes on the XT4 nodes. By only reducing one dimension, we will minimize the coding effort required to support heterogeneous problem sizes per task in S3D. The resulting time per grid point per time step will then depend on the proportion of XT4 nodes to XT3 nodes, as shown in figure 3. In the current configuration, 46% of the nodes are XT4 nodes, leading to a predicted performance of 61 ms per grid point per time step when averaged across all of the processors in the hybrid system.

### *4.1. Improving the node performance of S3D*

Data-intensive codes such as S3D tax the capabilities of microprocessor-based computer systems. The memory subsystem of Jaguar's XT3 compute nodes can deliver less than 3 B per clock cycle from memory to the Opteron processor. For this reason, the speed of computations that do not reuse data in registers or cache is limited by the machine's memory bandwidth; without careful optimization, scientific codes often achieve as little as 5–10% of peak performance on microprocessor-based systems. Therefore, restructuring S3D's data-intensive kernels to reduce their memory bandwidth requirements offers an opportunity for boosting application performance.

Another motivation for restructuring S3D to reduce its memory bandwidth demands is that the XT3 and XT4 nodes in Jaguar differ in the maximum memory bandwidth that they support. Because of their faster memory, the XT4 nodes are 24% faster, as shown in figure 1. Restructuring S3D to reduce its memory bandwidth needs will reduce the performance disparity between the XT3 and XT4 nodes in the hybrid system.

To investigate the node performance of S3D, we used Rice University's HPCTOOLKIT performance tools [19] to study a single-processor execution of a pressure wave test on a $50^3$ domain. For the study, we used a single node of a Cray XD1 with a 2.2 GHz Opteron 275 processor and DDR 400 memory, which provides 6.4 GB $s^{-1}$ of memory bandwidth (as on Jaguar's XT3 nodes). For the model problem, S3D achieved 0.305 FLOPs $cycle^{-1}$, which represents 15% of peak.

Using HPCTOOLKIT, we pinpointed several data-intensive kernels in S3D that did not fully exploit the memory hierarchy. Figure 4 shows S3D's diffusive flux computation, which was the most costly loop nest in the execution. The loop nest updates a 5D data structure. Two loops over the direction and the number of species appear explicitly in the source code; other 3D loops are implicit in the Fortran 90 array operations. For

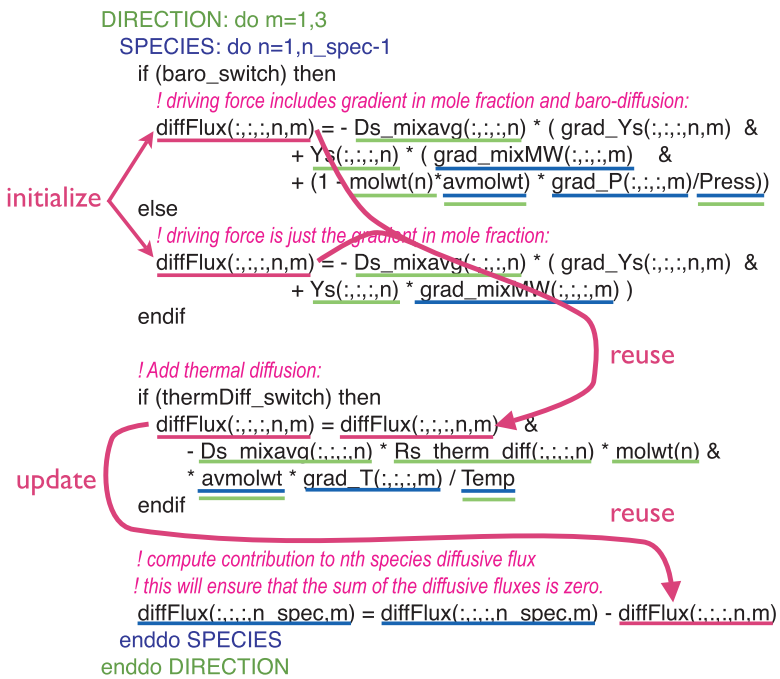


**Figure 4.** S3D's diffusive flux computation annotated with opportunities for reusing data values.

the model problem, this 5D loop nest accounts for 11.3% of S3D's execution time on a Cray XD1 node. By comparing the FLOPs executed by the loop nest and the cycles spent in it, we found that the loop nest achieves only 4% of the theoretical peak performance.

In figure 4, the code for the diffusive flux computation is overlaid with colored markings that indicate data reuse. Definitions and uses of the `diffFlux` array are underlined in red. The red arrows show how definitions of values are reused by later statements within the DIRECTION and SPECIES loops. The outer loops also offer a myriad of other opportunities for reusing data. References underlined in green will be used by every iteration of the DIRECTION loop since they lack an `m` subscript. References underlined in blue will be used by every iteration of the SPECIES loop since they lack an `n` subscript. If the code is executed as naturally written, almost all of the values will be evicted from cache rather than being reused because each $50^3$ slice of the `diffFlux` array almost completely fills the 1 MB secondary cache.

To restructure the diffusive flux computation to exploit the potential reuse, we used Rice University's LoopTool utility [20], which supports source-to-source restructuring of loop nests written in Fortran. To apply LoopTool, we outlined the loop nest into a separate file, and marked up the code with directives indicating the transformations that it should perform. Figure 5 provides a pictorial rendering of the code restructuring performed with LoopTool. The left side of the figure shows a diagram that represents the structure of the original code. The green and brown arcs represent the DIRECTION and SPECIES loops. The two conditionals are shown explicitly, though the logical predicate variables have been abbreviated. The thick red, purple, blue and black lines represent the code's Fortran 90 array statements.

The right side of the figure shows a diagram that represents the code after LoopTool applied the transformations listed in the figure. Unswitching the two conditionals out of the original loop nest yields four loop nests, each customized for a particular setting of the switches. The gray arcs in the transformed code represent the loops over the first three dimensions of the `diffFlux` array that arise when LoopTool scalarizes the Fortran 90 array notation. Within each of the four customized loop nests, all of the colored lines representing statement instances have been fused into a single set of triply nested loops. The arc for the green `m` loop is not present in the transformed code. Instead, the code has been unrolled three times; replications of the inner loop resulting from this unrolling are indicated by the green shading. Similarly, the brown `n` loop has been unrolled by two, which causes an orthogonal duplication of each of the three copies of the innermost loop body present after unrolling the `m` loop. This additional duplication is shown by the left-to-right duplication

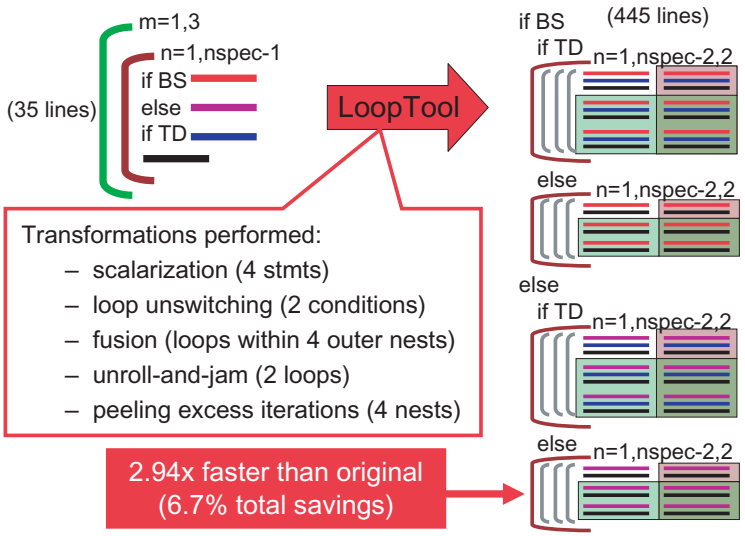


**Figure 5.** Using LoopTool to optimize S3D's diffusive flux computation on a $50^3$ domain.

of statements shaded in brown. This figure conveys some of the complexity of the resulting code following LoopTool's transformations. The LoopTool-generated code for this loop nest runs 2.94 times faster than the original. This change alone reduced the entire program's execution time by 6.8% for a $50^3$ problem size.

Further analysis of S3D's node performance with HPCTOOLKIT, tuning loops with LoopTool, and some manual adjustment of procedure argument passing conventions to avoid copying 4D slices of a 5D array yielded an aggregate improvement of roughly 12.7% for the $50^3$ problem size on the Cray XD1.

## 5. Parallel I/O enhancement

Modern parallel file systems employ multiple file servers and disk arrays in order to sustain the high data throughput requirement from today's I/O intensive applications [54]. In theory, this strategy allows the aggregate I/O bandwidth to scale as the number of storage devices increases. However, some factors both in hardware and software prevent one from observing data transfer rates that are close to the I/O system's peak performance. First of all, the access granularity of a disk drive is the size of a disk sector. It is the file system's responsibility to enforce atomic access to disk sectors. Traditionally, file systems adopt file locking to accomplish this task. File locking is also used for data consistency and cache coherence controls when multiple processes operate on a shared file in parallel [55]. Obviously, when the file ranges of two lock requests conflict, I/O must be carried out in sequence. As file systems enforcing the sector's I/O atomicity, parallel I/O can also be serialized when the requests conflict at the sector boundaries, even if the requests do not conflict in bytes. In real world, parallel applications' I/O is seldom aligned with the disk sector size and thus the I/O performance hardly reaches the hardware potential.

It is important to address this lock contention problem particularly in the parallel environments where applications employ multiple processes to solve a single problem. In parallel applications, the problem domain is usually represented by a set of global data structures, such as multi-dimensional arrays, which are partitioned among the processes. When writing the global data structures to files, it is desirable to maintain the global canonical order. Combining the partitioned data structures from multiple clients involves concurrent write operations to a shared file. We developed a file-caching layer in MPI-I/O library to face the lock problem in parallel shared-file I/O [56]. This layer acts as a buffer zone between applications and the underlying file system, providing an opportunity to reorganize the I/O requests among the MPI processes so that requests from each process can be aligned with the system lock boundaries. We refer to this work as MPI-I/O caching. The goal is through process collaboration to achieve better, effective client-side file caching. One immediate benefit is to relieve the cache coherence control workload from I/O servers. While the MPI-I/O caching handles general I/O patterns that can consist of mixed read and write operations, we have also designed a

two-stage write-behind buffering scheme to specifically address the write-only I/O pattern, where majority of the I/O operations in today's parallel applications are write-only.

### *5.1. MPI-I/O caching*

As it is designed for MPI applications, our caching system uses the MPI communicator supplied to the file open call to identify the scope of processes that will collaborate with each other to perform file caching. To preserve the existing optimizations in ROMIO, like two-phase I/O and data sieving, we incorporate our design in the Abstract Device IO (ADIO) layer where ROMIO interfaces with underlying file systems [57]. To enable the collaboration of MPI processes, the caching system needs a transparent mechanism in each process that can run independently and concurrently with the main program. We create a thread in each MPI process when the first file is opened and keep the threads alive until the last file is closed. With the I/O thread running in the background to carry out the work of file I/O and caching, the program main thread can continue without interruption. In our design, each I/O thread handles both local and remote requests to the locally cached data, and cooperates with remote threads for coherence control.

Our caching scheme logically divides a file into equally-sized pages, each of which can be cached. The default cache page size is set to the file system block size and is also adjustable through an MPI hint. A page size aligned with the file system lock granularity is recommended, since it prevents false sharing. Cache metadata describing the cache status of these pages are statically distributed in a round-robin fashion among the MPI processes that collectively open the shared file. This distributed approach avoids centralization of metadata management. To ensure atomic access to metadata, a distributed locking mechanism is implemented where each MPI process manages the lock requests for its assigned metadata. Metadata locks must be obtained before an MPI process can freely access the metadata, cache page, and the file range corresponding to the page.

To simplify coherence control, we allow at most a single cached copy of file data among all MPI processes. When accessing a file page that is not being cached anywhere, the requesting process will try to cache the page locally, by reading the entire page from the file system if it is a read operation, or by reading the necessary part of the page if it is a write operation. An upper bound, by default 32 MB, indicates the maximum memory size that can be used for caching. If the memory allocation utility, `malloc()` finds enough memory to accommodate the page and the total allocated cache size is below the upper bound, the page will be cached. Otherwise, under memory pressure, the page eviction routine is activated. Eviction is solely based on only local references and a least-recent-used policy. If the requested page is already cached locally, a simple memory copy will satisfy the request. If the page is cached at a remote process, the request is forwarded to the page owner. An example I/O flow for a read operation is illustrated in figure 6 with four MPI processes. In this example, process $P_1$ reads data in file page 7. The first step is to lock and retrieve the metadata of page 7 from $P_3$ ($7 \bmod 4 = 3$). If the page is not cached yet, $P_1$ will cache it locally (into local page 3) by reading from the file system, as depicted by steps (2.a) and (3.a). If the metadata indicates that the page is currently cached on $P_2$, then an MPI message is sent from $P_1$ to $P_2$ asking for data transfer. In step (3.b), assuming file page 7 is cached in local page 2, $P_2$ sends the requested data to $P_1$. When closing a file, all dirty cache pages are flushed to the file system. A high water mark is used in each cache page to indicate the range of dirty data, so that flushing needs not always be an entire page.

### *5.2. Two-stage write-behind buffering*

We used our MPI-I/O caching design as a basis for constructing a two-stage write-behind method to focus on improving the performance of write-only operations. Although MPI-I/O caching can handle I/O mixed with read and write operations, this write-behind method has two restrictions: the file must be opened in the write-only mode and the MPI atomic mode must be set to false. The former is indicated by the use of `MPI_MODE_WRONLY` in `MPI_File_open()` and the later is the default mode in MPI I/O. In the first stage of this method, write data are accumulated in local buffers along with the requesting file offset and length. Once the local buffer is full, the data are sent to the global buffer at the second stage. Double buffering is used so one buffer can accumulate incoming write data, whereas the other is asynchronous sent to the global buffer. The global buffer's assignment consists of file pages statically distributed among the MPI processes that collectively

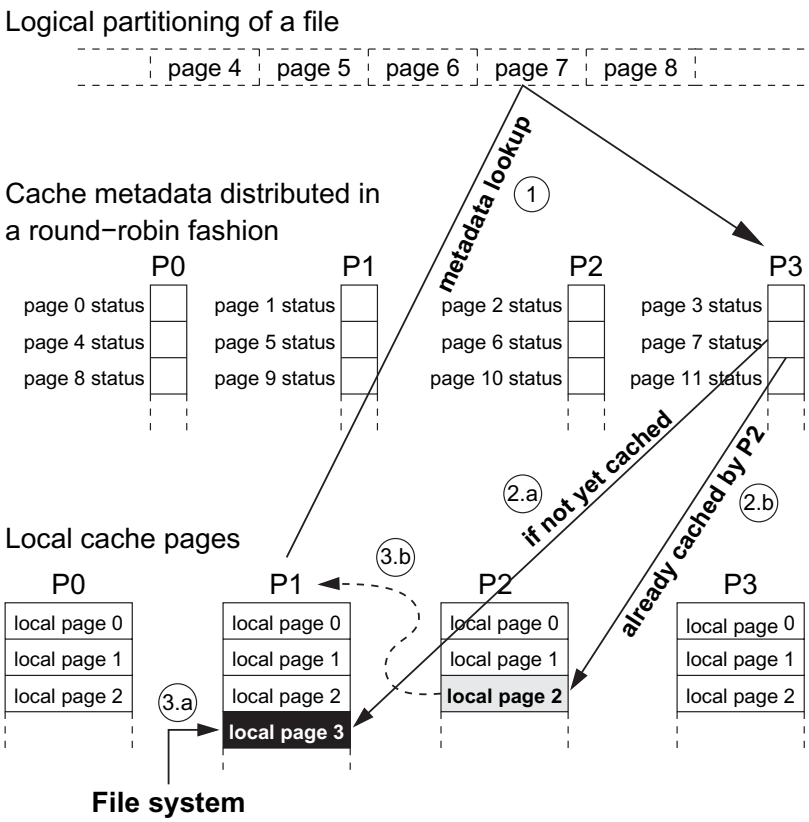


**Figure 6.** Example of the I/O flow in MPI-I/O caching where MPI process $P_1$ reads data from logical file page 7.

open a shared file. A file is logically divided into equally sized pages with each page bound to a single MPI process in a round-robin striping scheme. Accordingly, page $i$ resides on the process of rank ($i$ mod *nproc*), where *nproc* is the number of processes. To minimize lock contention in the underlying file system, the default page size is set to the file system stripe size, but it is adjustable through an MPI hint.

The first-stage local buffer is separated into $(nproc - 1)$ sub-buffers: each of which are dedicated to a remote MPI process. When accumulating in the local buffer, write data are appended to the sub-buffer corresponding to its destination MPI process. The default size for each local sub-buffer is 64 kB, but can be changed by the application through an MPI hint. Global file pages reside in memory until their eviction is necessary. Figure 7 illustrates the operations of the two-stage write-behind method. In this example, there are four MPI processes and each has three first-stage local sub-buffers. Each sub-buffer is corresponding to a remote process. At $P_2$, when the sub-buffer tied to $P_1$ is full, it is flushed to $P_1$. After receiving the flushed data from $P_2$, $P_1$ redistributes it to the global pages based on the included offset-length information.

Since the global buffer is distributed over multiple processes, each process must be able to respond to remote requests for flushing the first-stage write-behind data. Similar to the caching implementation, we use the I/O thread approach to permit the process collaboration without interrupting the main program thread. Each process can have multiple files opened, but only one thread is created. Once an I/O thread is created, it enters an infinite loop to serve both local and remote write requests until it is signaled to terminate by the main thread.

### *5.3. I/O kernel benchmark*

We evaluated the S3D I/O kernel on two machines at the National Center for Supercomputing Applications: Tungsten and Mercury. Tungsten runs a Red Hat Linux operating system and the compute nodes are interconnected by both Myrinet and Gigabit Ethernet communication networks. A Lustre parallel file system version 1.4.4.5 is installed on Tungsten. Output files are saved in a directory configured with a stripe count

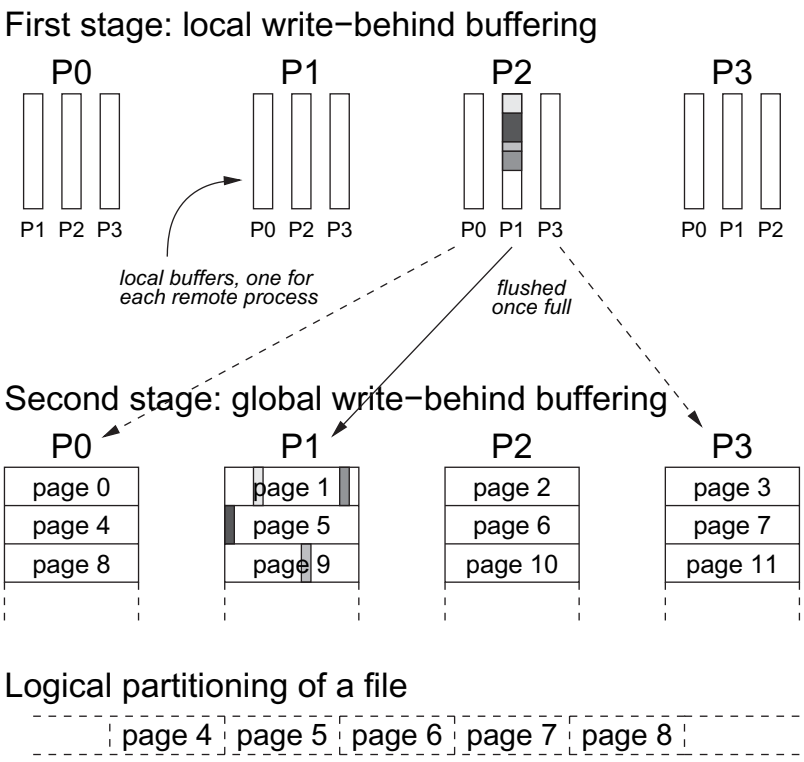


**Figure 7.** Design of two-stage write-behind buffering.

of 16 and a 512 kB stripe size. Mercury runs a SuSE Linux operating system and the compute nodes are also inter-connected by both Myrinet and Gigabit Ethernet. Mercury runs an IBM GPFS parallel file system version 3.1.0 configured in the Network Shared Disk server model with 54 I/O servers and 512 kB file block size. MPI-I/O caching is implemented in the ROMIO layer of MPICH version 2-1.0.5, the latest thread-safe version of MPICH2 at the time our experiments were performed. Support for thread safety is limited, however, to the default sock channel of MPICH2; thereby restricting inter-process communication in our experiments to the slower Gigabit Ethernet. We used a 512 kB page size for both MPI-I/O caching and the two-stage write-behind method. Setting the cache page size to the file system stripe size aligns all write requests to the stripe boundaries and hence lock boundaries.

In the S3D-I/O kernel, a checkpoint is performed at regular intervals, and its data consist primarily of the solved variables in 8 B 3D arrays, corresponding to the values at the 3D Cartesian mesh points. At each checkpoint, four global arrays are written to files and they represent the variables of mass, velocity, pressure and temperature, respectively. Mass and velocity are 4D arrays, whereas pressure and temperature are 3D arrays. All four arrays share the same size for the lowest three spatial dimensions $X$, $Y$ and $Z$, and they are all partitioned among MPI processes along $X$–$Y$–$Z$ dimensions in the same block–block–block fashion. For the mass and velocity arrays, the length of the fourth dimension is 11 and 3, respectively. The fourth dimension is not partitioned.

In the original S3D, file I/O is programmed in Fortran I/O functions and each process writes its sub-arrays to a new, separate file at each checkpoint. We added a functionality of using MPI I/O to write the arrays into a shared file in their canonical order. With this change, there is only one file created per checkpoint, regardless of the number of MPI processes used. Figure 8 shows the data partitioning pattern on a 3D array and the mapping of a 4D sub-array to the global array in file. 3D arrays can be considered a special case with a fourth dimension length of one. For performance evaluation, we keep the size of partitioned $X$–$Y$–$Z$ dimensions a constant $50 \times 50 \times 50$ in each process. This produces about 15.26 MB of write data per process per checkpoint. As we increase the number of MPI processes, the aggregate I/O amount proportionally increases as well. Given the array sizes, it is clear that the S3D I/O does not generate write request offsets that are aligned with file system lock boundaries. Thus, conflict locks due to false sharing and I/O serialization are expected.

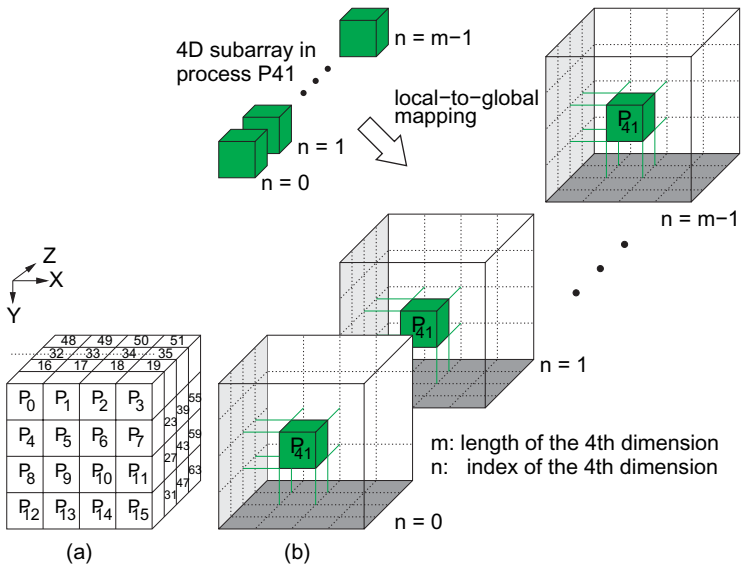


**Figure 8.** S3D-I/O data partitioning pattern. (a) For 3D arrays, the sub-array of each process is mapped to the global array in a fashion of block partitioning in all $X$–$Y$–$Z$ dimensions. (b) For 4D arrays, the lowest $X$–$Y$–$Z$ dimensions are partitioned the same as the 3D arrays, whereas the fourth dimension is not partitioned. This example uses 64 processes and highlights the mapping of process $P_{41}$'s sub-array to the global array.

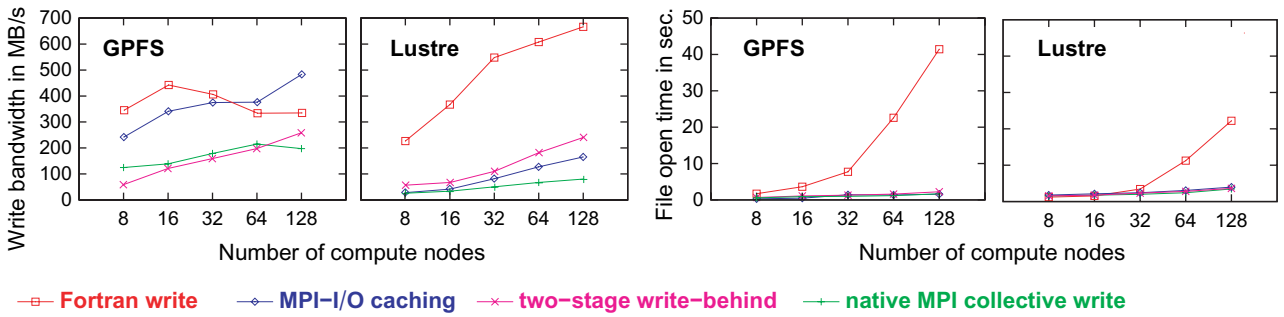


**Figure 9.** Write bandwidth and file open timing for S3D-I/O benchmark.

We measure the execution time and calculate the write bandwidth for running ten checkpoints. Figure 9 shows the write bandwidth results for using Fortran I/O, MPI collective I/O natively, collective I/O with MPI-I/O caching and independent I/O with the two-stage write-behind method. Fortran I/O has significantly better performance than the others cases on Lustre, but this situation changes on GPFS. On GPFS, Fortran I/O is even worse than the MPI-I/O caching for 64 and 128 processes. Further investigation by separately measuring the file open time of ten checkpoints shows that file open costs increase more dramatically on GPFS than Lustre when scaling the number of processes. Lustre seems to handle larger number of files more efficiently than GPFS. As for the MPI-I/O performance, we observe that MPI-I/O caching outperforms the native collective I/O on both GPFS and Lustre. This improvement is the effect of I/O alignment with the file system lock boundaries.

The two-stage write-behind behaves differently on each file system in that it is worse than the native collective I/O on GPFS, but outperforms the MPI-I/O caching on Lustre. We think write-behind performance is worse because the S3D-I/O pattern results in many remote data flushes from the local first-stage to global second-stage buffers. Note that the native MPI-I/O case uses collective I/O functions, whereas the write-behind method uses independent I/O functions. It has been reported in [58] that using independent I/O natively results in significantly worse write bandwidth than using collective I/O. This is because there is a significant difference in the number of the write requests to the file system. Compared to the use of native

independent writes that produces I/O bandwidth less than 5 MB per second, our write-behind method shows a clear improvement and achieves a bandwidth close to, or better than, the collective I/O method.

In fact, both the MPI-I/O caching and two-stage write-behind methods have their own advantages and disadvantages. As we described earlier, the distributed lock protocol used in MPI-I/O caching to maintain the cache metadata integrity incurs a certain degree of communication overhead. Because the write-behind method deals with write-only patterns, no coherence control is required at all. MPI-I/O caching allows the first process that requests a page to buffer it locally; thereby, taking advantage of data locality and increasing the likelihood of local cache hits. In contrast, since the second-stage buffers of the write-behind method are statically assigned across MPI processes in a round-robin fashion, the data written by a process in the first-stage buffers will most likely need to be flushed to remote processes.

The S3D-I/O experiments demonstrate the significance of aligning the I/O requests with the file system lock boundaries. In fact, enforcing POSIX semantics such as I/O atomicity has become substantial obstacles to parallel file systems for efficiently handling shared-file I/O at sustained I/O rates close to the maximum network bandwidth. Since the primary I/O patterns of large-scale parallel applications are write-only and non-overlapping, enforcing strict semantics like POSIX atomicity is not always necessary. We plan to add a mechanism in MPI-I/O caching to selectively relax the atomicity requirement. Our current design for MPI-I/O caching conservatively chooses to keep at most a single copy of cached file data. Although simplifying coherence control, this restriction increases the likelihood of remote cache page access over local accesses. We will investigate a new implementation that allows multiple cached copies of the same file date. We also plan to explore other issues such as cache load rebalancing and data prefetching.

## 6. DNS of a lifted turbulent jet flame in vitiated coflow

### *6.1. Introduction*

Turbulent lifted flames have been widely investigated due to their important role both in practical applications such as direct injection stratified spark ignition engines, diesel engines and commercial boilers, and in understanding fundamental combustion phenomena as a building-block flame in turbulent combustion. In particular, the stabilization mechanism of a lifted flame base has drawn great attention because the lifted flame base determines the overall flame stability and the characteristics of combustion systems [21–23]. Despite the importance of flame base stabilization, however, there has thus far been little consensus among researchers regarding the dominant mechanism which stabilizes the lifted flame base, not only because of the complex structure and propagation characteristics of turbulent lifted flames, but also because of the difficulty in obtaining time-series multi-scalar velocity measurements.

Several theories have been proposed to explain the stabilization mechanism of turbulent lifted flames [21]. In general, these theories can be categorized into five different groups depending on the degree of premixedness of the mixture upstream of the lifted flame base and on the scale of the turbulence structure: premixed flame theory, non-premixed flamelet theory, edge flame theory, turbulence intensity theory and large eddy theory. In spite of the proposed theories, it is still highly controversial as to what the stabilization mechanism is. Definitive evidence substantiating a dominant stabilization mechanism is still unattainable due to inherent limitations in scalar-velocity measurements. Recently, autoignition was proposed as an additional stabilization mechanism of lifted flames in a heated coflow [24]. As a consequence of the stabilizing influence of autoignition, the recirculation region of hot combustion products is often used to stabilize a lifted flame in practical combustors. For example, in diesel engines, fuel is injected and mixed with a heated oxidizer coflow in the chamber at temperatures above the ignition limit, such that the stability and overall characteristics of the lifted flame and soot processes are highly affected by the heated oxidizer stream [25]. Thus, to investigate the stabilization mechanism of a turbulent lifted hydrogen jet flame in a heated coflow, 3D DNS with detailed hydrogen/air chemistry was performed. The role of autoignition resulting from the heated coflow is examined in detail to determine the stabilization mechanism of the flame. In addition, the flame structure is characterized at different axial locations downstream of the burner exit in terms of conditional flame statistics.

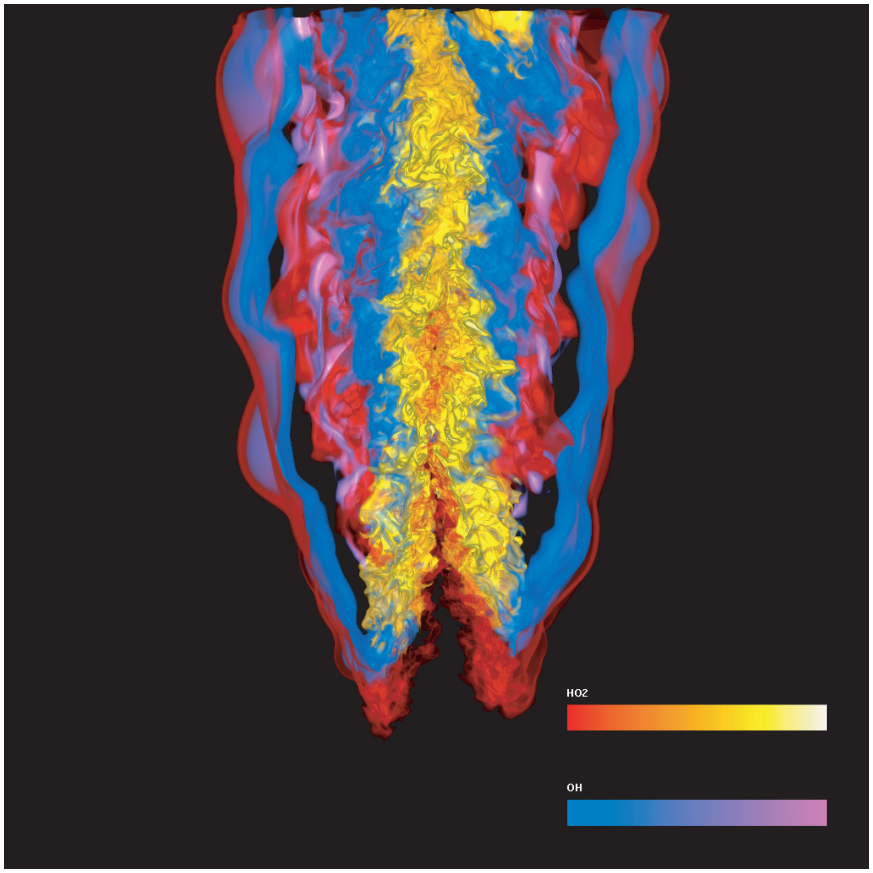


**Figure 10.** Instantaneous image of $HO_2$ and OH from DNS of a lifted hydrogen/air jet flame.

### *6.2. Problem configuration*

The simulation was performed in a 3D slot-burner configuration. Fuel issues from a central jet, which consists of 65% hydrogen and 35% nitrogen by volume with an inlet temperature of 400 K. The central jet is surrounded on either side by coflowing heated air at 1100 K. This temperature is greater than the crossover temperature of hydrogen/air chemistry, such that the mixture upstream of the flame base is autoignitable. A uniform grid spacing of 15 $\mu$m was used in the streamwise direction, $x$ and spanwise direction, $z$, whereas an algebraically stretched mesh was used in the transverse direction, $y$. The domain size is $L_x$, $L_y$, $L_z$ = 2.4, 3.2, 0.64 $\text{cm}^3$ with 1600, 1372 and 430 grid points. The slot width and fuel jet velocity are, respectively, 1.92 mm and 347 $\text{m s}^{-1}$ and the jet Reynolds number is 11 000. Based on local turbulence intensity and length scale, the turbulent Reynolds number is 360 at the 1/4th streamwise location along the jet centerline. To facilitate the simulation, the central hydrogen/nitrogen jet is ignited by artificially imposing a high-temperature region in the central jet. Based on the prescribed inlet jet velocity and the streamwise domain length, a flow-through time is approximately 0.07 ms. The solution was advanced at a constant time step of 4 ns through 12 flow-through times to provide stationary statistics. The simulation was performed on the 50 Tflop Cray XT3 at Oak Ridge National Laboratories and required 3.5 million CPU-hours running for 10 days on about 10 000 processors.

### *6.3. Results and discussion*

The global structure of the flame stabilization base is revealed from instantaneous images of the flame structure at different times [17]. Figure 10 shows a 3D volume rendering of the mass fraction of OH ($Y_{\text{OH}}$) and $HO_2$ ($Y_{\text{HO}_2}$) at $t$ = 0.40 ms. At first glance, one can see that fine flow structures upstream of the flame base are readily dissipated as the flow traverses downstream, primarily due to the effect of heat release by the flame. In addition, the flame base marked by OH isocontour is highly irregular and strongly affected by the instantaneous

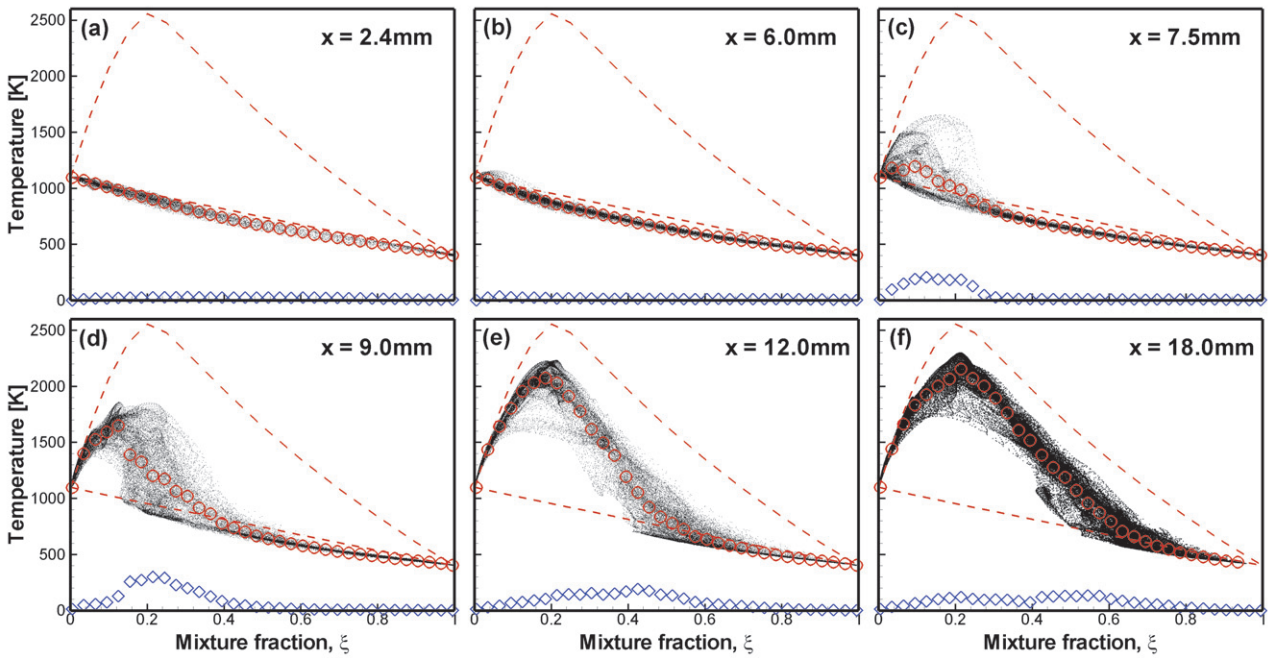


**Figure 11.** Temperature versus mixture fraction at different axial positions downstream of slot.

local flow and mixture conditions. Therefore, the stabilization of the lifted jet flame is a highly localized, and not a global phenomenon. Also note that $HO_2$ radical accumulates upstream of OH and other high-temperature radicals such as H and O, which are not shown here. $HO_2$ radical is a precursor of autoignition in hydrogen/air chemistry such that its existence upstream of other intermediate radicals provides strong evidence that the lifted flame base is stabilized by autoignition by heated coflow rather than by flame propagation.

To understand the stabilization mechanism and flame structure from the statistical point of view, scatter plots of temperature versus mixture fraction at different axial locations at $t = 0.42$ ms are presented in figure 11. Open circles and diamonds represent, respectively, the conditional mean and standard deviation of temperature. The frozen inflow and equilibrium temperature are also represented by dashed lines. Similar to the instantaneous plots, temperature first increases in a fuel-lean mixture, and subsequently the peak shifts toward richer mixtures, clearly indicating that ignition occurs first under hot, fuel-lean conditions where ignition delays are shorter. This has also been demonstrated in previous 2D DNS of auto-ignition in an inhomogeneous hydrogen/air mixture [26].

In summary, 3D DNS of a turbulent lifted hydrogen/air jet flame in an autoignitive heated coflow was performed using detailed chemistry and mixture-averaged transport properties. The results show that autoignition is the key mechanism responsible for flame stabilization, and $HO_2$ radical is important in initiating the autoignition ahead of the flame base. Autoignition is found to occur primarily in hot, fuel-lean regions where the scalar dissipation rate, or local mixing rate, is low and developing ignition kernels are sheltered from radical and heat losses.

## 7. Simulation of premixed combustion under intense turbulence

### *7.1. Introduction*

Premixed combustion under intense turbulence is of fundamental interest due to its relevance to practical applications such as lean premixed stationary gas turbines. Premixed flames under lean conditions tend to be thicker, propagate slower and the flame structure is more susceptible to the influence of turbulence. Recently, Peters [27] has provided a model for flame propagation in the regime where the turbulence scales are capable of penetrating and influencing the preheat zone, but are incapable of penetrating the reaction zones. This regime is called the thin reaction zones (TRZ) regime.

Premixed flame structure consists of a broader preheat layer upstream of a narrower reaction layer. Usual modeling approaches for the TRZ regime assume that the turbulent eddies can enter and influence the preheat layer—but the reaction zone, being thinner than the preheat layer by an order of magnitude, is not penetrated. The assumption of an order of magnitude disparity in the thickness of the preheat layer versus the reaction layer is based on theoretical analysis of the flame structure using activation energy asymptotics. In practical fuels such as methane–air, especially at lean conditions, the preheat layer is only approximately three times wider than the reaction layer. It needs to be determined if the reaction layer indeed stays intact in the TRZ regime despite the fact that it is not as thin as commonly assumed.

It is also not clear whether the flame dynamics in this regime are dominated by the entrainment of small eddies into the flame structure or by the large-scale flow straining which does not alter the flame structure. The penetration of the local flame structure by small eddies is expected to cause thickened flames in the TRZ regime [27, 28]. However, the competitive effect of the large-scale structure is expected to thin the flame. Thus, it is unclear whether flames are thicker or thinner in the TRZ regime. Several experimental studies in this regime have reported [29, 30] thicker flames, whereas others have reported thinner flames [31, 32]. Computations have also not yet delivered a definitive result. 1D linear eddy model computations [33] show thickened flames and 3D DNS [34] show there is some probability of encountering thicker flames, but more likely flames are thinner. 2D decaying turbulence simulations [35] and 3D expanding flame simulations [36] found that, on average, the flame gets thinner. These results, not only contradict the common theoretical description of the flame structure [28], but are also inconclusive due to the lack of realism in 2D turbulence and the lack of statistical stationarity in the expanding flame configuration.

Here, 3D fully resolved DNS of turbulent premixed combustion are performed in a spatially developing slot-burner Bunsen flame configuration with a detailed methane–air chemical mechanism. Three simulations in the TRZ regime at successively higher turbulence intensities have been performed as part of a parametric study. Data are analyzed to obtain statistical measures of the influence of turbulence on flame structure. In particular, the effect of turbulent strain on preheat layer thickness and the integrity of the reaction layers are studied.

### *7.2. Problem configuration*

The simulation was performed in a slot-burner Bunsen flame configuration. The slot-burner Bunsen configuration is especially interesting due to the presence of mean shear in the flow and is similar in configuration to the burner used in experimental studies, for example by Filatyev *et al* [37]. This configuration consists of a central reactant jet through which premixed reactants are supplied. The central jet is surrounded on either side by a heated coflow, whose composition and temperature are those of the complete combustion products of the reactant jet. This arrangement is similar to the pilot flame surrounding slot burners commonly used in experiments [37]. The reactant jet was chosen to be a premixed methane–air jet at 800 K and mixture equivalence ratio, $\phi = 0.7$. The unstrained laminar flame properties at these conditions computed using PREMIX [38] are as follows:

(a) flame speed, $S_L = 1.8\,\mathrm{m\,s^{-1}}$,

(b) thermal thickness based on maximum temperature gradient, $\delta_L = 0.3\,\mathrm{mm}$,

(c) full-width at half-maximum (FWHM) of heat release rate, $\delta_H = 0.14\,\mathrm{mm}$, and

(d) flame timescale, $\tau_f = \delta_L/S_L = 0.17\,\mathrm{ms}$.

One of the reasons for choosing a preheated inflow condition is that the cost of computation is inversely proportional to the Mach number at the inflow. Preheating the reactants leads to a higher flame speed and allows a higher inflow velocity without blowing out the flame. Also, many practical devices such as internal combustion engines, gas turbines and recirculating furnaces operate at highly preheated conditions. One important consequence of preheating is that the reaction zone is broadened at 800 K ($\delta_L/\delta_H = 2$) compared to 300 K ($\delta_L/\delta_H = 3$). However, the preheat temperature chosen here is low enough that flameless combustion does not occur.

A parametric study was performed to investigate the effect of increasing turbulence intensity on lean premixed combustion. The problem configuration, mixture equivalence ratio and temperature are the same for

**Table 1.** Simulation parameters.

| Slot width ($h$) | Case A 1.2 mm | Case B 1.2 mm | Case C 1.8 mm |
|---|---|---|---|
| Domain size in the streamwise, crosswise and spanwise directions | $12h \times 12h \times 3h$ | $20h \times 12h \times 3h$ | $20h \times 12h \times 3h$ |
| Number of grid points | 52 million | 88 million | 195 million |
| Turbulent jet velocity ($U$) | 60 m s$^{-1}$ | 100 m s$^{-1}$ | 100 m s$^{-1}$ |
| Laminar coflow velocity | 15 m s$^{-1}$ | 25 m s$^{-1}$ | 25 m s$^{-1}$ |
| Jet Reynolds number ($Re_{\text{jet}} = Uh/\nu$)[a] | 840 | 1400 | 2100 |
| Turbulence intensity[d] ($u'/S_{\text{L}}$) | 3 | 6 | 10 |
| Turbulence length scale[b,d] ($l_{\text{t}}/\delta_{\text{L}}$) | 0.7 | 1 | 1.5 |
| Integral length scale[c,d] ($l_{33}/\delta_{\text{L}}$) | 2 | 2 | 4 |
| Turbulence Reynolds number ($Re_{\text{t}} = u' l_{33}/\nu$)[a] | 40 | 75 | 250 |
| Karlovitz number ($\delta_{\text{L}}/l_{\text{k}})^2$ | 100 | 100 | 225 |
| Damkohler number ($S_{\text{L}} L_{\text{t}}/u' L$) | 0.23 | 0.17 | 0.15 |

[a] Kinematic viscosity at the inflow conditions, $\nu = 8.5e-5\,\text{m}^{-2}\,\text{s}^{-1}$, is used to compute Reynolds number.
[b] Turbulence length scale $l_{\text{t}}$ is estimated as $l_{\text{t}} = u'^3/\tilde{\epsilon}$, where $\tilde{\epsilon}$ is the average turbulent kinetic energy dissipation rate.
[c] Integral length scale $l_{33}$ is defined as the integral of the auto-correlation of the spanwise component of velocity in the spanwise direction.
[d] The turbulence scales evolve from the synthetic turbulence specified at the inflow. The $u'$, $l_{\text{t}}$ and $l_{33}$ values reported here are at the 1/4th streamwise location along the jet centerline.

all three simulations. However, they differ in the domain sizes and inflow turbulence conditions. The simulation parameters are given in table 1. A uniform grid spacing of 20 $\mu$m was used in the streamwise, $x$ and spanwise, $z$ directions, whereas an algebraically stretched mesh was used in the transverse, $y$ direction. Although the uniform grid spacing at the center of the jet ensures numerical fidelity and flexibility in post-processing, the boundaries are pushed farther away to reduce their influence on the flame.

### *7.3. Results and discussions*

A reaction progress variable, $c$ is defined such that it is a linear function of the mass fraction of $O_2$ and varies from $c = 0$ in the reactants to $c = 1$ in the products. Based on the laminar flame solution at the chosen reactant conditions, the heat release is a maximum at $c = 0.65$. Therefore, the iso-surface corresponding to $c = 0.65$ is taken as the flame surface. Figure 12 shows the instantaneous flame surface for the three cases. In all three cases the flame is initially planar at the inlet, but is wrinkled within a short distance in the downstream direction. Also the scale of wrinkling increases in the downstream direction. Comparing the three cases, it is seen that the amount of wrinkling increases from case A to case C. Also there is a significantly higher amount of flame–flame interaction in cases B and C compared to case A. The flame–flame interaction leads to pinch off as evident in the far downstream location in case A. This effect is more pronounced in cases B and C where the pinch off is noticed further upstream. In case C, the pinch off and mutual annihilation of flame surface due to interaction are found to occur even at locations very close to the jet inlet. This shows that the flame–flame interaction is a dominant mechanism limiting the flame surface area generated by wrinkling due to turbulence. Another interesting observation is that the shape of the flame is mostly convex toward the products and forms sharper cusps towards the reactants. This is contrary to the expected behavior for a Huygens-type self-propagation and is evidence that the flame topology is strongly influenced by turbulent straining.

Here, the data from the DNS are analyzed to determine if, on average, the flame thickness increases or decreases relative to a laminar flame. The reciprocal of the magnitude of progress variable gradient, $1/|\nabla c|$ yields a flame thickness analogous to the definition used for the laminar thermal thickness $L$. $|\nabla c|$ is averaged over intervals of $c$ and compared with the unstrained laminar flame profile in figure 13. In [15], the conditional

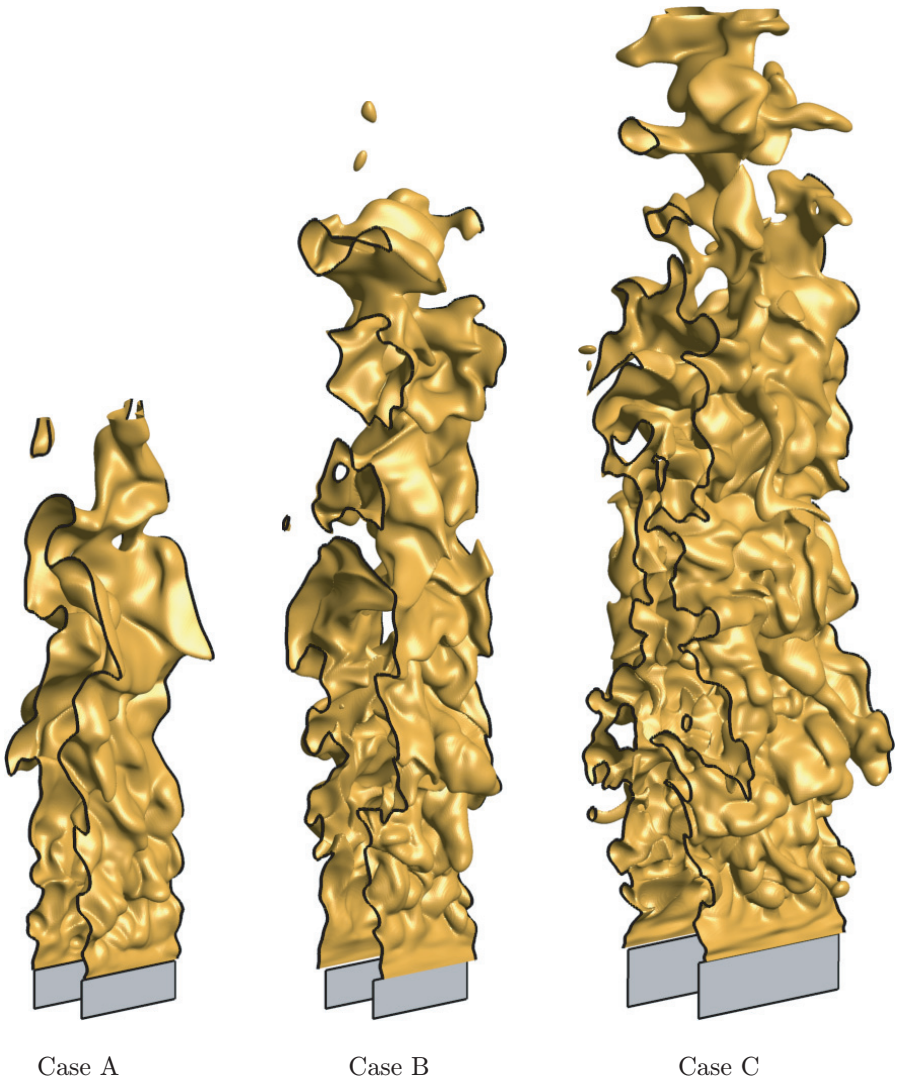


**Figure 12.** Instantaneous iso-contour of the progress variable ($c = 0.65$) representing flame surface for cases A, B and C.

mean of $|\nabla c|$ was presented for case A and the results showed that the mean $|\nabla c|$ was lower in the turbulent flame than in a laminar flame, which indicated flame thickening. Here, the same analysis is also applied to cases B and C, to verify if the flame continues to get thicker as the turbulence intensity is increased. A comparison of case A with case B in figure 13 shows that the mean gradients are further reduced. This again indicates an increase in flame thickening due to the increase in turbulence intensity from $u'/S_L$ between 3 and 6. However, a comparison of case B with case C shows that there is negligible increase in flame thickness even though the turbulence intensity was increased from $u'/S_L$ between 6 and 10. This is a very interesting result and shows that any further increase in turbulence intensity beyond a threshold level does not result in thicker flames.

## 8. Visualization for validation and discovery

Knowledge discovery from terabytes of raw data is a daunting task due to the sheer size and the complexity of the temporally evolving intermittent phenomena. Manipulating terabytes of data will undoubtedly stress the network and the storage infrastructure, and data in the future will inevitably increase in size as petascale leadership class computers become imminent in 2009. Moreover, the difficulty of knowledge discovery is also compounded by the complexity of the turbulent autoignition and flame phenomena being studied, and the complex relationship between several dozen reacting scalar fields together with the turbulence field.

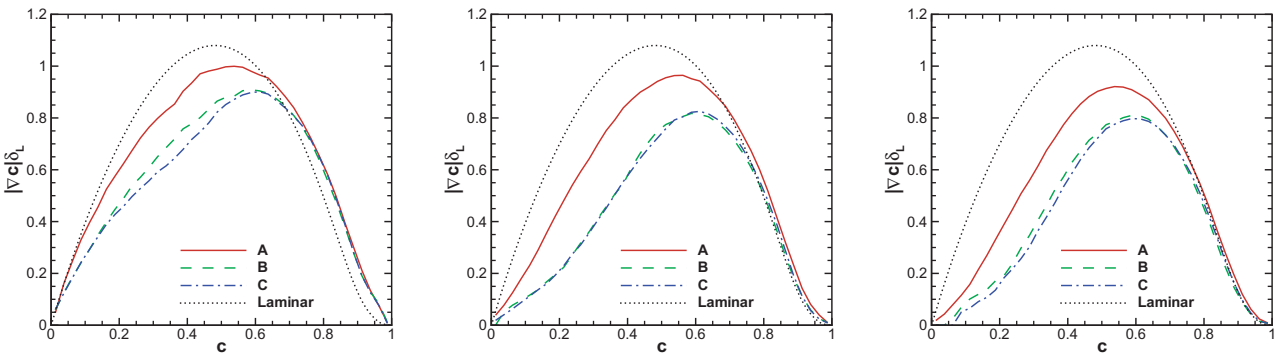


**Figure 13.** Conditional means of $|\nabla c|$ compared to the laminar flame profile for the three simulations. The means are computed at three chosen streamwise locations corresponding to $1/4$, $1/2$ and $3/4$th of the domain length in the stream-wise direction.

Researchers at the SciDAC Institute for Ultrascale Visualization and University of California at Davis (UC Davis) are developing new visualization technologies that will provide scientists with the tools to validate their simulations and to discover new phenomena from petabytes of simulated data [39].

### *8.1. Multivariate visualization*

To understand the correlation of scalar fields such as temperature, mixing rates and species concentrations in turbulent flames, we need the capability to visualize two or more scalars simultaneously. Conventional visualization tools do not directly support this need. The common practice is to make side-by-side comparisons of images of different variables by hand, which is tedious and time consuming. Furthermore, the information that can be derived by looking at separate images is quite limited. Thus, we need effective methods for simultaneously visualizing multiple time-varying variables from large data sets in an interactive fashion.

In many fields not limited to combustion, the capability to simultaneously visualize different variables that describe the same spatial domain and to determine their correlations is very desirable. In a multidisciplinary computing environment, for example, several engineering analysis programs such as structural and flow solvers run concurrently and cooperatively to perform a multidisciplinary design. The goal might be to identify the relevant design variables that can explain a particular phenomenon's causes, such as the effect of high strain rate or mixing rate on local flame extinction or stabilization. Other examples include the need to visualize data from multimodalities in medical imaging, and the desire to compare simulation and experimental data in many engineering design studies.

To visualize data from S3D, however, we are concerned only with rendering scalar volume data on the same mesh, which is a simpler problem than multimodality volume visualization in medical imaging or comparative visualization of simulated scientific data from different codes. The techniques we have developed can generate effective visualizations that reveal interaction and causal effect between different scalar properties of the same flow domain. Our approach to the simultaneous visualization problem is to use hardware-accelerated volume rendering, user-controlled data fusion and mixed rendering styles [40]. Interactive rendering lets users freely change rendering and visualization parameters, as well as data-fusion schemes. Rendering different volumes with different styles, if done appropriately, can enhance perception of shape, structure and spatial relationship. The data-fusion problem here is to determine how to display multiple data values defined at the same spatial location.

A few approaches prove effective for simultaneously visualizing an S3D simulation of a turbulent jet flame undergoing local extinction and reignition [40]. To understand the strong coupling between turbulence, the turbulent mixing of scalars such as temperature and species concentrations, and reaction, it is helpful to render the stoichiometric mixture fraction isoline overlaid on the scalar dissipation rate (that is, local mixing intensity) image and on the hydroxyl radical concentration image. Doing so will identify the actively burning flame surface relative to these other quantities.

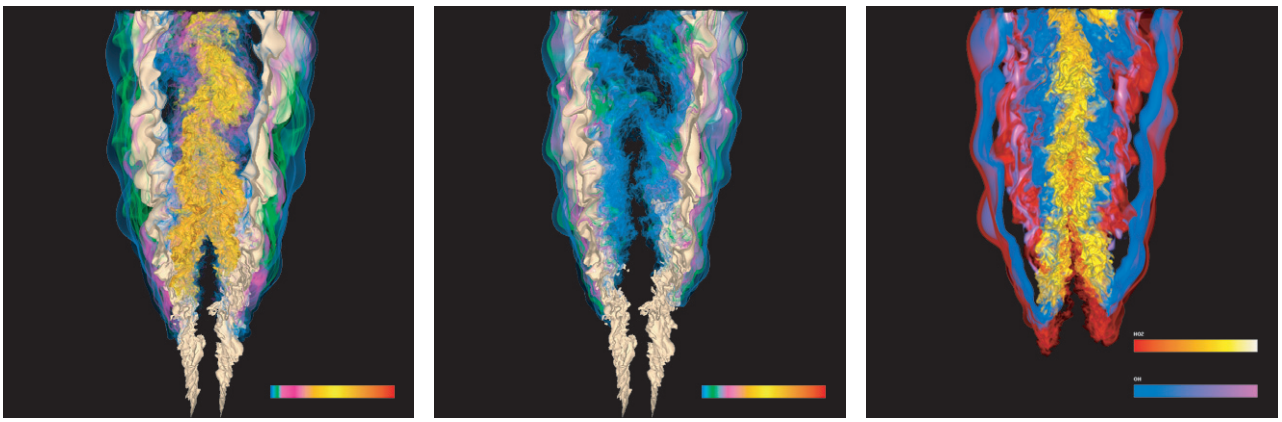

**Figure 14.** Simultaneous visualization two variables. Left: mixture fraction isosurface (gold) and $HO_2$. Middle: mixture fraction isosurface and OH. Right: OH and $HO_2$.

In another simulation using S3D for modeling an autoignitive lifted turbulent hydrogen/air jet flame, simultaneous visualization of the stoichiometric mixture fraction isosurface and the hydroperoxy radical ($HO_2$) shows that autoignition occurs in fuel-lean mixtures where the temperature is higher, the mixing rates are lower and ignition delay times are shorter. $HO_2$ is a good marker of autoignition regions preceding the high-temperature lifted flame base as discussed earlier. The image sequence generated with the simultaneous visualization techniques shows unambiguously that the $H_2$/air flame in heated coflow is stabilized by autoignition. The simultaneous visualization of these two key quantities in non-premixed autoignitive combustion are necessary to determine the spatial relationship between localized autoignition and the location of the high-temperature flame base which resides downstream of the $HO_2$ and at the stoichiometric mixture fraction where the ideal proportions of fuel and oxidizer are brought together. Figure 14 displays selected simultaneous visualization examples.

### *8.2. Visualization interface*

The ability to simultaneously visualize multiple variables is very powerful. However, the relationship between these variables is often nontrivial, requiring repeated exploration in the data's temporal, spatial and variable domains. An easy-to-use, interactive interface for specifying different combinations of multivariate data visualization is therefore desirable. We have developed an interface based on parallel coordinates and time histograms [41] that effectively supports time-varying multivariate feature extraction. Figure 15 shows the visualization interface. In the parallel coordinate interface, selected points in the three-variable space are represented as polylines with vertices on the parallel line segments. The parallel-coordinate interface can thus show the relationships between two or more variables or time steps. The time-histogram interface displays each variable's temporal characteristic and helps users identify time steps of interest. Together with volume, surface and cutting-plane visualizations, these different views of the data are tightly coupled to facilitate trispace data exploration [42].

The above trispace visualization interface design is being deployed in a desktop setting. We will extend this interface design to support animating features of interest enabling scientists to track the evolution of features of interest—e.g. ignition kernels, extinction holes—in time. In particular, the new system will make it possible to produce animations with much greater ease.

### *8.3. In-situ visualization*

For extreme-scale combustion simulations, massive I/O requirements render traditional feature extraction and tracking approaches unusable. The source data sets are typically comprised hundreds of instantaneous snapshots, presently each 50–150 GB in size, containing evolution data for dozens of 3D scalar and vector field components (fluid velocity, molecular species concentrations, temperature, density, etc). Moreover, the

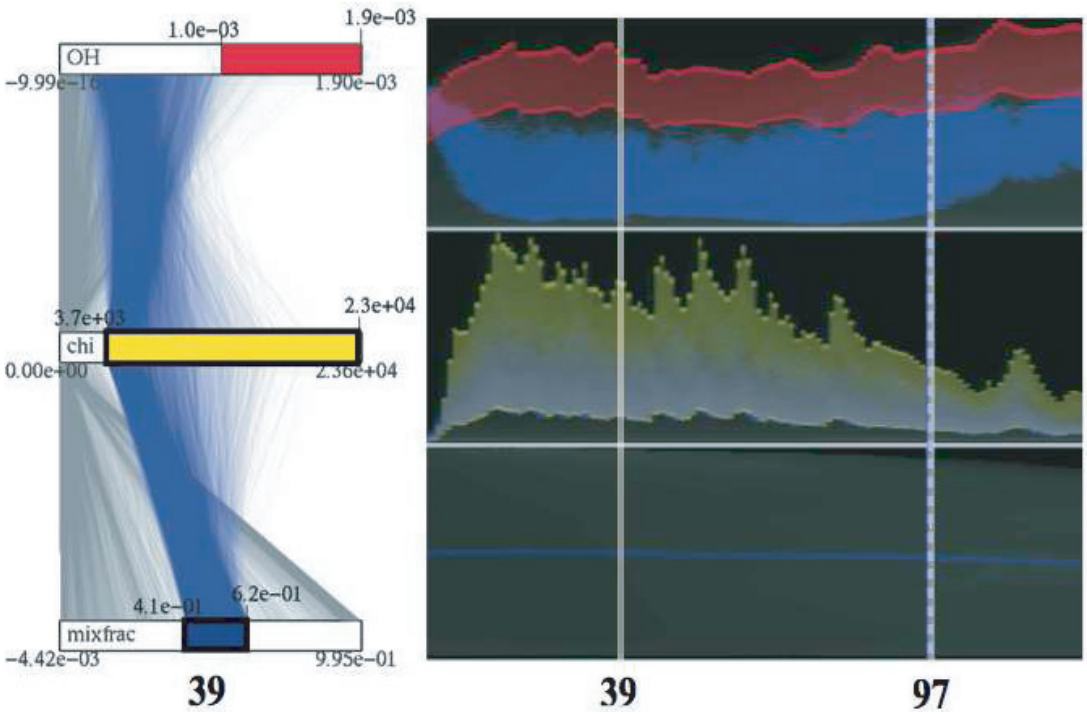


**Figure 15.** Interactive visualization to find negative spatial correlation between variables $\chi$ and OH near the isosurface of mixture fraction (mixfrac) over time. Three user-interface components are shown including the multivariate view for timestep 39, the temporal view for three variables where the $X$- and $Y$-axes correspond to time step and data value, respectively, and the spatial views for time steps 39 and 97. In the multivariate view, the transfer function widgets of *chi* and *mixfrac* are used as the brushing tool, thus highlighting voxels of yellow thin flow features shown in the spatial view near the blue isosurface. The temporal view shows the histogram of the voxels over time in blue as well as the temporal transfer functions.

features are typically specified in terms of data such as kinetic rates, thermodynamic quantities or transport characteristics, which are derived from the original data. To ensure the fidelity of these derived quantities, a significant portion of the solver used to generate the data is reused, e.g. chemistry, thermodynamic, transport properties, derivative stencils, interpolants, etc. Data sets of this magnitude, even when limited to just the primitive state data, cannot be loaded en masse into computer memory prior to analysis; the data must be split into subsets and staged for processing. Furthermore, the data are typically archived to mass-storage systems shortly after being written to disk by the simulation software in order to make space for the subsequent time snapshots. Retrieving simulation data for further analysis therefore must include transferring the necessary files from the mass-storage device onto active disks. All these challenges strongly suggest an *in situ* approach.

Investigations have begun on how to conduct *in situ* feature extraction and visualization for S3D. Compared with a traditional visualization task that is performed in a post-processing fashion, *in situ* visualization brings some unique challenges [43]. First of all, the visualization code must interact directly with the simulation code, which requires both the scientist and the visualization specialist to commit to this integration effort. To optimize memory usage, strategies need to be developed for simulation and visualization codes to share the same data structures. Secondly, visualization load balancing is more difficult to achieve since the visualization has to comply with the simulation architecture and builds tightly coupled with it. Unlike parallelizing visualization algorithms for standalone processing where we can partition and distribute data best suited for the visualization calculations, for *in situ* visualization, data partition and distribution is dictated by the simulation code. Moving data frequently among processors is not an option for visualization processing. We need to retool to balance the visualization workload so the visualization is at least as scalable as the simulation. Finally, the visualization calculations must be low cost, with decoupled I/O for delivering the rendering results, while the simulation is running. Since the visualization calculations on the supercomputer

cannot be hardware accelerated, alternative methods must be developed to simplify the calculations such that adding visualization would present only a small overhead on top of the cost of the simulation.

Further studies are required for *in situ* processing and visualization to understand the impact of this approach on S3D, subsequent visualization tasks, and the entire scientific discovery process employing extreme-scale computing.

## 9. Workflow for S3D

The terascale S3D DNS simulations enabled by large allocations of computing time through the DOE INCITE awards on leadership class computers are producing 30–130 TB of data per simulation, creating a significant challenge for data management, movement and analysis and visualization. We believe workflow automation will help us maintain our scientific productivity as we move toward petascale computing.

The overall goal of automating the workflow for monitoring the S3D code is to reduce manpower cost of managing large and complex data generated by this code. Currently, the S3D code is producing three types of data files/data types at various data generation rates. These files are: (i) restart files which contain the bulk of the analysis data, (ii) netcdf analysis files [44], used to monitor specific quantities from the S3D codes, more frequently than the restart files, and finally (iii) ASCII files which contain both the minimum and maximum volume average data from the S3D code. These data must be archived to local disk, morphed from N-files for the restarts, to M-files, moved over from ORNL to Sandia for local data analysis, and archived to the High Performance Storage System [45] at ORNL. Furthermore, we would like to visualize the fields in the netcdf files, plotting both *xy*-plots and colormap/contour plots. Since the data that are generated by the S3D code can potentially be extremely large, we must use effective ways to move data around, using secure methods. We must also be able to track the provenance of these files, so that we can recover from failures across the wide area network.

The Kepler [46] workflow system is adopted to automate the S3D workflow. Kepler is an open-source scientific workflow system developed with the collaboration of several projects of different scientific and technology areas. Kepler is based on Ptolemy II [47], a modeling tool for heterogeneous, concurrent systems. The main advantage of Ptolemy II lies in a modeling and design paradigm called actor-oriented modeling [48, 49], in which the data-centric computational components (actors) and the control-flow centric scheduling and activation mechanisms (frameworks) are clearly separated, an essential way to deal with the complex design issues of scientific workflows. In Kepler, a scientific workflow is viewed as a graph of independent components called actors where the edges denote the communication links between them, and parameters to customize their behavior. The execution semantics of the graph of connected actors is specified by a separate director, which defines how actors are executed and how they communicate with one another. A Kepler actor itself can be a nested workflow, even under the control of a different director than the top level. During workflow execution, tokens, encapsulating data, flow among the actors according to the schedule determined by the director. Workflows, exploiting the functionality of the underlying workflow system, provide many advantages beyond simply automating the mundane tasks like archiving data; they allow for real-time monitoring of simulations in progress, and they can track the provenance of the data and the workflow in real-time. We choose to use a workflow instead of a scripting language such as Python because of its ability to deal with concurrent and pipelined operations, such as graphing a particular file, at the same time as another actor is transferring the next file. Kepler is also being extended to support the integration of provenance tracking, for the workflow as well as for the data, which can greatly help the scientist to find the original data sets contributing to a particular image or to compare different experiments. Scientific workflows can exhibit different features, i.e. service orientation and data analysis, user interaction and leadership class computing. Each of these different classes of workflows attempts to highlight different aspects and technical capabilities of scientific workflows.

In our case, the main aspects of that we will use come from the Center for Plasma Edge Simulations (CPES) SciDAC project [50]. In this project, we have developed a custom set of actors in Kepler to solve common tasks in high-performance computing within scientific workflows, which are used here to work with the S3D workflow as well. One of the main actors that we use in the S3D-monitoring workflow is the ProcessFile actor that models the execution of an operation on a remote file as a (sub-) workflow itself. It uses an ssh connection to execute a command on a remote system, parameterized by the filenames, which are

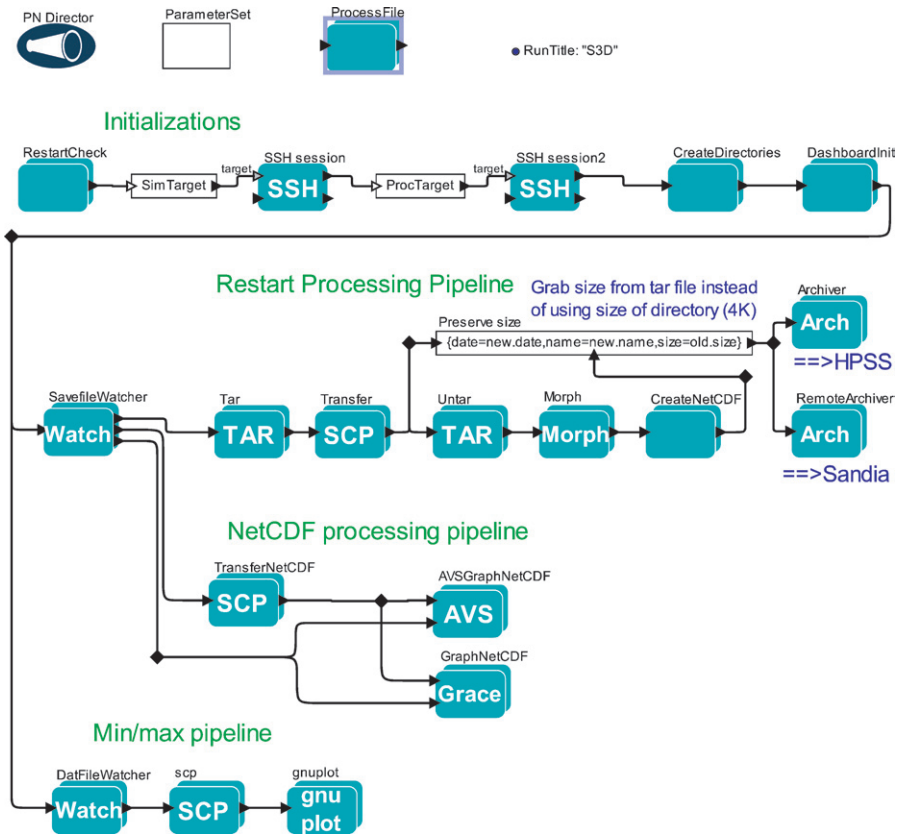


**Figure 16.** S3D Kepler workflow.

passed along through the workflow. It also keeps a checkpoint on the successfully executed actions, writes operation errors into log files and lists the result of the operation. It is designed in a way that long pipelines of file processing can be constructed conveniently, just varying the actual commands for each stage. In case the workflow is stopped and restarted, the automatic check pointing within this actor allows the workflow to skip steps that had already been accomplished, while retrying the failed ones; without any further need in the workflow construction to deal with the restarts and thus, keeping the workflow design clean and straightforwardly simple. If there are failures during the data processing, the workflow can be restarted later and the files that were not processed can continue, even after the simulation. This form of fault tolerance allows the users to concentrate on the science and not the mundane task of keeping track of the file movements manually. The commands executed remotely range from utilities like tar and scp to Python scripts created to take care of various aspects of the processing, thus combining the advantages of both the graphical workflows to express concurrency, task and data dependency and scripting for inherently sequential tasks that would be too awkward to implement as a workflow or just unnecessary duplication of work already existing in the form of a script. Another important actor is the FileWatcher, which is a generic component to regularly check a remote directory for new or modified files, and thus creates an indirect connection between the simulation code and the workflow, where the former's output drives the activity of the latter. The use of the ProcessFile actor with its restart and retry capabilities and the FileWatcher actor allows the workflow to stay isolated from the simulation, to keep up with the simulation and handle erroneous operations while not to have any effect on the simulation even if the workflow fails; which is critical for S3D simulations that can consume millions of CPU hours in a single simulation. We run the S3D code on the Cray XT4 system jaguar.ccs.ornl.gov at ORNL. All data are transferred to an Intel Infiniband cluster, ewok.ccs.ornl.gov, for morphing, imaging and archival. Each of these systems runs the PBS batch system, supports both interactive and batch nodes and has a lustre parallel file system, which has the capability of writing data up to 3 GB $s^{-1}$. Data are also transferred to another cluster at Sandia, for later (post-execution) analysis. Finally, data are backed up to the High Performance Storage System at ORNL. The S3D Kepler workflow is shown in figure 16.

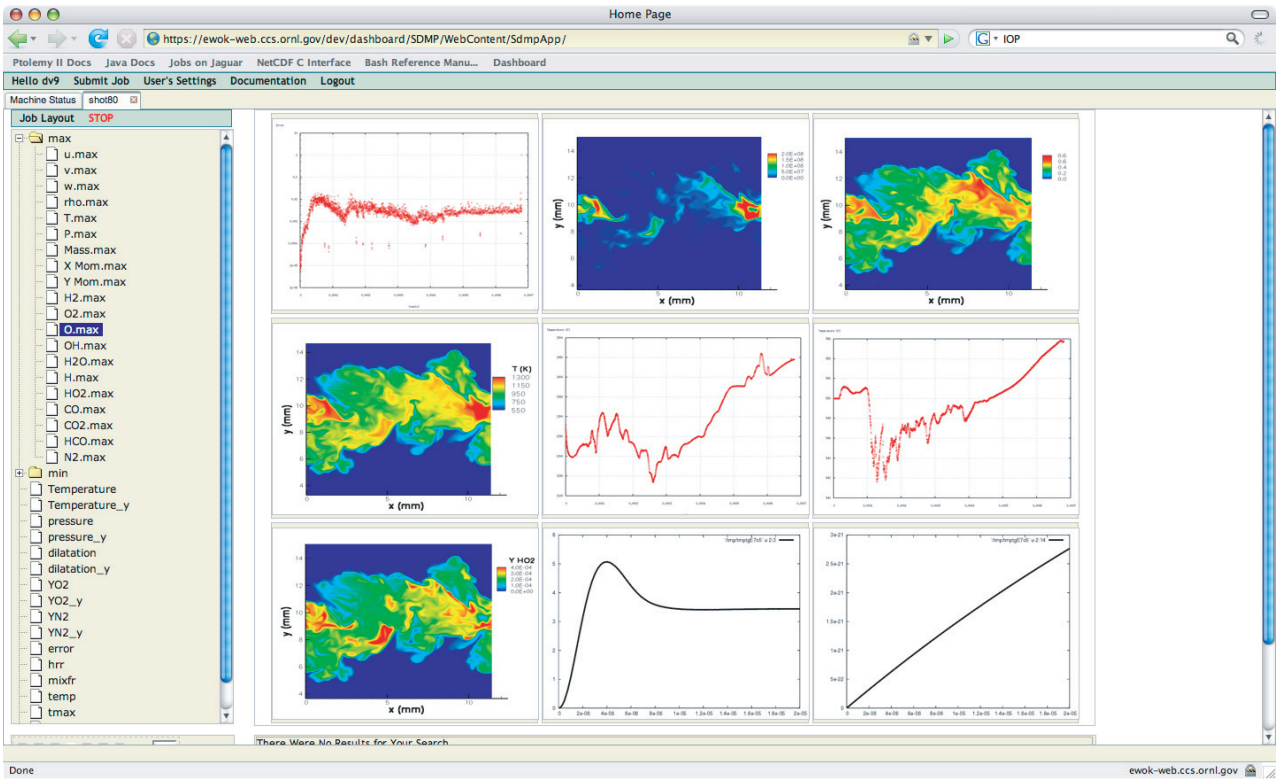


**Figure 17.** Interactive monitoring of the S3D workflow.

There are three pipelines in the workflow, executed in parallel: the restart/analysis, the netcdf file transformation and the min/max log pipelines. S3D produces restart files approximately every hour. The restart files consist of several files in a separate directory on jaguar. These files are transferred to ewok via multiple ssh connections, which can effectively move the data at 100 MB $s^{-1}$. In theory, we can move the data over to ewok at 350 MB $s^{-1}$ if we further parallelize the data moving process. In order to move the files only when they are fully written to the disk, the workflow watches a log file generated by S3D for an entry indicating that the output for that timestep is complete. Once the files are transferred to the lustre file system on ewok, the workflow morphs these files into a smaller number of files, so that the S3D analysis can be done on a smaller number of files. These files are simultaneously transferred to both the HPSS system at ORNL, and over to the cluster at Sandia, where we can later analyze the results. One of the last additions that we are providing is to transform the files into netcdf files on ewok, and then move them to our visualization partners at UC Davis. The next pipeline is for processing the netcdf files. A new netcdf file is potentially created much more often than the restart files. Each netcdf file contains both 2D slices, and 1D slices from the 3D data set. The netcdf file is transferred over to ewok, and then a file watcher actor calls another actor which generates images using Grace [51]. This library can be used for the $xy$ plots only, therefore we use a service-oriented visualization routine developed in AVS/Express [52] to visualize the colormap/contour results. This provides us with the additional flexibility that we can perform 3D visualization on the fly, and output images. The AVS/Express service is launched on ewok as a batch job. The job submission script writes down the location of the server so that we can communicate to the AVS/Express service through a socket connection. Each variable from the netcdf file is either processed in the Grace portion of the pipeline or the AVS/Express portion. Furthermore, the min and max of each variable of the netcdf file is calculated and transferred to a file that will be read in by the dashboard. The final portion of the workflow is to take the ASCII files for both the min and max of several S3D variables and move this file over to ewok so that the quantities can be included in the min and max files for the dashboard. We also run gnuplot at every instance, so that we can generate an $XY$ plot of the min and max of each variable. This permits us to log onto our web dashboard, described below, and visualize the results during the execution of S3D on the leadership class computer as presented in figure 17.

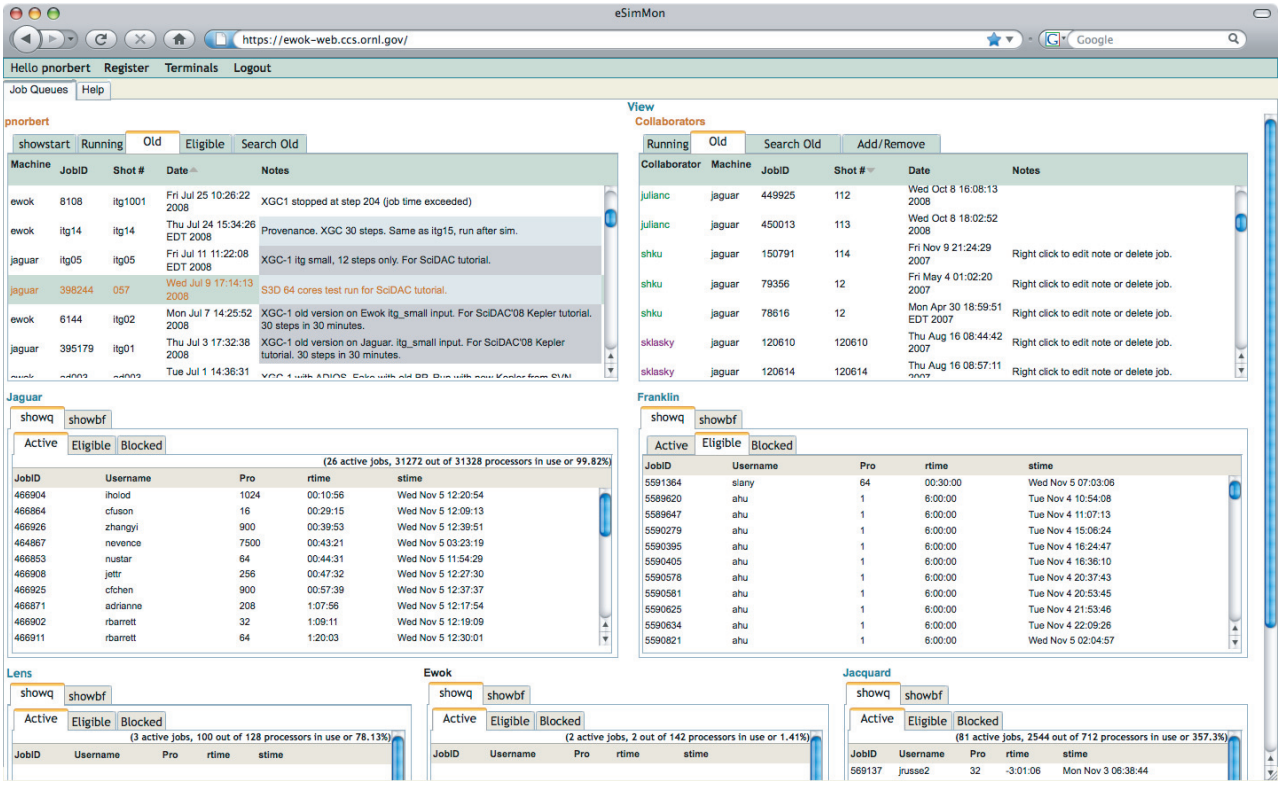


**Figure 18.** Monitoring jobs on DOE machines.

Figures 17 and 18 show two aspects of our web-2 based dashboard. Figure 17 displays interactive monitoring of maximum and minimum time traces of simulation variables and 2D isocontour plots of variables whereas figure 18 displays active monitoring of jobs on various DOE machines. Users must use their One Time Password (OTP) to log onto the dashboard. They are then presented with a display (see figure 18) which shows the jobs running at both ORNL and NERSC (if they have an account at NERSC). We choose to use Asynchronous Javascript And Xml (AJAX) to program the dashboard, so that each machine can update independently from one another. The dashboard communicates through a MySQL database running on one of the ewok nodes, to update all of the quantities. Users can see all of their running jobs on the lower right window on the dashboard. When one clicks on a job ID belonging to an S3D job the simulation monitoring page is presented as the one shown in figure 17. The user sees a tree of their data, and can move the name over the boxes, to visualize the results. The results are updated whenever the workflow creates new images from the newer time steps of the simulation. The user can also animate each window separately. We are in the process of moving the monitoring part of the dashboard over to FLASH [53]. This will allow us to control the memory caching of each image better, and allow us to seamlessly move from image deliver to movie delivery. The movies can be created once the simulation is over, to allow users to interact with the dashboard. Since all of the information on the dashboard is located in a database, we are allowing the users to annotate each image, so that users can type notes for each run. Further changes in the dashboard will allow the user to track both the workflow and data provenance during and after the simulation.

## 10. Concluding remarks

Terascale combustion simulations of simple laboratory-scale flames have been performed on CrayXT machines. These simulations provide fundamental insight into the 'chemistry-turbulence' interactions required to develop and validate predictive coarse-grained models used to design practical combustion devices. The path forward to petascale computing and beyond will enable exploration of an even wider range of scales—both

for turbulence and flame—at thermo-chemical conditions closer to practical combustion regimes in engines. Performance monitoring and optimization will become even more important if science applications are to scale, as future architectures are likely to have large numbers of multi-core processors and deep memory hierarchies. Careful management of cache will become even more important. Finally, analysis and visualization of 100s of terabytes of raw time-varying data from the simulations will require the development of scaleable data reduction strategies including distributed feature segmentation and tracking algorithms, and the ability to interactively query the data. The scaleable implementation of these algorithms will likely depend on future compute and analysis architectures and rely on close collaboration between computer and computational scientists.

## Acknowledgments

University of Oregon research was sponsored by contracts DE-FG02-07ER25826 and DE-FG02-05ER25680 from the MICS program of the U.S. Department of Energy (DOE), Office of Science, Rice University research was sponsored in part by the DOE's Office of Science under cooperative agreements DE-FC02-07ER25800 and DE-FC02-06ER25762, Northwestern University research was sponsored in part by DOE SCIDAC-2: Scientific Data Management Center for Enabling Technologies (SDM) grants DE-FC02-07ER25808, DE-FC02-01ER25485, and NSF SDCI OCI-0724599 and in part by the National Science Foundation through TeraGrid resources provided by NCSA, and University of California at Davis research was sponsored in part by the U.S. National Science Foundation through grant OCI-0325934 and the U.S. DOE through the SciDAC program with agreement no DE-FC02-06ER25777. Part of this work was performed under the auspices of the U.S. DOE by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Sandia National Laboratories (SNL) research was sponsored by the Division of Chemical Sciences, Geosciences, and Biosciences, Office of Basic Energy Sciences of the U.S. DOE, and the U.S. DOE SciDAC Program. SNL is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. DOE under contract DE-AC04-94AL85000. Oak Ridge National Laboratory (ORNL) was supported by and this research used resources of the National Center for Computational Sciences (NCCS) at ORNL, which is supported by the Office of Science of the U.S. DOE under contract DE-AC05-00OR22725 and in part by a SCIDAC-2 SDM grant DE-FC02-01ER25486.

## References

[1] Hawkes E R, Sankaran R, Sutherland J C and Chen J H 2005 Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models *J. Phys.: Conf. Ser.* **16** 65–79
[2] Hirschfelder J O, Curtiss C F and Bird B R 1965 *Molecular Theory of Gases and Liquids* (New York: Wiley)
[3] Bird R B, Stewart W E and Lightfoot E N 1960 *Transport Phenomena* (New York: Wiley)
[4] Giovangigli V 1999 *Multicomponent Flow Modeling* (Boston, MA: Birkhauser)
[5] Warnatz J, Maas U and Dibble R W 1996 *Combustion* (Berlin: Springer)
[6] Ern A and Giovangigli V 1998 Thermal diffusion effects in hydrogen–air and methane–air flames *Combust. Theory Modelling* **2** 349–72
[7] Dixon-Lewis G 1968 Flame structure and flame reaction kinetics *Proc. Roy. Soc. A* **307** 111–35
[8] Kennedy C A and Carpenter M H 1994 Several new numerical methods for compressible shear-layer simulations *Appl. Num. Math.* **14** 397–433
[9] Kennedy C A and Carpenter M H 2000 Low-storage, explicit Runge–Kutta schemes for the compressible Navier–Stokes equations *Appl. Numer. Math.* **35** 117–219
[10] Kee R J, Rupley F M, Meeks E and Miller J A 1996 Chemkin-III: a Fortran chemical kinetic package for the analysis of gas-phase chemical and plasma kinetics *Technical Report SAND96-8216* Sandia National Laboratories
[11] Kee R J, Dixon-Lewis G, Warnatz J, Coltrin M E and Miller J A 1986 A Fortran computer code package for the evaluation of gas-phase multicomponent transport properties *Technical Report SAND86-8246* Sandia National Laboratories
[12] Yoo C S, Wang Y, Trouve A and Im H G 2005 Characteristic boundary conditions for direct numerical simulations of turbulent counterflow flames *Combust. Theory Modelling* **9** 617–46

[13] Yoo C S and Im H G 2007 Characteristic boundary conditions for simulations of compressible reacting flows with multi-dimensional, viscous and reaction effects *Combust. Theory Modelling* **11** 259–86
[14] Sankaran R, Hawkes E R and Chen J H 2006 Direct numerical simulations of turbulent lean premixed combustion *J. Phys.: Conf. Ser.* **46** 38–42
[15] Sankaran R, Hawkes E R, Chen J H, Lu T and Law C K 2007 Structure of a spatially-developing turbulent lean methane–air bunsen flame *Proc. Combust. Inst.* **31** 1291–8
[16] Hawkes E R, Sankaran R and Chen J H 2007 Scalar mixing in direct numerical simulations of temporally evolving plane jet flames with skeletal $CO/H_2$ kinetics *Proc. Combust. Inst.* **31** 1633–40
[17] Yoo C S, Sankaran R and Chen J H Three-dimensional direct numerical simlation of a turbulent lifted hydrogen/air jet flame in heated coflow. Part I: stabilization and structure *J. Fluid Mech.* submitted
[18] Shende S and Malony A D 2006 The TAU parallel performance system *Int. J. High Perform. Comput. Appl.* **20** 287–331
[19] Mellor-Crummey J, Tallent N, Fagan M and Odegard J 2007 Application performance profiling on the Cray XD1 using HPCToolkit *Proc. Cray User's Group Meeting* (Seattle, WA) Available at http://www.cs.rice.edu/∼johnmc/papers/hpctoolkit-cug-2007.pdf.
[20] Qasem A, Jin G and Mellor-Crummey J 2008 Improving performance with integrated program transformations *Technical Report* TR03-419, Department of Computer Science, Rice University, October 2003
[21] Lyons K M 2007 Toward an understanding of the stabilization mechanisms of lifted turbulent jet flames: experiments *Progr. Energy Combust. Sci.* **33** 211–31
[22] Pitts W M 1988 Toward an understanding of the stabilization mechanisms of lifted turbulent jet flames: experiments *Proc. Combust. Inst.* **22** 809–16
[23] Peters N 2000 *Turbulent Combustion* (New York: Cambridge University Press)
[24] Pitts W M Simultaneous laser Raman–Rayleigh–LIF measurements and numerical modeling results of a lifted turbulent $H_2/N_2$ jet flame in a vitiated coflow *Proc. Combust. Inst.* **29** 901–9
[25] Pickett L M Low flame temperature limits for mixing-controlled diesel combustion *Proc. Combust. Inst.* **30** 2727–35
[26] Echekki T and Chan J H 2003 Direct numerical simulation of autoignition in nonhomogeneous hydrogen–air mixtures *Combust. Flame* **134** 169–91
[27] Peters N 1999 The turbulent burning velocity for large-scale and small-scale turbulence *J. Fluid Mech.* **384** 107–32
[28] Peters N 2000 *Turbulent Combustion* (New York: Cambridge University Press)
[29] Mansour M S, Peters N and Chen Y C 1998 Investigation of scalar mixing in the thin reaction zones regime using a simultaneous CH-LIF/Rayleigh laser technique *Proc. Combust. Inst.* **27** 767–73
[30] Chen Y C and Mansour M S 1998 Investigation of flame broadening in turbulent premixed flames in the thin reaction zones regime *Proc. Combust. Inst.* **27** 811–8
[31] Dinkelacker F, Soika A, Most D, Hofmann D, Leipertz A, Polifke W and Dobbeling K 1998 Structure of locally quenched highly turbulent lean premixed flames *Proc. Combust. Inst.* **27** 857–65
[32] Soika A, Dinkelacker F and Leipertz A 1998 Measurement of the resolved flame structure of turbulent premixed flames with constant reynolds number and varied stoichiometry *Proc. Combust. Inst.* **27** 785–92
[33] Sankaran V and Menon S 2000 Structure of premixed turbulent flames in the thin reaction zones regime *Proc. Combust. Inst.* **28** 203–9
[34] Tanahashi M, Nada Y, Ito Y and Miyauchi T 2002 Local flame structure in the well-stirred reactor regime *Proc. Combust. Inst.* **29** 2041–9
[35] Hawkes E R and Chen J H 2003 Direct numerical simulation of premixed flames in the thin reaction zones regime *Proc. 2003 Winter Meeting of the Western States Section of the Combustion Institute Paper No. 03F-68*
[36] Thevenin D 2005 Three-dimensional direct simulations and structure of expanding turbulent methane flames *Proc. Combust. Inst.* **30** 629–37
[37] Filatyev S A, Driscoll J F, Carter C D and Donbar J M 2005 Measured properties of turbulent premixed flames for model assessment, including burning velocities, stretch rates, and surface densities *Combust. Flame* **141** 1–21
[38] Kee R J, Grcar J F, Smooke M D and Miller J A 1985 PREMIX: a Fortran program for modeling steady laminar one-dimensional premixed flames *Technical Report SAND85-8240* Sandia National Laboratories
[39] Ma K L, Ross R, Huang J, Humphreys G, Max N, Moreland K, Owens J D and Shen H W 2007 Ultra-scale visualization: research and education *J. Phys.: Conf. Ser.* **78** 012088
[40] Akiba H, Ma K L, Chen J H and Hawkes E R 2007 Visualizing multivariate volume data from turbulent combustion simulations *IEEE Comput. Sci. Eng.* p 86
[41] Akiba H, Fout N and Ma K L 2006 Simultaneous classification of time-varying volume data based on the time histogram *Proc. Eurographics Visualization Symp.* p 1

[42] Akiba H and Ma K L 2007 A tri-space visualization interface for analyzing time-varying multivariate volume data *Proc. Eurographics/IEEE VGTC Symp. on Visualization* p 115
[43] Ma K L, Wang C, Yu H and Tikhonova A 2007 *In-situ* processing and visualization for ultrascale simulations *J. Phys.: Conf. Ser.* **78** 012043
[44] Rew R and Davis G 1990 Netcdf: an interface for scientific data access *Comput. Graph. Appl.* **10** 76–82
[45] Teaff D, Watson R W and Coyne R A 1995 The architecture of the high performance storage system (HPSS) *Proc. 3rd Goddard Conf. on Mass Storage Systems and Technologies* (March 1995)
[46] Ludäcker B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee E A, Tao J and Zhao Y 2006 Scientific workflow management and the kepler system *Concurr. Comput.: Pract. Exp.* **18** 1039–65
[47] Eker J, Janneck J W, Lee E A, Liu J, Liu X, Ludvig J, Neuendorffer S, Sachs S and Xiong Y 2003 Taming heterogeneity—the ptolemy approach *Proc. IEEE*
[48] Lee E A and Neuendorffer S 2004 Actor-oriented models for codesign: balancing re-use and performance *Formal Methods and Models for System Design: A System Level Perspective*, pp 33–56
[49] Bowers S and Ludäscher B 2005 Actor-oriented design of scientific workflows *24th Int. Conf. on Conceptual Modeling* (*ER*) (*Klagenfurt, Austria, LNCS, 2005*)
[50] http://www.cims.nyu.edu/cpes/. Center for plasma edge simulations (CPES) SciDAC project
[51] http://plasma-gate.weizmann.ac.il/Grace/
[52] Vroom J 1995 AVS/Express: *A New Visual Programming Paradigm. Boston, MA: Proc. AVS*. Release 3.0
[53] Weil K, Weinman L and Green G 2001 *Flash 5 Hands-On Training* (Berkeley, CA: Peachpit Press)
[54] May J 2001 *Parallel* I/O *for High Performance Computing* (San Francisco, CA: Morgan Kaufmann)
[55] Tanenbaum A and van Steen M 2002 *Distributed Systems—Principles and Paradigms* (Englewood Cliffs, NJ: Prentice Hall)
[56] Liao W, Ching A, Coloma K, Nisar A, Choudhary A, Chen J, Sankaran R and Klasky S Using MPI file caching to improve parallel write performance for large-scale scientific applications *ACM/IEEE Conf. Supercomputing* (November 2007)
[57] Thakur R, Gropp W and Lusk E 1996 An abstract-device interface for implementing portable parallel-I/O interfaces *6th Symp. on the Frontiers of Massively Parallel Computation* (October 1996)
[58] Fineberg S, Wong P, Nitzberg B and Kuszmaul C 1996 PMPIO—a portable implementation of MPI-I/O*6th Symp. on the Frontiers of Massively Parallel Computation* (October 1996)