Communications in Applied Mathematics and Computational Science

ACHIEVING ALGORITHMIC RESILIENCE FOR TEMPORAL INTEGRATION THROUGH SPECTRAL DEFERRED CORRECTIONS

RAY W. GROUT, HEMANTH KOLLA, MICHAEL L. MINION AND JOHN B. BELL

vol. 12 no. 1 2017

msp

COMM. APP. MATH. AND COMP. SCI.
Vol. 12, No. 1, 2017
dx.doi.org/10.2140/camcos.2017.12.25

ACHIEVING ALGORITHMIC RESILIENCE FOR TEMPORAL INTEGRATION THROUGH SPECTRAL DEFERRED CORRECTIONS

RAY W. GROUT, HEMANTH KOLLA,
MICHAEL L. MINION AND JOHN B. BELL

Spectral deferred corrections (SDC) is an iterative approach for constructing higher-order-accurate numerical approximations of ordinary differential equations. SDC starts with an initial approximation of the solution defined at a set of Gaussian or spectral collocation nodes over a time interval and uses an iterative application of lower-order time discretizations applied to a correction equation to improve the solution at these nodes. Each deferred correction sweep increases the formal order of accuracy of the method up to the limit inherent in the accuracy defined by the collocation points. In this paper, we demonstrate that SDC is well suited to recovering from soft (transient) hardware faults in the data. A strategy where extra correction iterations are used to recover from soft errors and provide algorithmic resilience is proposed. Specifically, in this approach the iteration is continued until the residual (a measure of the error in the approximation) is small relative to the residual of the first correction iteration and changes slowly between successive iterations. We demonstrate the effectiveness of this strategy for both canonical test problems and a comprehensive situation involving a mature scientific application code that solves the reacting Navier–Stokes equations for combustion research.

1. Introduction

Since its introduction by Dutt et al. [12], the iterative nature of spectral deferred corrections (SDC) has been leveraged extensively to create efficient, high-accuracy methods for temporal integration tailored to specific types of problems. For example, in multi-implicit spectral deferred correction methods [3; 27], the terms in an advection-diffusion-reaction system are integrated separately with different time steps but coupled together using the SDC approach to achieve higher-order temporal accuracy than is achievable with traditional operator-splitting schemes. A similar approach is used to reduce splitting errors in a low-Mach combustion code by Nonaka et al. [32], where the SDC iterates are used to couple together interacting

MSC2010: primary 65D30, 65M12, 65M22, 80A25, 94B99; secondary 65M20.
Keywords: SDC, deferred correction, resilience, time integration, combustion.

physical processes. In this case, a significant advantage is realized in that the reduction in splitting error reduces nonphysical excursions into a chemical state space that artificially excites the intrinsic stiffness in the system. SDC has also been used to construct efficient time-parallel methods for partial differential equations (PDEs) [15]. Such desirable features that are not readily available in classical methods such as linear multistep or Runge–Kutta (RK) methods can make SDC an attractive choice for time integration despite the fact that SDC often requires relatively more function evaluations per time step.

There is growing concern about the impact of hardware errors — especially those that can lead to successful completion with erroneous results known as *silent data corruption*. This concern is driven by trends towards increasing concurrency as well as operation near design limits. Reducing voltage to improve energy efficiency has long been known to increase susceptibility to soft errors (e.g., [2; 10]). Further, modern designs tend to have elevated operating temperatures, which also increases the soft error rate [38; 9]. Wei et al. [41] define error resilience eloquently as *the ability of a program to prevent an error from becoming a silent data corruption*. We will look to leverage the iterative nature of SDC to provide algorithmic error resilience for temporal integration in the face of soft errors in the arithmetic operations and scratch variables used to update the solution. We expect that the iterative nature will be well suited to recover from transient errors. Chen et al. [7] note that an adaptive RK scheme, where the solution update is computed for two different time steps, should be able to detect soft faults as the two evaluations will be dramatically different if a soft fault has occurred. Benson et al. [1] constructed an error detector by evaluating an alternative time integration scheme (tailored for speed rather than stability, such as an embedded RK scheme of lower order, an explicit counterpart to a linear multistep method, or simple extrapolation) and examining the norm of the difference between the base and alternative schemes for anomalies in the context of a window of time steps. Benson et al. used the window of time steps because they noted that a hard threshold on the difference norm is meaningless because the expected norm of the difference changes with the solution. We use similar logic when inspecting the convergence rate between successive correction iterations to determine if the solution is acceptable.

The primary contributions of this paper are, firstly, to show that monitoring the residual in SDC correction sweeps can be used to detect soft (transient) errors resulting from hardware faults that could lead to silent data corruption using a reference integration algorithm and, secondly, to demonstrate the feasibility of recovering from soft errors by continuing SDC correction iterations. The intent of this paper is not to look at the details of low-level fault injection but rather at how a time integration algorithm can recover from those faults that migrate up the call tree through the return values of kernels. Here we use the term *kernel* to refer to

routines at the application level that compute terms in the governing differential equations being integrated. For example, the kernels from the application discussed in Section 2.4 are operations that compute advective or diffusive terms for the method-of-lines formulation or evaluate transport coefficients.

The remainder of this paper is organized as follows. In the next section, we present a brief outline of the SDC algorithm, relevant aspects of the state of research on fault injection and algorithmic resilience, and an overview of the combustion code used as an application benchmark later in the paper. We then turn in Section 3 to the behavior of the application in the context of single-occurrence synthetic errors, using the explicit Runge–Kutta integrator traditionally employed in the application code as a baseline to assess susceptibility to silent data corruption. We also examine the ability of the SDC algorithm to recover from such errors in an application test case and in a linear problem to demonstrate how the damping proceeds in a controlled setting. Finally, in Section 3.4, we look at a comprehensive error injection test case where we inject errors at an elevated rate into many runs of the application test case to see how our SDC iteration strategy narrows the distribution of the simulation output in a challenging scenario.

2. Preliminaries and related work

2.1. ***SDC formulation.*** Spectral deferred correction schemes were proposed by Dutt et al. [12] and subsequently developed significantly by Minion and colleagues (e.g., [31; 3; 28]). The basic approach is briefly recapped in this subsection before we consider additional aspects of its performance relevant to use in practical applications.

SDC schemes are based on recasting the ordinary differential equation (ODE)

$$\psi' = F(\psi, t), \qquad \psi(t_n) = \psi_n \tag{1}$$

over the time interval $t^n \le t \le t^{n+1}$ in integral form as

$$\psi(t) = \psi_n + \int_{t_n}^{t} F(\psi, \tau)\, d\tau. \tag{2}$$

Subdividing the interval $[t_n, t_{n+1}]$ by choosing $M+1$ Gauss–Lobatto quadrature nodes t_m ($t^n = t_0$ and $t^{n+1} = t_M$), for each node we can write the approximation

$$\phi_m = \psi_n + \Delta t \sum_{j=0}^{M} q_{m,j} F(\phi_j, t_j). \tag{3}$$

This integral provides an approximation to the solution $\psi_{n+1} \approx \phi_M$ at t^{n+1}; however, it effectively couples the solution at all of the quadrature nodes in the interval. Equation (3) is referred to as the collocation formulation (see, e.g., [20]) and

is equivalent to a fully implicit Runge–Kutta method with stages given by the quadrature nodes and coefficients in the Butcher tableau corresponding to the $q_{m,j}$. SDC can be thought of as providing an efficient iterative approach for computing the solution to this coupled system by iterative substepping over the nodes.

The basic idea is, given an approximate continuous solution $\phi^k(t)$, one can define a residual that measures the error in the approximation ϕ^k as

$$R(\phi^k, t) = \phi_n + \int_{t_n}^{t} F(\phi^k(\tau), \tau)\, d\tau - \phi^k(t). \tag{4}$$

If we define $c^k(t) = \phi(t) - \phi^k(t)$, then by substituting the definition of the residual into the integral form of the original equation, we obtain

$$c^k(t) = \int_{t^n}^{t} [F(\phi^k(\tau) + c^k(\tau), \tau) - F(\phi^k(\tau), \tau)]\, d\tau + R(\phi^k(t), t). \tag{5}$$

We then discretize this equation using the approximate residual

$$R_m(\phi^k) = \phi_n + \Delta t \sum_{j=0}^{M} q_{m,j} F(\phi_j^k, t_j) - \phi_m^k. \tag{6}$$

An explicit Euler-type method to discretize (5) gives the resulting update formula for the k-th iterate

$$c_{m+1}^k = c_m^k + \Delta t_m [F(\phi_m^{k+1}, t_m) - F(\phi_m^k, t_m)] + R_{m+1}(\phi^k) - R_m(\phi^k) \tag{7}$$

or, in a direct update form for $\phi_m^{k+1} = \phi_m^k + c_m^k$,

$$\phi_{m+1}^{k+1} = \phi_m^{k+1} + \Delta t_m [F(\phi_m^{k+1}, t_m) - F(\phi_m^k, t_m)] + I_m^{m+1}(\phi^k), \tag{8}$$

where

$$I_m^{m+1}(\phi^k) = \Delta t \sum_{j=1}^{M} (q_{m+1,j} - q_{m,j}) F(\phi^k(t_l), t_l) \approx \int_{t_m}^{t_{m+1}} F(\phi^k(\tau), \tau)\, d\tau. \tag{9}$$

I_m^{m+1} is the equivalent to the integral of the polynomial interpolant of ϕ^k over the interval $[t_m, t_{m+1}]$. Each such iteration can improve by one the formal order of accuracy of the approximate solution up to the order of the underlying quadrature. In the case of $M+1$ Lobatto nodes, the method achieves order $2M$ of convergence.

In the numerical results, we focus on SDC methods using three Lobatto nodes. If the initial value at all three nodes is taken to be the value at t_n and four correction iterations as described by (8) are performed, then the resulting method is formally fourth-order accurate. This is the same order of accuracy as the collocation or fully implicit Runge–Kutta method using the same three nodes, but the two schemes are not identical. In fact, the fourth-order method with four iterations can be considered

an explicit Runge–Kutta method with eight stages (see, e.g., [8]). Additional SDC iterations will not raise the formal order of accuracy but will drive the numerical solution to that from the collocation formulation with linear convergence with a rate proportional to the time step.

2.2. ***Soft error fault injection.*** For the purpose of this study, we follow the taxonomy of Bridges et al. [4], wherein hard faults are those that cause program interruptions and clearly denote an incomplete program execution while soft faults are typically observed as random bit flips, where one or more bits of memory are reversed. These faults are transient and do not indicate hardware damage, as opposed to persistent faults such as bits that are immutable due to a physical defect ("stuck bit" errors). Depending on where in the memory hierarchy they occur and the robustness of the algorithm, soft faults may not always lead to a solution failure but might result in an erroneous solution despite completely evading detection [14]. It might be acceptably inexpensive to provide soft fault detection and correction mechanisms for some, but not all, memory levels. For instance, error correction codes have been shown to correct a majority of soft faults in main memory [38] while processor registers are difficult to protect from soft faults [24]. Many factors such as altitude, age, temperature, and utilization are thought to affect error rates in real machines with a significant variability observed across various DRAM vendors. Recent studies have attempted to characterize and quantify error rates by surveying hardware logs from real machines, although a consensus is far from apparent. Schroeder et al. [35] study error rates from commodity clusters in Google's server fleet and observe that a majority of the errors are hard errors and soft errors are far less probable (a soft error probability of $\sim 2\%$ for every hard error). On the other hand, Sridharan et al. [39] find the opposite to be the case from a survey of data from two high-performance computing systems: Cielo at Los Alamos National Laboratory and Jaguar at Oak Ridge National Laboratory. Nonetheless, the most dominant mode seems to be single-bit errors (60%) with hard and soft errors being approximately equiprobable.

Considering the various enmeshed layers of software and hardware, the propagation of soft faults from one layer to another can be complicated to model. Strictly speaking, a bit flip at the level of hardware instructions is unlikely to migrate up to the application level as a single bit flip after several operations have been performed on the data. Even near the hardware level, a single bit flip in an instruction input might result in multiple bit flips in the destination register [14]. Despite this, there is some evidence that injecting single bit flips at higher levels produces similar effects from an application perspective as injecting errors near the hardware level. We choose this approach because it allows us to reason about the algorithmic sensitivity to the errors while eliminating the potentially confounding effects of interaction of the errors before reaching the application level.

Wei et al. compare the behavior of high-level fault injection (implemented at the LLVM intermediate representation (IR) level) to low-level fault injection (using Intel Pin tools) and find that, while there were significant differences in the number of program crashes between the two techniques, the IR-level fault injection is effective for assessing the impact of soft faults that result in silent data corruption. Wei et al. [41] also note that it is an established de facto standard that single bit flips [16] are an appropriate approach. In a related issue, Fang et al. [16] look at the effect of fault injection on multithreaded programs implemented using OpenMP and consider the sensitivity of the thread where the faults are injected due to the emphasis of the master thread on problem setup/teardown (phases of their chosen benchmarks that are particularly prone to resulting in ultimate silent data corruption in the output from fault injection). In our present application of interest, the setup/teardown phases are a very small portion of the overall run time, and otherwise the application follows a bulk-synchronous model.

Since our focus here is on the algorithmic robustness of SDC, we adopt a simple fault injection model. Considering that processor registers and arithmetic lookup units (ALUs) and floating point units (FPUs) are the most vulnerable to soft faults [41], we model soft faults as single bit flips in processor registers. However, we inject errors at the level of the application rather than at or very near the hardware level. We adopt an approach similar to, but even closer to the application level than, that of Wei et al. [41] and inject faults as if they manifest as single bit flips in register work arrays of the application level kernels that evaluate the terms contributing to the time derivative (F) of our system of ODEs.

2.3. ***Algorithmic approaches to resilience.*** Since a large number of scientific applications employ linear system solvers, methods to incorporate resilience in iterative linear solver algorithms have received wide attention. For example, Heroux and Bridges et al. [23; 4] propose a *fault-tolerant* version of the generalized minimal residual method (FT-GMRES) whereby the inner iteration that corresponds to the preconditioning step for the outer iteration is allowed to be unreliable. Rank deficiency of the subsequent upper-Hessenberg matrix could signal a potentially faulty execution of the inner iteration that would require some recovery strategy. The decision about whether a fault has occurred, and the subsequent recovery, is a global operation and involves agreement and hence global communication. Sloan et al. [36] suggest that error detection and recovery should instead be localized near the fault occurrence. The most expensive computational kernel in linear solver algorithms such as GMRES, the quasiminimal residual method (QMR), and conjugate gradient (CG) is usually a matrix-vector multiplication. Sloan et al. [36] contend that a soft error is most probable in this kernel and suggest an identity check that involves projecting the result of the matrix-vector multiplication onto a test vector. The

projection can be computed two different ways, so the results should agree if there were no faults in the original matrix-vector multiplication. By choosing the test vector to initially have all elements set to unity, they suggest a recursive hierarchical algorithm to hone in on the exact locations of faulty execution. Stoyanov and Webster [40] consider Jacobi and Gauss–Seidel fixed point iteration algorithms and leverage the identity that the norm of the difference between successive iterates should reduce at the same rate as the convergence of the algorithm. They suggest that checking this identity can be used as a method to identify errors due to soft faults and propose rejecting iterations that fail this test as a means to incorporate resilience.

Alternative approaches have also been proposed for explicit PDE schemes that are not iterative in nature. Mayo et al. [30] suggest combining two extremes in the tradeoff space for resilient explicit PDE algorithms: *artificial viscosity*, the physical mechanism that damps perturbations, and *triple modular redundancy*, the strategy of performing computations three times and accepting a result that was reproducible at least twice. They propose using multiple finite difference schemes over stencils of different widths at each grid point of the same formal order of accuracy to identify and discard outliers that might have been corrupted due to soft faults. Donzis and Aditya [11] propose asynchronous explicit finite difference schemes for PDEs that could be viewed as a potential resilience strategy. Typically, the explicit scheme for spatial derivatives requires the solution from neighboring grid points, which involves communication of ghost regions across processing elements (PEs). In the conventional implementation of such schemes, the communication and the calculation of spatial derivatives are completed by all PEs before the next time step is begun; i.e., all portions of the domain advance the solution in a time-step-synchronized fashion. However, one might envision that soft faults cause some portions of the domain to take longer to execute an iteration, introducing an asynchrony between PEs. Donzis and Aditya [11] propose asynchronous schemes whereby neighboring PEs could be at different time steps but still perform spatial derivatives to an intended order of accuracy. They model the asynchrony between neighboring PEs as a random process and show that while such schemes can be stable the accuracy in both time and space might be degraded. However, the desired order of accuracy severely limits the maximum asynchrony allowable between any pair of PEs.

2.4. ***S3D reacting flow solver and ignition benchmark problem.*** S3D is a solver for compressible reacting flows developed by Chen et al. [6]. S3D uses eighth-order finite-difference approximations of spatial derivatives with a method-of-lines discretization integrated temporally using a six-stage, fourth-order compact Runge–Kutta integrator from the family developed by Kennedy et al. [26]. Second

derivatives are obtained by repeated application of the discrete first-derivative operator. The code has been used to produce direct numerical simulations (e.g., sufficiently resolved to capture all relevant continuum scales for turbulence, chemical reaction, and turbulence-chemistry interaction) of a variety of turbulent combustion problems. Past problems include premixed flames [21; 33; 19], nonpremixed flames [22; 42; 18; 17], and autoignition problems [13; 34]. The code solves the compressible Navier–Stokes equations along with transport of the mass fractions of K chemically reacting species using a mixture-averaged transport model. The species density, momentum, and energy equations of hydrodynamics are given by

$$\frac{\partial}{\partial t}(\rho_k) + \nabla \cdot (\rho_k \boldsymbol{v}) + \nabla \cdot \boldsymbol{\mathcal{F}}_k = \dot{S}_k, \tag{10}$$

$$\frac{\partial}{\partial t}(\rho \boldsymbol{v}) + \nabla \cdot [\rho \boldsymbol{v}\boldsymbol{v}^T + p\boldsymbol{I}] + \nabla \cdot \tau = 0, \tag{11}$$

$$\frac{\partial}{\partial t}(\rho E) + \nabla \cdot [(\rho E + p)\boldsymbol{v}] + \nabla \cdot [\boldsymbol{\mathcal{Q}} + \tau \cdot \boldsymbol{v}] = 0, \tag{12}$$

where ρ_k, $\boldsymbol{v}$, p, E, and $\dot{S}_k$ denote, respectively, the mass density for species k, fluid velocity, pressure, total specific energy, and chemical source term for species k for a mixture with K species ($k = 1, \ldots, K$). We note that $\sum_k \boldsymbol{\mathcal{F}}_k = 0$ and $\sum_k \dot{S}_k = 0$ so that summing the species equations gives conservation of mass with $\sum_k \rho_k = \rho$, the total fluid density. Note that $\boldsymbol{v}\boldsymbol{v}^T$ is a (tensor) outer product with T indicating transpose and $\boldsymbol{I}$ is the identity tensor (i.e., $\nabla \cdot p\boldsymbol{I} = \nabla p$). Transport properties are given in terms of the species diffusion flux $\boldsymbol{\mathcal{F}}$, viscous stress tensor τ, and heat flux $\boldsymbol{\mathcal{Q}}$. The viscous stress tensor is

$$\tau = -\eta(\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T) + \tfrac{2}{3}\eta(\nabla \cdot \boldsymbol{v})\boldsymbol{I}, \tag{13}$$

where η is the shear viscosity. The heat flux is

$$\boldsymbol{\mathcal{Q}} = \sum_k h_k \boldsymbol{\mathcal{F}}_k - \lambda \nabla T, \tag{14}$$

where h_k is the enthalpy of the k-th species and λ is the thermal conductivity.

The diffusion velocity of the k-th species is modeled with a mixture-average formulation for $k-1$ species:

$$\boldsymbol{\mathcal{F}}_k = -\bar{D}_k \left[\nabla Y_k + Y_k \overline{W} \nabla W_k + (1 - M_k \overline{W}) \frac{1}{p} \nabla p \right], \tag{15}$$

where $Y_k = \rho_k/\rho$ is the mass fraction of species k, $\bar{D}_k$ is the mixture-averaged diffusion of species k, W_k is molecular weight of species k, and $\overline{W}$ is the mean molecular weight. The final species diffusion velocity is computed so as to enforce

conservation of mass:

$$\mathcal{F}_K = \sum_{k=1}^{K-1} -\mathcal{F}_k, \tag{16}$$

where K is the dominant species, typically N_2. Thermodynamic properties are temperature-dependent; the temperature is related to the energy by

$$E = e_s + \tfrac{1}{2} u_k u_k, \quad e_s = \int_{T_0}^{T} C_v \, dT - \frac{RT_0}{\overline{W}}, \tag{17}$$

where C_v is the mixture constant volume specific heat and R is the ideal gas constant.

The chemical source terms appearing in the species equations are computed by evaluating a chemical reaction network

$$\dot{S}_k = W_k \sum_{j=1}^{N_r} \nu_{kj} R_j, \tag{18}$$

where ν_{kj} are the stoichiometric coefficients for reaction j and the rates of the N_r reactions are given by expressions of the Arrhenius form used by [25]. For example, for a reaction where reactants A and B are converted into products C and D,

$$A + B \Leftrightarrow C + D, \tag{19}$$

the forward rate is given by

$$R_f = [A][B] k_f, \quad k_f = A_{fj} T^{\beta_j} \exp\left(\frac{-T_{aj}}{T}\right), \tag{20}$$

where A_{fj}, β_j, and T_{aj} are coefficients describing the j-th reaction with the reverse rate given by

$$R_b = [C][D] k_b, \quad k_b = \frac{k_f}{k_{\text{eq}}}, \; k_{\text{eq}} = \left(\frac{p}{RT}\right)^{\sum_{n=1}^{N} \nu_{nj}} \exp\left(\frac{\Delta S_j^0}{R} - \frac{\Delta H_j^0}{RT}\right), \tag{21}$$

where ΔS_j^0 and ΔH_j^0 are the entropy and enthalpy of formation differences across the reaction, respectively.

The ideal gas equation of state ($p = \rho RT/\overline{W}$) completes the description of the system. To solve (10)–(12), a method-of-lines approach is used where spatial derivatives are replaced by a finite difference operator of the form

$$\left[\frac{\partial \phi}{\partial x}\right]_i \approx \sum_{m=1}^{4} (\alpha_m \phi_{i-m} + \alpha_m \phi_{i+m}). \tag{22}$$

In the course of evaluating the time derivatives, S3D computes the various terms much as written here where various kernels (e.g., compute operand, apply derivative

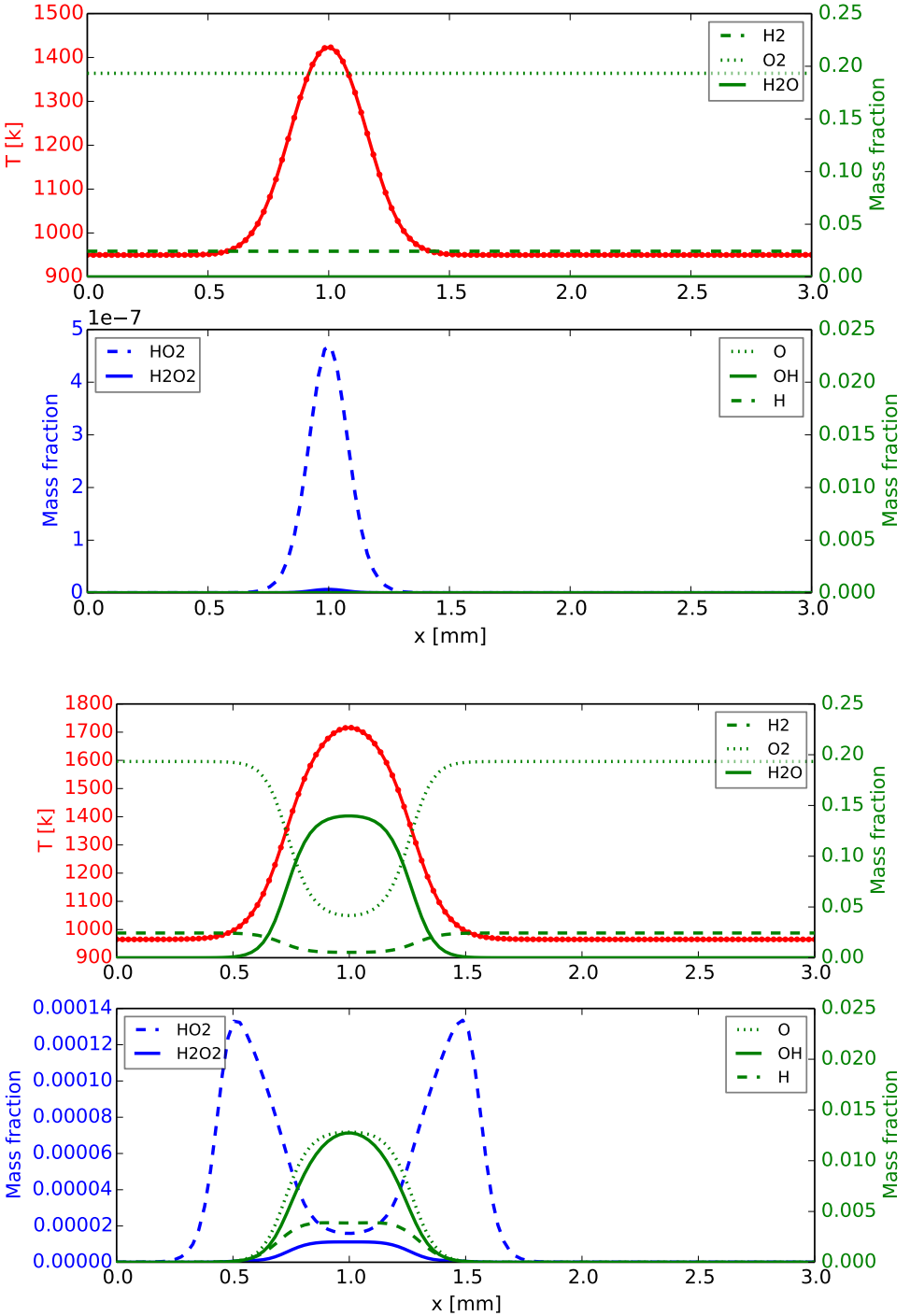

Figure 1. Spatial profiles of temperature and species mass fractions at $t = 5.5\,\mu$s (top) and $t = 30\,\mu$s (bottom) from reference solution obtained with 6,4-RK integrator.

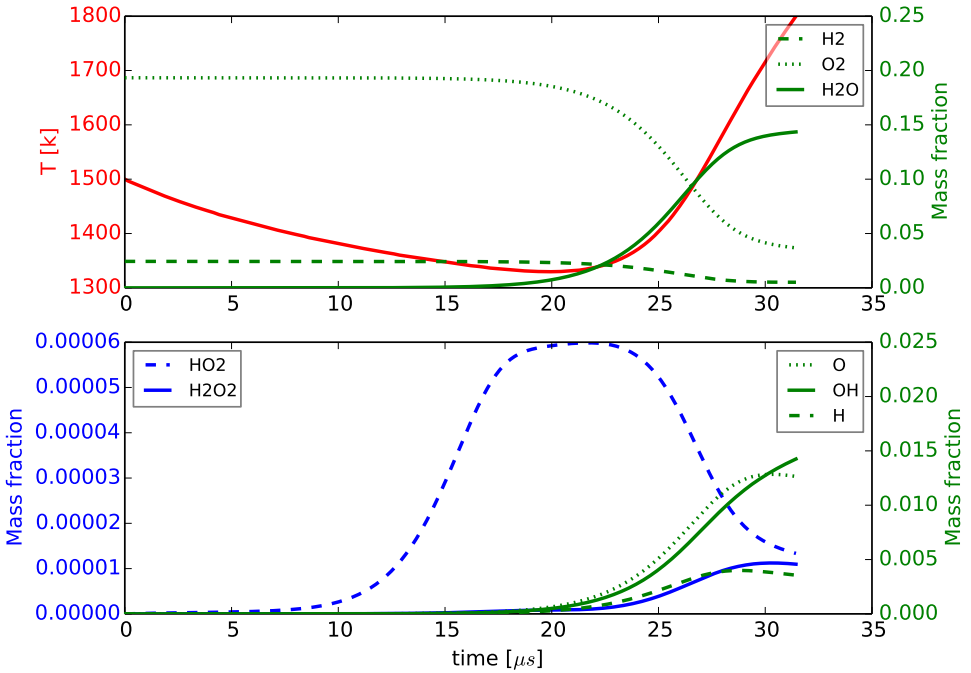

Figure 2. Temporal evolution of maximum temperature and species mass fractions at grid point coinciding with maximum temperature from reference solution obtained with 6,4-RK integrator.

operator, and compute diffusion velocity) operate on the entire solution grid until all of the time derivatives are completely assembled.

In the tests that follow, we will use a fixed time step and tolerate the extra computational cost as a necessary expense to remove one aspect that would make the results more difficult to interpret; in future work we plan to study the combination of SDC and adaptive time step control. The canonical problem is a one-dimensional simulation of a homogeneous mixture composed of hydrogen and air mixed in a stoichiometric ratio with a Gaussian temperature hot spot placed in the center of the domain according to

$$T(x) = T_0 + (T^* - T_0)\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-x^*)^2/(2\sigma^2)}. \tag{23}$$

Solutions for this problem obtained using S3D and the native integrator used historically in S3D (the 6,4-RK algorithm) are shown in Figures 1 and 2. The problem is one-dimensional; 120 grid points are used to spatially resolve the ignition process, and a fixed time step of 5 ns is used in all cases. Figure 1 shows that the initial spatial temperature profile drives formation of a broad pool of hot radicals, led by HO_2 that is eventually consumed as the mixture proceeds towards ignition

and takes the first steps towards the formation of a front. In the time histories shown in Figure 2, the peak temperature decreases due to diffusive processes along with the slow buildup in HO_2 followed by an increase in H_2O_2, OH, and O and finally a rapid rise in temperature. The chemical mechanism is that of Li et al. [29]; *CHEMKIN's* [25] *tranlib* is used to evaluate transport coefficients for a mixture-averaged diffusion formulation. This test case has a relatively long "soaking" period, requiring approximately 5000 time steps before the onset of thermal runaway at 20 μs. This provides ample opportunity for small errors to compound into a large effect on the solution yet is relatively manageable for experimentation. A similar test case, a zero-dimensional ethylene-air ignition problem, is used by Spafford et al. [37] to study the effects of single precision on chemical reaction rate evaluation in the context of porting S3D kernels to a graphics coprocessor, where the test case proved sufficiently sensitive to the accuracy of the function evaluation that evaluating the reaction rates in single precision is insufficient to achieve an acceptable solution.

3. Soft error injection response

In this section we look at injecting two types of soft errors into major work arrays (those of the dimension of the solution grid) during the computation of the solution:

(A) scaling a single value within a work array by a large factor (i.e., multiplying by 10^4) and

(B) reversing the value of a bit at any position within the array (i.e., the value at any grid point could have any bit within it flipped, including the sign bit, the mantissa, and the exponent positions).

We use the type-A errors to explore the sensitivity of the solution to various intermediate values and to study how continued SDC sweeps can correct for such errors. Type-A errors produce a moderate response in that they typically produce a perturbed state that is incorrect but still physically plausible — the circumstance where silent data corruption is intuitively likely. Type-B errors are more realistic but can result in perturbed states that are physically inconsistent (e.g., negative temperatures). We use the bit flip approach, described in Section 2.2, for a comprehensive assessment of the technique integrated into the application code at the end of this section. In all cases, we limit our study to the work arrays that form return values of basic "simulation kernels" (which will be described in the following subsection). In other words, we leave persistent variables (e.g., stencil coefficients), control flow and instruction logic, and the solution vector at the start of the time step unperturbed.

3.1. ***Work array sensitivity.*** The S3D algorithm computes several quantities that are stored in work arrays during the evaluation of the right-hand-side function, and the sensitivity of the solution to perturbations varies widely between quantities.

To demonstrate this, we modified the code that evaluates the temporal derivatives of given quantities so that the results of kernel functions are perturbed. That is, evaluation of the time derivative involves application of several kernels called as functions that manipulate a set of work arrays:

$$\mathcal{R}^k(\vec{W}) \leftarrow \text{kernel}_k(\mathcal{I}^k(\vec{W}), \vec{q}, \dots), \tag{24}$$

where $\vec{W}$ is the vector of multidimensional work arrays, $\mathcal{R}^k(\vec{W})$ is the subset of the work arrays altered by the k-th kernel, $\mathcal{I}^k(\vec{W})$ is the subset of the work arrays used as input to the k-th kernel, $\vec{q}$ is the vector of conservative state variables at the start of the time step, and the $(\dots)$ represents the constants that complete the closure for the kernel. In this nomenclature we apply a perturbation function $\mathcal{P}$ that applies a single bit error (as discussed near the end of Section 2.2) to the return values of each kernel immediately after each kernel completes:

$$\widehat{\mathcal{R}}^k(\vec{W}) \leftarrow \mathcal{P}[\mathcal{R}^k(\vec{W})]. \tag{25}$$

The scratch/work arrays to which the error injection was applied have dimension $8 \cdot$ (nx, ny, nz), (nx, ny, nz, 3, 3), (nx, ny, nz, ns, 3), and (nx, ny, nz, ns) whereas the carryover arrays with dimension (nx, ny, nz, ns, 2) and the various setup arrays of smaller dimension as well as executable code have not been made vulnerable. Comparing perturbed runs to the baseline calculation described in Section 2.4 (again with a fixed 5 ns time step and the 6,4-RK time integration method), we obtain a sensitivity profile for the various work arrays. Figure 3 indicates the difference in the maximum temperature in the simulation domain at a fixed time near the end of the ignition delay when the various quantities are subjected individually to a one-time perturbation. The perturbation is applied to the output work array at the grid point where the temperature is maximum by multiplying the value by 10^4 immediately after the value is calculated during the first substage function evaluation for the time step beginning at $t = 5.5\,\mu$s. We observe, as many others have previously (e.g., [16]), that such error injection can result in different categories of behavior:

(1) The simulation fails in a detectable manner before completion, frequently as soon as the perturbation is injected. This is typically due to an unrealizable condition (e.g., temperature outside physical bounds or the sum of the species mass fractions becoming much larger than unity).

(2) There is no detectable effect on the calculated ignition delay.

(3) The calculation proceeds without apparent error to completion, and the calculated ignition delay is altered, with the size of the error depending on the size of the perturbation earlier in the calculation.

While the first type may slow scientific progress due to frequent restarts, it is the final type — the silent, undetectable errors that alter the result of the calculation —

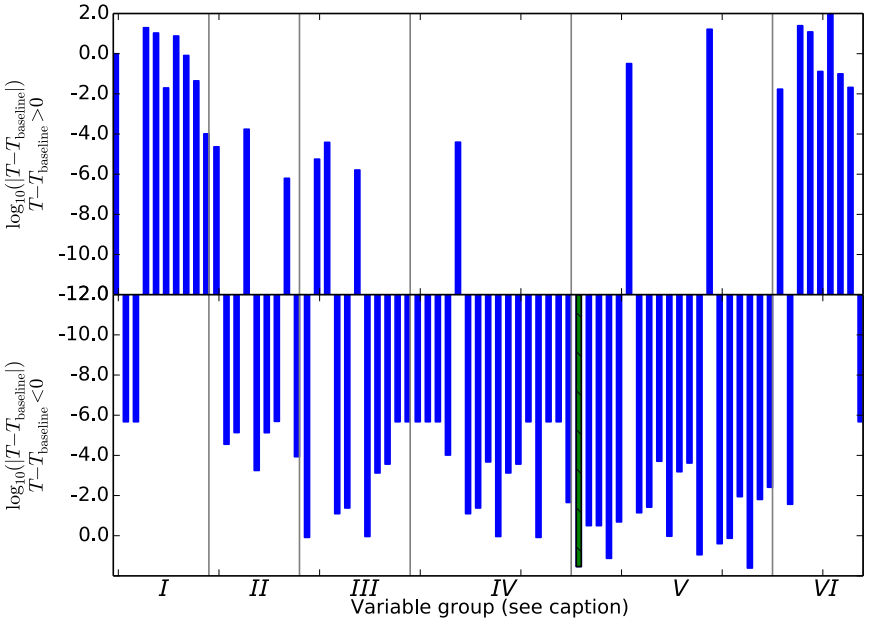

Figure 3. Difference in maximum temperature in domain from baseline at fixed time near end of ignition delay resulting from one-time perturbation of work arrays during calculation using traditional (6,4-RK) integration algorithm. Derivative of density is highlighted in green crosshatch. Work arrays where perturbation resulted in simulation crash are not shown. Variable groups are as follows: Group I, primaries (u, γ, c_p, Y_α); Group II, enthalpies (h_α); Group III, gradients (∇u, ∇T, ∇Y_α); Group IV, diffusive fluxes (τ_{ij}, J_α, J_T); Group V, second derivative operands and results (momentum, energy, species); and Group VI, reactions (S_α).

that are the most serious. Of the 93 kernel return values perturbed individually, for 74 of those variables the calculation proceeds to completion. The remainder result in simulation crashes (e.g., from out-of-bounds temperature extremes) and are not shown in Figure 3. The error in the temperature at the end of the solution ranged from 70 K below the correct temperature to 93 K above the correct temperature; this corresponds to impacting the calculated ignition delay by more than 5%. While it is difficult to make generalizations, perturbations that increased the reaction rate involving known ignition promoters for this mechanism (O, OH, and H) resulted in a significant temperature increase (hence, shorter ignition delay). Conversely, perturbations that increased the transport rates and hence hindered the buildup of radicals led to a decrease in temperature (hence, longer ignition delay). The perturbation that increased the source term for the continuity equation led to a decrease in temperature and is indicated in green in Figure 3 and will be considered in detail in the following subsection as representative of the error injections that led to silent data corruption.

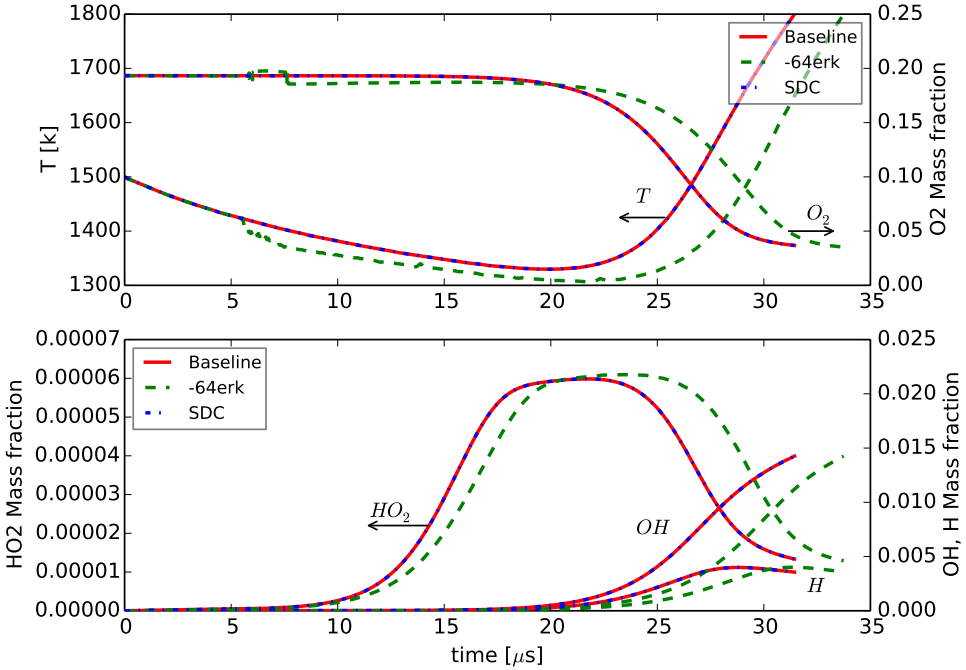

Figure 4. The effect of perturbation of the continuity equation source term on solution temporal evolution using 6,4-RK and SDC integration schemes; temporal plots were extracted at a fixed spatial location where error is injected. Notably, the SDC solution is indistinguishable from the baseline solution while the Runge–Kutta solution is silently and significantly corrupted.

3.2. ***Solution after injection of perturbation.*** Modifying the term that forms the density time derivative in the RHS evaluation, that is,

$$\frac{\partial \rho}{\partial t} = \frac{\partial(-\rho u)}{\partial x}, \tag{26}$$

results in a greater than 5% increase in the eventual predicted ignition delay and a significant change in the temperature at the end of the baseline ignition delay as highlighted in Figure 3 using the 6,4-RK integration method. Figure 4 shows the temporal evolution of the solution for temperature and key species for the baseline, unperturbed case, for the 6,4-RK integration and for SDC integration. The SDC integration is performed using three Gauss–Lobatto quadrature nodes and four correction sweeps. Figure 5 compares the spatial profiles at two different times: the time step after the error is injected and the time step when the baseline case reaches the ignition criterion. The perturbation grows over time after the injection (at $t = 5.5\,\mu$s). In keeping with the silent nature of the corruption, by inspection of the portion of the time history after the fault injection, it is difficult to tell that an error has occurred. Similarly, while it is obvious from looking at the spatial

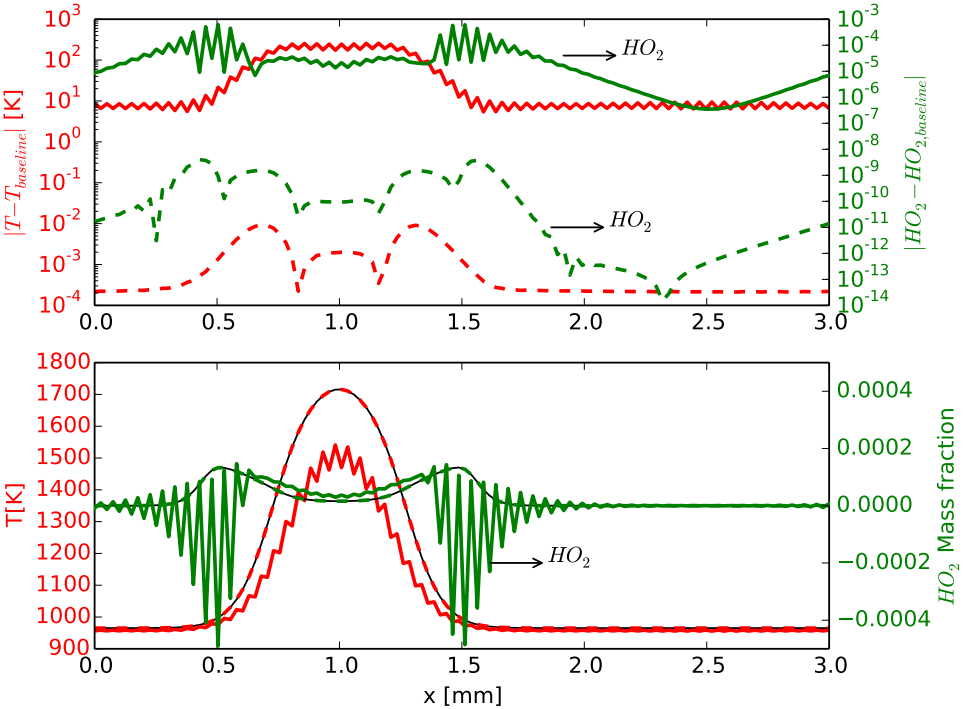

Figure 5. Effect of perturbation of continuity equation source term on solution using 6,4-RK and SDC integration schemes at time of baseline case ignition. Solid lines are the 6,4-RK solution, and dashed lines are the SDC solution for temperature (red) and HO_2 mass fraction (green). The upper plot shows the difference between the computed solution and the baseline while the lower plot shows the computed solution alongside the baseline (in solid black).

profiles at later times in Figure 5 that the solution is contaminated by ringing, it is not clear how to distinguish this from under-resolved physics [5]. Conversely, the solution traces obtained when using SDC with a fixed number of iterations are indistinguishable from the baseline, unperturbed case. This is an empirical demonstration of the tendency of the SDC iterations to recover from soft errors that result in silently corrupted data when using the traditional integration algorithm.

In Figure 6, the residual as given in (4) is shown over time; the curves shown for $|\mathcal{R}_k|$ are the magnitude of the residual for the k-th correction iteration. There is one value per time step plotted obtained at the end of the time step; the lower portion of the figure is an enlargement of the upper portion. We observe that the error injection can be detected by monitoring the residual, which increases sharply when the error is injected. In this experiment the number of SDC correction iterations is held constant. While this is sufficient to damp the error injected to the point where the solution is not qualitatively deteriorated, the residual at the final iteration has

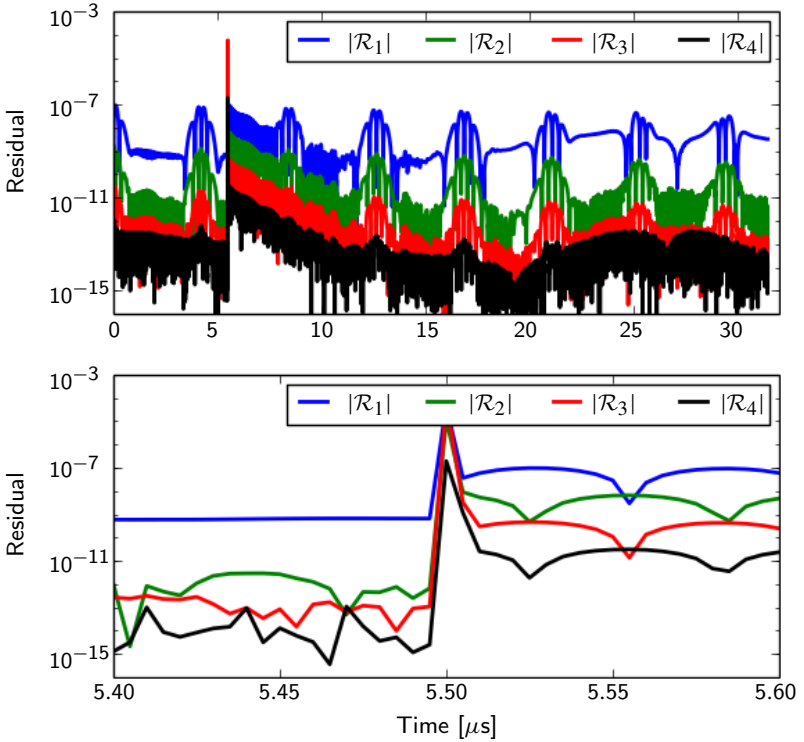

Figure 6. SDC residual response to soft perturbation for four correction sweeps given by $\mathcal{R}_1$–$\mathcal{R}_4$. The lower plot is a zoom in on the region of the fault injection; note that the fault is clearly evident by examining the residual.

not reached its final value prior to the error injection. It is several time steps later that the residual after the final time step reaches approximately the same magnitude as the final residual prior to the error injection. In the next subsection we will look at the response of a linear problem to shed more light on how further SDC iterations reduce the error in a contaminated solution.

3.3. ***Response of linear problem to perturbation.*** In Figures 7 and 8, a similar experiment is performed on the linear test problem

$$y'(t) = y(t), \qquad y(0) = 1 \tag{27}$$

over the interval [0, 1]. Three quadrature nodes are used, and four correction sweeps, including the initial explicit Euler predictor, are performed, giving a formally fourth-order method. The baseline behavior is shown in Figure 7 for comparison to the perturbed solution in Figure 8. A perturbation to the solution is introduced by using $y' = (1.5)y$ for the derivative evaluation during the third SDC sweep at the second quadrature node. For the unperturbed case, the solution error decreases monotonically with iteration count as seen in Figure 7. However, when the error is

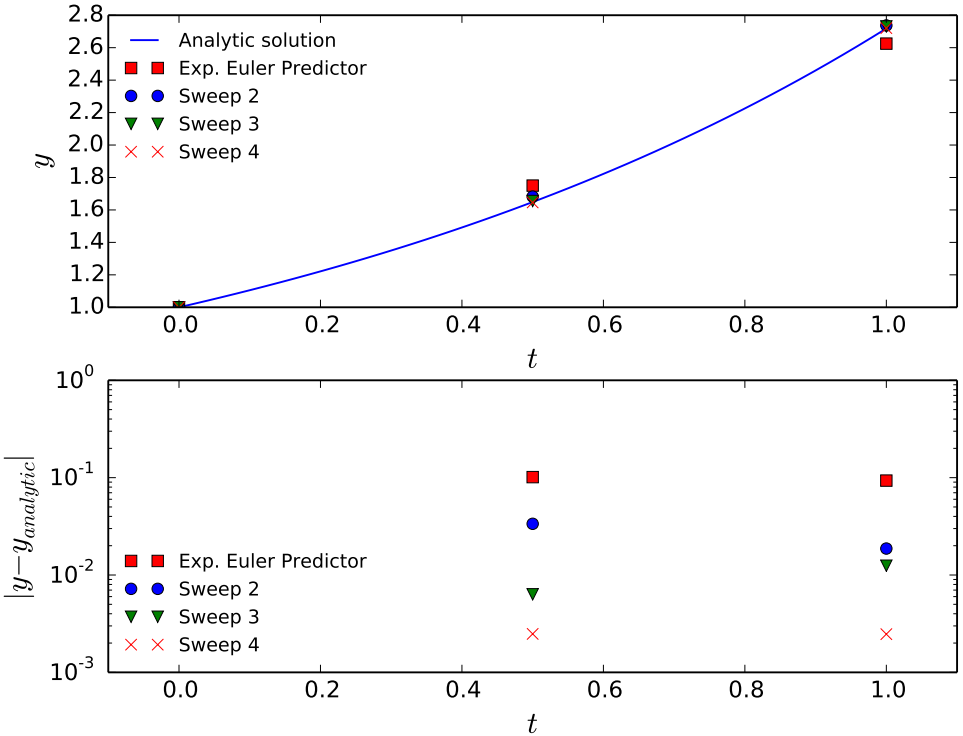

Figure 7. SDC iteration behavior for the linear problem ($y' = Ay$).

injected during the third sweep, we see the error jump up again to near the error in the initial predictor (Sweep 4 in Figure 8). After subsequent sweeps the error is reduced until after Sweep 7 the error in the solution is less than before the error is injected. Figure 9 demonstrates that the error damping is geometric for a wide range of perturbation magnitudes. The horizontal axis in Figure 9 corresponds to the size of the multiplicative perturbation to the derivative computation $y' = sy$. We find that across a wide range of s, both larger and smaller than unity, the error is damped with a consistent ratio. Also of note in Figure 9, we look at continuing the SDC iterations beyond the number of passes necessary for convergence of the reference solution. Even for large perturbations that result in errors several orders of magnitude larger than the reference solution converged error, the converged solution remains the same. This feature of SDC — the ability to recover from such large excursions from the true solution — leads to its natural resilience.

3.4. ***Response to multiple errors.*** In this subsection we conduct an experiment to assess the potential of the SDC iterations to recover from soft errors in a more realistic scenario. We use the baseline test case described above, running the simulation until a fixed time t_{ign}. We then extract the maximum temperature in

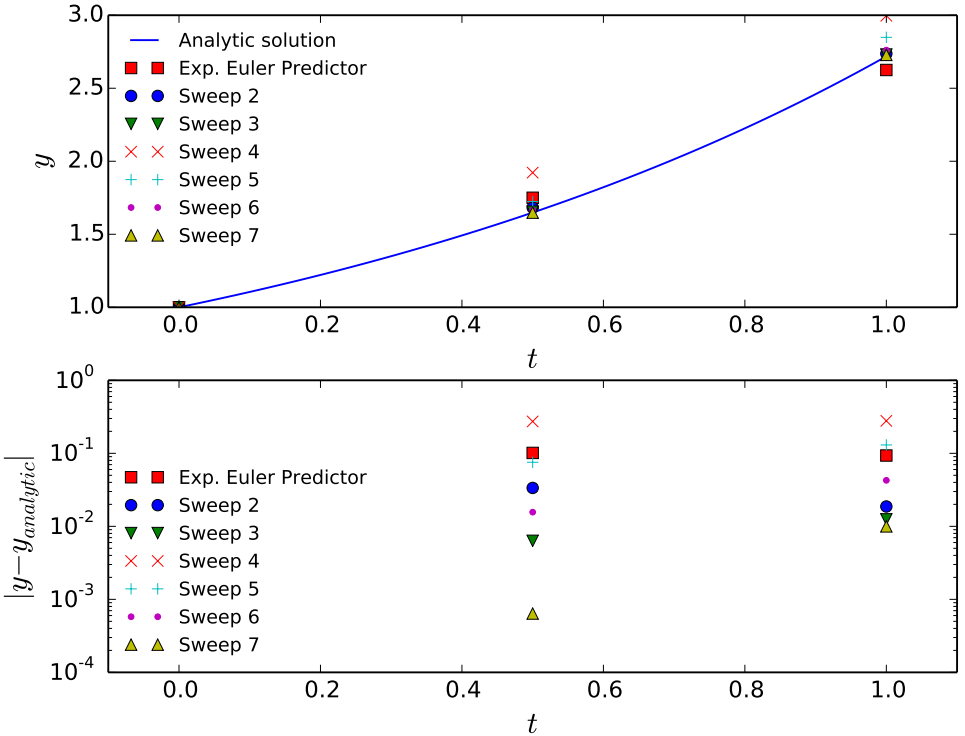

Figure 8. Effect of perturbation of the linear problem on SDC iteration convergence. Error is injected during Sweep 3 which results in an error larger than the initial predictor but is then damped by Sweeps 5 and 6.

the domain as a global measure of the simulation result. We set up our fault injection framework to inject bit flips into random bits in the return values of randomly selected kernels at random times, as discussed in Section 3. Specifically, we injected one fault every 5580 calls to the error injection callback per rank; this corresponds to approximately one fault every 10 time steps using the baseline RK time-advancement algorithm without error injection. Within this window, each process (MPI rank) chooses a random location where the fault will be injected. At the start of the fault injection window, each rank initializes a counter zero and chooses a random number in the range (0, 5580) to be the fault call. The counter is incremented each time the error injection callback is executed, and when it equals the fault call, a random bit within the valid range of the argument pointer is flipped. The counter continues to increment with successive calls, but without error injection, until it reaches the window size when it is reset and a new fault call count is chosen for the next window. For this one-dimensional calculation five MPI ranks were used.

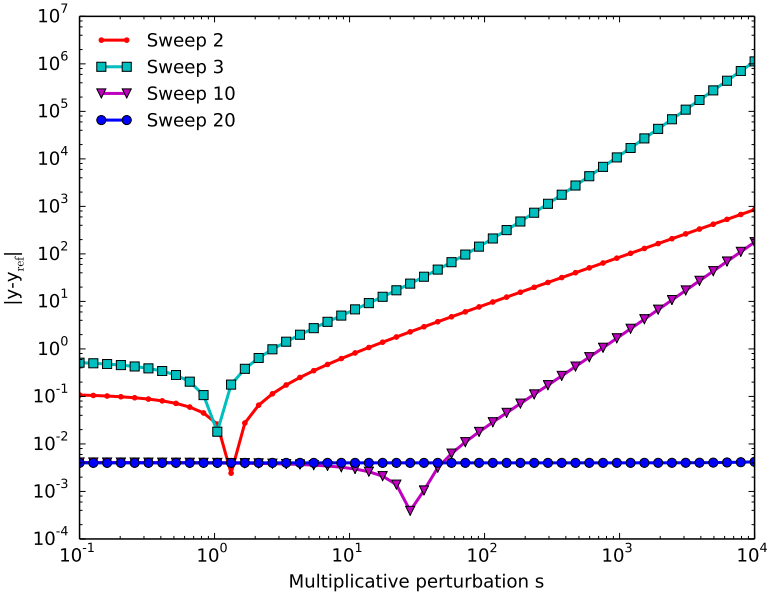

Figure 9. Effect of perturbation magnitude on SDC convergence rate. Reference solution y_{ref} is the analytic solution with $s = 1$.

When faults are injected randomly across the variable array, there is the potential that some faults will result in immediate crashes of the program as identified as the first type in Section 3.1, i.e., flipping the sign bit of major variables or changing the most significant bit in the exponent. These types of faults will cause the program to experience an unrecoverable error that is easily detectable. The test code solves a transport equation for total energy and computes temperature by using a Newton search to solve (17). Hence, a bounds check on the temperature is likely to pick up out-of-bounds issues on any of the variables that participate in (17). The code historically monitors the temperature range during the solution of (17) and terminates if it goes out of bounds. In order to allow the simulation to continue without a full restart, we cache the solution vector at the start of every outer time step and allow the simulation to restart from that point rather than terminating and restarting from a save file.

Furthermore, to deal with the final type of error (those leading to silent corruption), we modified the SDC algorithm to monitor its convergence through the reduction in the residual. We propose a strategy for mitigating soft errors — hardware-introduced faults that are stochastic and transient in nature — based on monitoring the behavior of the SDC correction through the residual to identify when a soft error has occurred and continuing iteration until the residual drops to the prescribed tolerance. In the case of nonrecoverable errors detected during the correction iterations, we restart

	Mean	Minimum	Maximum	Span	Variance
RK	1737.32	1728.92	1758.82	29.90	0.95
SDC	1737.30	1736.70	1743.69	7.00	0.04

Table 1. Variance in temperature at the end of calculation with error injection using baseline Runge–Kutta integration and the SDC approach of the same order. The result from both methods without error injection is 1737.25.

the time step. For each correction iteration (after the first) we compute

$$\mathcal{R}_1 = \frac{\max|\vec{R}_n|}{\max|\vec{R}_1|}, \tag{28}$$

$$\mathcal{R}_{n-1} = \frac{\max|\vec{R}_n|}{\max|\vec{R}_{n-1}|} \tag{29}$$

and continue the correction sweeps until $\mathcal{R}_1 < 10^{-5}$ and $\mathcal{R}_{n-1} > 0.9$, that is, until the residual is small compared to the residual from the first correction pass and is also not changing significantly between successive correction passes. The tolerance values for $\mathcal{R}_1$ and $\mathcal{R}_{n-1}$ were chosen to be consistent with the ratios found in the baseline case without fault injection at the end of the SDC iterations. The maximum number of correction passes is limited to eight, after which the time step is accepted. In practice, only a few time steps encountered this limit.

We conducted 1500 independent runs using both the baseline RK time integration and the proposed SDC method; the distribution of the temperature at the end of the calculation is shown in Figure 10 and Table 1. The data in the table demonstrates that the temperature values using SDC are significantly more clustered near the reference value than those computed using RK. There are some occurrences where the error introduced is sufficiently large that maximum SDC iteration limit is reached before without fully damping the error, which accounts for the nonzero variance in the sample of the SDC solutions. Despite this, the width of the distribution is far narrower than the corresponding baseline distribution. In a production environment, two alternatives to narrow the distribution further are available: more SDC iterations could be allowed, or the time step could be restarted if the iteration limit is reached. For this test, the rate of error injection is significantly magnified from realistic error rates, so either option is likely acceptable with minimal computational cost under realistic error rates for a target platform. This is meant to be illustrative: given the uncertainty in the error rates for future architectures, we demonstrate that the simulation can make progress and the effect of those errors are mitigated, but it is difficult to assess computational cost without knowing what the error rates are. This is left for future work as more realistic predictions and measurements of soft error rates on extreme-scale architectures become available. Satisfyingly, the resilient

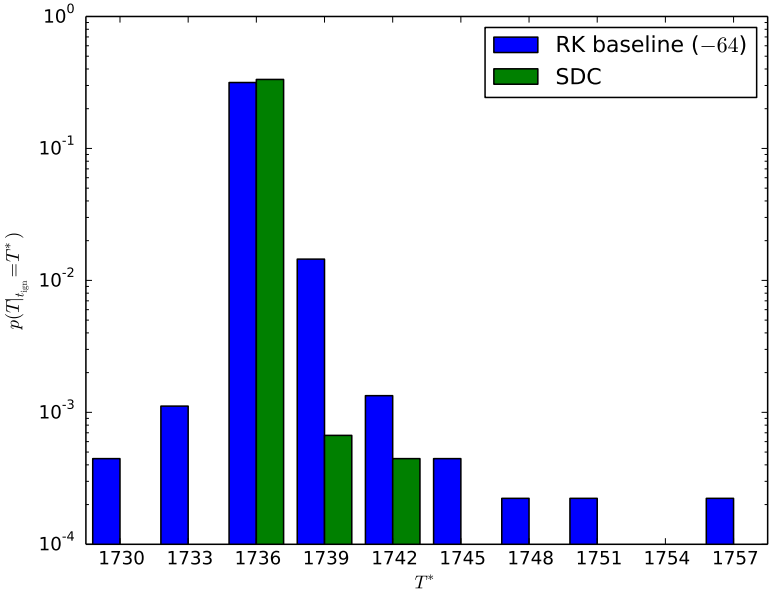

Figure 10. Distribution of temperature at end of calculation with error injection using baseline Runge–Kutta integration and SDC approach of same the order.

form of SDC does not add extra cost beyond a general formulation when there are no hardware faults. In the presence of extreme error rates, the algorithm still makes progress, with the vast majority of runs resulting in no silent data corruption and a clear path to including the remaining outliers available.

4. Conclusion

Natural extensions to a generic SDC algorithm have been proposed that are demonstrated to provide improved algorithmic resilience. It is shown that, in the face of a single transient error, continued SDC iterations beyond those normally required provide a viable approach to error recovery. In the case of elevated rates of stochastic errors, the algorithm can still make progress. In addition, although it is not explored here, the method provides a mechanism for detecting stuck bit errors that could potentially be used to trigger restarting the affected time step using different memory for the work arrays. When no errors are introduced, the suggested formulation reverts to a generic SDC algorithm, so there is no significant cost penalty for the modifications. The formulation is predicated on the ability to protect the integrity of the solution state between successive time steps as well as the program control flow. However, the work arrays used by the application code during a time step that typically comprise the majority of the memory usage can be exposed to significant error rates. This provides an opportunity for savings, where the need

for error correction is potentially reduced without resorting to measures such as redundant calculation that increase computational cost irrespective of the actual error rate realized. As such, the method is a way for application developers to design for potential increased soft error rates on future hardware without the penalty of degraded performance on less error-prone architectures.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research. Work at Lawrence Berkeley National Laboratory was supported by the Co-Design Program of the U.S. DOE Office of Advanced Scientific Computing Research under contract DE-AC02005CH11231.

References

[1] A. R. Benson, S. Schmit, and R. Schreiber, *Silent error detection in numerical time-stepping schemes*, Int. J. High Perform. C. **29** (2015), no. 4, 403–421.

[2] S. Borkar, *Design challenges of technology scaling*, IEEE Micro **19** (1999), no. 4, 23–29.

[3] A. Bourlioux, A. T. Layton, and M. L. Minion, *High-order multi-implicit spectral deferred correction methods for problems of reactive flow*, J. Comput. Phys. **189** (2003), no. 2, 651–675.

[4] P. G. Bridges, M. Hoemmen, K. B. Ferreira, M. A. Heroux, P. Soltero, and R. Brightwell, *Cooperative application/OS DRAM fault recovery*, Euro-Par 2011: parallel processing workshops (Bordeaux, 2011), vol. II, Lecture Notes in Computer Science, no. 7156, Springer, Berlin, 2012, pp. 241–250.

[5] D. L. Brown and M. L. Minion, *Performance of under-resolved two-dimensional incompressible flow simulations*, J. Comput. Phys. **122** (1995), no. 1, 165–183.

[6] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo, *Terascale direct numerical simulations of turbulent combustion using S3D*, Comput. Sci. Disc. **2** (2009), no. 1, 015001.

[7] S. Chen, G. Bronevetsky, M. Casas-Guix, and L. Peng, *Comprehensive algorithmic resilience for numeric applications*, technical note LLNL-CONF-618412, Lawrence Livermore National Laboratory, Livermore, CA, 2013.

[8] A. Christlieb, B. Ong, and J.-M. Qiu, *Comments on high-order integrators embedded within integral deferred correction methods*, Commun. Appl. Math. Comput. Sci. **4** (2009), 27–56.

[9] C. Constantinescu, *Impact of deep submicron technology on dependability of VLSI circuits*, International Conference on Dependable Systems and Networks (Washington, DC, 2002), IEEE, Los Alamitos, CA, 2002, pp. 205–209.

[10] V. Degalahal, R. Ramanarayanan, N. Vijaykrishnan, Y. Xie, and M. J. Irwin, *The effect of threshold voltages on the soft error rate*, 5th International Symposium on Quality Electronic Design (San Jose, 2004), IEEE, Los Alamitos, CA, 2004, pp. 503–508.

[11] D. A. Donzis and K. Aditya, *Asynchronous finite-difference schemes for partial differential equations*, J. Comput. Phys. **274** (2014), 370–392.

[12] A. Dutt, L. Greengard, and V. Rokhlin, *Spectral deferred correction methods for ordinary differential equations*, BIT **40** (2000), no. 2, 241–266.

[13] T. Echekki and J. H. Chen, *Direct numerical simulation of autoignition in non-homogeneous hydrogen-air mixtures*, Combust. Flame **134** (2003), no. 3, 169–191.

[14] J. Elliott, F. Mueller, M. Stoyanov, and C. Webster, *Quantifying the impact of single bit flips on floating point arithmetic*, technical note ORNL/TM-2013/282, Oak Ridge National Laboratory, Oak Ridge, TN, 2013.

[15] M. Emmett and M. L. Minion, *Toward an efficient parallel in time method for partial differential equations*, Commun. Appl. Math. Comput. Sci. **7** (2012), no. 1, 105–132.

[16] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, *Evaluating the error resilience of parallel programs*, 44th annual IEEE/IFIP International Conference on Dependable Systems and Networks (Atlanta, 2014), IEEE, Los Alamitos, CA, 2014, pp. 720–725.

[17] R. W. Grout, A. Gruber, H. Kolla, P.-T. Bremer, J. C. Bennett, A. Gyulassy, and J. H. Chen, *A direct numerical simulation study of turbulence and flame structure in transverse jets analysed in jet-trajectory based coordinates*, J. Fluid Mech. **706** (2012), 351–383.

[18] R. W. Grout, A. Gruber, C. S. Yoo, and J. H. Chen, *Direct numerical simulation of flame stabilization downstream of a transverse fuel jet in cross-flow*, P. Combust. Inst. **33** (2011), no. 1, 1629–1637.

[19] A. Gruber, R. Sankaran, E. R. Hawkes, and J. H. Chen, *Turbulent flame–wall interaction: a direct numerical simulation study*, J. Fluid Mech. **658** (2010), 5–32.

[20] E. Hairer and G. Wanner, *Solving ordinary differential equations, II: stiff and differential-algebraic problems*, 2nd ed., Springer Series in Computational Mathematics, no. 14, Springer, 1996.

[21] E. R. Hawkes and J. H. Chen, *Evaluation of models for flame stretch due to curvature in the thin reaction zones regime*, P. Combust. Inst. **30** (2005), no. 1, 647–655.

[22] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen, *Scalar mixing in direct numerical simulations of temporally evolving plane jet flames with skeletal CO/H_2 kinetics*, P. Combust. Inst. **31** (2007), no. 1, 1633–1640.

[23] M. A. Heroux, *Scalable computing challenges: an overview*, presentation at 2009 SIAM Annual Meeting, Sandia National Laboratories, Livermore, CA, 2009.

[24] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, *Cosmic rays don't strike twice: understanding the nature of DRAM errors and the implications for system design*, Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (London, 2012), ACM, New York, 2012, pp. 111–122.

[25] R. J. Kee, F. M. Rupley, E. Meeks, and J. A. Miller, *CHEMKIN-III: A FORTRAN chemical kinetics package for the analysis of gas-phase chemical and plasma kinetics*, technical note SAND96-8216, Sandia National Laboratories, Livermore, CA, 1996.

[26] C. A. Kennedy, M. H. Carpenter, and R. M. Lewis, *Low-storage, explicit Runge–Kutta schemes for the compressible Navier–Stokes equations*, Appl. Numer. Math. **35** (2000), no. 3, 177–219.

[27] A. T. Layton and M. L. Minion, *Conservative multi-implicit spectral deferred correction methods for reacting gas dynamics*, J. Comput. Phys. **194** (2004), no. 2, 697–715.

[28] ———, *Implications of the choice of quadrature nodes for Picard integral deferred corrections methods for ordinary differential equations*, BIT **45** (2005), no. 2, 341–373.

[29] J. Li, Z. Zhao, A. Kazakov, and F. L. Dryer, *An updated comprehensive kinetic model of hydrogen combustion*, Int. J. Chem. Kinet. **36** (2004), no. 10, 566–575.

[30] J. Mayo, R. Armstrong, and J. Ray, *Efficient, broadly applicable silent-error tolerance for extreme-scale resilience*, technical note SAND2012-8131, Sandia National Laboratories, Livermore, CA, 2012.

[31] M. L. Minion, *Semi-implicit spectral deferred correction methods for ordinary differential equations*, Commun. Math. Sci. **1** (2003), no. 3, 471–500.

[32] A. Nonaka, J. B. Bell, M. S. Day, C. Gilet, A. S. Almgren, and M. L. Minion, *A deferred correction coupling strategy for low Mach number flow with complex chemistry*, Combust. Theor. Model. **16** (2012), no. 6, 1053–1088.

[33] R. Sankaran, E. R. Hawkes, J. H. Chen, T. Lu, and C. K. Law, *Structure of a spatially developing turbulent lean methane–air Bunsen flame*, P. Combust. Inst. **31** (2007), no. 1, 1291–1298.

[34] R. Sankaran, H. G. Im, E. R. Hawkes, and J. H. Chen, *The effects of non-uniform temperature distribution on the ignition of a lean homogeneous hydrogen–air mixture*, P. Combust. Inst. **30** (2005), no. 1, 875–882.

[35] B. Schroeder, E. Pinheiro, and W.-D. Weber, *DRAM errors in the wild: a large-scale field study*, Commun. ACM **54** (2011), no. 2, 100–107.

[36] J. Sloan, R. Kumar, and G. Bronevetsky, *An algorithmic approach to error localization and partial recomputation for low-overhead fault tolerance*, 43rd annual IEEE/IFIP International Conference on Dependable Systems and Networks (Budapest, 2013), IEEE, Los Alamitos, CA, 2013.

[37] K. Spafford, J. Meredith, J. Vetter, J. Chen, R. Grout, and R. Sankaran, *Accelerating S3D: a GPGPU case study*, Euro-Par 2009: parallel processing workshops (Delft, Netherlands, 2009), Lecture Notes in Computer Science, no. 6043, Springer, Berlin, 2010, pp. 122–131.

[38] V. Sridharan and D. Liberty, *A study of DRAM failures in the field*, SC '12: International Conference on High Performance Computing, Networking, Storage and Analysis (Salt Lake City, 2012), IEEE, Los Alamitos, CA, 2012.

[39] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, *Feng Shui of supercomputer memory positional effects in DRAM and SRAM faults*, SC '13: International Conference on High Performance Computing, Networking, Storage and Analysis (Denver, 2013), IEEE, Los Alamitos, CA, 2013.

[40] M. Stoyanov and C. Webster, *Numerical analysis of fixed point algorithms in the presence of hardware faults*, SIAM J. Sci. Comput. **37** (2015), no. 5, C532–C553.

[41] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, *Quantifying the accuracy of high-level fault injection techniques for hardware faults*, 44th annual IEEE/IFIP International Conference on Dependable Systems and Networks (Atlanta, 2014), IEEE, Los Alamitos, CA, 2014, pp. 375–382.

[42] C. S. Yoo, R. Sankaran, and J. H. Chen, *Three-dimensional direct numerical simulation of a turbulent lifted hydrogen jet flame in heated coflow: flame stabilization and structure*, J. Fluid Mech. **640** (2009), 453–481.

Received March 3, 2016. Revised September 9, 2016.

RAY W. GROUT: ray.grout@nrel.gov
Computational Science Center, National Renewable Energy Laboratory, 15013 Denver West Parkway, Golden, CO 80401, United States

HEMANTH KOLLA: hnkolla@sandia.gov
Sandia National Laboratories, P.O. Box 969, M.S. 9158, 7011 East Ave, Livermore, CA 94551-0969, United States

MICHAEL L. MINION: mlminion@lbl.gov
Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory, MS 50A-3141, 1 Cyclotron Road, Berkeley, CA 94720, United States

JOHN B. BELL: jbbell@lbl.gov
Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory, MS 50A-3141, 1 Cyclotron Road, Berkeley, CA 94720, United States