

**Enabling New Flexibility in the SUNDIALS Suite of Nonlinear and Diferential/Algebraic Equation Solvers**

DAVID J. GARDNER, Lawrence Livermore National Laboratory, USA DANIEL R. REYNOLDS, Southern Methodist University, USA   
CAROL S. WOODWARD, Lawrence Livermore National Laboratory, USA CODY J. BALOS, Lawrence Livermore National Laboratory, USA

In recent years, the SUite of Nonlinear and DIferential/ALgebraic equation Solvers (SUNDIALS) has been redesigned to better enable the use of application-speciic and third-party algebraic solvers and data structures. Throughout this work, we have adhered to speciic guiding principles that minimized the impact to current users while providing maximum lexibility for later evolution of solvers and data structures. The redesign was done through the addition of new linear and nonlinear solvers classes, enhancements to the vector class, and the creation of modern Fortran interfaces. The vast majority of this work has been performed łbehind-the-scenes,ž with minimal changes to the user interface and no reduction in solver capabilities or performance. These changes allow SUNDIALS users to more easily utilize external solver libraries and create highly customized solvers, enabling greater lexibility on extreme-scale, heterogeneous computational architectures.

CCS Concepts: · **Mathematics of computing** → **Solvers**; **Ordinary diferential equations**; **Diferential algebraic equations**; **Nonlinear equations**.

Additional Key Words and Phrases: Numerical software, object-oriented design, time integration, nonlinear solvers, high-performance computing

1 INTRODUCTION

The SUite of Nonlinear and DIferential/ALgebraic equation Solvers (SUNDIALS) [41] is a collection of software packages designed to solve time-dependent and nonlinear equations on large-scale, high performance computing (HPC) systems. The packages have evolved from a multi-decade history of highly eicient solution packages for ordinary diferential equations (ODEs) [13, 38, 39], diferential-algebraic equations (DAEs) [61], nonlinear systems [16, 18], and sensitivity analysis enabled integrators [53, 69]. The SUNDIALS codes take advantage of the many innovations this history and the methods community have delivered in terms of adaptive time step and adaptive order integrators for ODEs and DAEs [17, 19, 36, 37, 47, 50], the combination of Krylov and Newton methods for eicient nonlinear solvers [14, 15, 60], methods and software for sensitivity analysis [21, 32, 53, 56], and efective solver controls for time integrators and nonlinear solvers [20, 40, 42ś44]. Over the last two decades the SUNDIALS codes have been used in a number of applications on small to large-scale computing systems [22, 28, 29, 33ś35, 52, 57, 74, 77] as well as in other mathematical libraries and frameworks [3ś5, 55, 62, 68, 76, 77], garnering more than 120,000 downloads in 2021.

During this time, the science simulated as well as the software needed to carry out those simulations have grown in complexity as HPC architectures have transitioned from simplistic homogeneous systems to heterogeneous systems (e.g., CPU+GPU). With the trend toward heterogeneity, the object-oriented design of the SUNDIALS

Authors’ addresses: D. J. Gardner, C. S. Woodward, and C. J. Balos, Center for Applied Scientiic Computing, Lawrence Livermore National Laboratory, 7000 East Ave, L-561, Livermore, CA 94550; emails: {gardner48, woodward6, balos1}@llnl.gov; D. R. Reynolds, Department of Mathematics, Southern Methodist University, P.O. Box 750156, Dallas, TX 75275; email: reynolds@smu.edu.

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or ailiate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.

© 2022 Association for Computing Machinery.

0098-3500/2022/0-ART0 $15.00   
<https://doi.org/10.1145/3539801>

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:2 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

vector structure [41] has enabled incorporating new HPC programming models [49, 63]. However, this design did not extend to other aspects of SUNDIALS. Thus, creating interfaces to utilize new algebraic solver packages targeting large-scale heterogeneous systems would require extensive redundant code with limited extensibility. This paper describes recent enhancements following an object-oriented design to ensure adaptability to new machines and programming environments, allow users to construct architecture-aware and problem-speciic data structures and solvers, ease interoperability with other libraries, and provide greater lexibility to users and for future extensions. The design principles that guided this work have been: assume as little about data layouts as possible; deine interfaces that allow users to provide their own data structures and solvers with little new code; ensure SUNDIALS packages can evolve and change their oferings under these interfaces; produce highly sustainable code; and minimize the impact to current users.

The rest of this paper is organized as follows. The next section gives a brief overview of the SUNDIALS packages and is followed by a discussion of the core SUNDIALS classes. Section 4 describes changes to the vector class and new vector implementations. Sections 5 and 6 discuss new classes for linear and nonlinear solvers, respectively with accompanying demonstration programs. Section 7 overviews the modern, sustainable Fortran 2003 interfaces to SUNDIALS. Lastly, Section 8 gives a summary and next steps for SUNDIALS.

Throughout the text, the presentation of new features and enhancements is generally focused at the level of interfaces and functionality while many of the implementation-speciic details of the contributions are excluded. However, references to detailed descriptions in the SUNDIALS user guides are provided for the interested reader. As the user guide for each SUNDIALS package includes shared text regarding the overall SUNDIALS infrastructure, these citations refer to version 6.2.0 of the CVODE user guide [45]. For details on updating existing codes to the latest SUNDIALS version, see the łRecent Changesž section of the package’s user guide and the examples included with each release. The SUNDIALS source code, example programs, and user guides can be obtained from the SUNDIALS web page [71] or GitHub repository [72].

2 SUNDIALS OVERVIEW

This section provides a brief overview of the numerical methods implemented in the six SUNDIALS packages: CVODE, CVODES, IDA, IDAS, ARKODE, and KINSOL. Additional details on each package, with the exception of ARKODE, are available in [41]. As the native vector and matrix derived classes are written for problems in real N-space, the following overview considers problems posed in RN. Problems posed in other spaces (e.g., CN) can be solved using user-deined derived classes.

2.1 CVODE and CVODES

CVODE targets stif and nonstif initial value problems (IVPs) for ODEs in the explicit form

˙y = f (t,y), y(t0) = y0,

where t is the independent variable (e.g., time), the dependent variable is y ∈ RN, ˙y denotes dy/dt, and f : R × RN→ RN. CVODE uses variable order and variable step size implicit linear multistep methods in the ixed-leading-coeicient form [19, 47] given by

|  |  |  |  |
| --- | --- | --- | --- |
| K1 � | αn,iyn−i + hn | K2 � | βn,i f (tn−i,yn−i) = 0, |

where yn is a computed approximation to y(tn), hn = tn − tn−1 is the time step size, the values K1 and K2 are determined by the method type and order of accuracy, and the coeicients αn,i and βn,i are uniquely determined by the method type and order as well as the recent step size history. Adams-Moulton methods of order 1 to 12 are provided for non-stif problems and Backward Diferentiation Formulas (BDF) of order 1 to 5 for stif cases.

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| Enabling New Flexibility in SUNDIALS | • | 0:3 |

Additionally, CVODE supports projection methods [30, 70] for integrating ODE systems with constraints i.e., where the solution must satisfy д(t,y) = 0.

CVODES [69] is an extension of CVODE for conducting forward or adjoint sensitivity analysis of ODE IVPs with respect to the parameters p ∈ RNpi.e., problems of the form,

|  |  |  |  |
| --- | --- | --- | --- |
| 2.2 | IDA and IDAS | ˙y = f (t,y,p), | y(t0) = y0(p). |

IDA targets the more general case of DAE and implicit ODE IVPs in the form

|  |  |  |
| --- | --- | --- |
| F (t,y, ˙y) = 0, | y(t0) = y0, | ˙y(t0) = ˙y0. |

The state is evolved using variable order and variable step size BDF methods in ixed-leading-coeicient form

|  |  |  |
| --- | --- | --- |
| [12] of order q = 1, . . . , 5 given by, | q | αn,iyn−i = hn ˙yn. |
| � |

As with CVODES and CVODE, IDAS is an extension to IDA with forward and adjoint sensitivity analysis capabilities for DAE and implicit ODE IVPs of the form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2.3 | ARKODE | F (t,y, ˙y,p) = 0, | y(t0) = y0(p), | ˙y(t0) = ˙y0(p). |

The ARKODE package targets ODE IVPs in the linearly-implicit form

M(t) ˙y = f (t,y), y(t0) = y0,

where for any t, M(t) : RN→ RNis a nonsingular linear operator. In many uses cases M(t) is the identity matrix while in inite element computations it is an N × N mass matrix (or a function for its action on a vector). ARKODE supplies two classes of methods where the right-hand side function may be additively partitioned as f (t,y) = f1(t,y) + f2(t,y).

For stif, nonstif, and mixed stif/nonstif problems, where f1 = fEmay contain the nonstif components, which will be integrated using an explicit method, and f2 = fIthe stif components, which will be integrated implicitly, ARKODE provides explicit, implicit, and implicit-explicit (IMEX) additive Runge-Kutta methods [6, 7, 50, 51] with the general form

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| zi = yn−1 + hn | i−1 | AE i,jˆf E (tn−1 + cE jhn,zj) + hn | i� | AI i,jˆf I (tn−1 + cI jhn,zj), | | | i = 1, . . . ,s, |
| � |
| yn = yn−1 + hn | s� | �bE iˆf E (tn−1 + cE jhn,zi) + bI iˆf I (tn−1 + cI jhn,zi) | | | � | , |
| ˜yn = yn−1 + hn | s� | � ˜bE iˆf E (tn−1 + cE jhn,zi) + ˜bI iˆf I (tn−1 + cI jhn,zi) | | | � | . |

Here,ˆfE(t,y) = M(t)−1fE(t,y) andˆfI(t,y) = M(t)−1fI(t,y) convert the IVP from linearly implicit to explicit form, zi are internal method stages, and ˜yn is an embedded solution for temporal error estimation. The number of stages s and the coeicients A∗∈ Rs×s, b∗∈ Rs, c∗∈ Rs, and˜b∗∈ Rsare given by the explicit and implicit Butcher tables that deine the method and its embedding.

For multirate problems, the right-hand side may be split into łslowž components, f1 = fS, advanced with a large step Hn and łfastž components, f2 = fF, advanced with time step, hn ≪ Hn. For such problems, ARKODE

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:4 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

provides multirate ininitesimal step [65ś67] and multirate ininitesimal GARK [64] methods. These methods utilize an s-stage Runge-Kutta method where the stages zi are computed by solving (with a possibly diferent method) the auxiliary ODE IVP

˙v(θ) = ∆ci fF(Ti−1 + ∆ciθ,v(θ)) + r (θ), θ ∈ [0,Hn],

v(0) = zi−1,

where Ti = tn + ciHn, ∆ci = ci − ci−1, and r (θ) is a łforcing functionž constructed from evaluations of fS(Tj,zj), j = 1, . . . ,i, that propagates information from the slow time scale. The fast IVP solution is used for the new internal stage i.e., zi = v(Hn).

2.4 KINSOL

Finally the KINSOL package solves nonlinear systems in root-inding or ixed-point form,

|  |  |  |  |
| --- | --- | --- | --- |
| F (u) = 0 | and | G(u) = u | (1) |

respectively, where u ∈ RN, and the functions F and G map RN→ RN. For the root-inding case, KINSOL provides inexact and modiied Newton methods [25, 59] which may optionally utilize a line search strategy [27]. For general ixed-point systems and systems with the special form F (u) = Lu − N (u) ⇒ G(u) = u − L−1F (u), where L is a nonsingular linear operator (an N × N matrix or a function for its action on a vector) and N is (in general) nonlinear, KINSOL implements ixed-point and Picard iterations un+1 = G(un). The convergence of the iteration may be signiicantly accelerated by optionally applying Anderson’s method [1, 75].

2.5 Integrator Nonlinear Systems

At each time step the implicit integration methods in CVODE(S), ARKODE, and IDA(S) must solve one or more nonlinear systems i.e.,

CVODE: yn − hnβn,0f (tn,yn) − an = 0,

ARKODE: zi − hnAI i,iˆf I (tn−1 + cI jhn,zi) − ai = 0, and

where a∗ is comprised of known data from prior time steps or stages. In adjoint sensitivity computations the IDA: F�tn,yn,h−1 n�q i=0αn,iyn−i�= 0.

same nonlinear systems are solved in the backward problem while forward sensitivity computations require solving an expanded nonlinear system that includes sensitivity equations. In all cases, these nonlinear systems can be viewed as generic root-inding or generic ixed-point problems and are solved using many of the same methods employed by KINSOL.

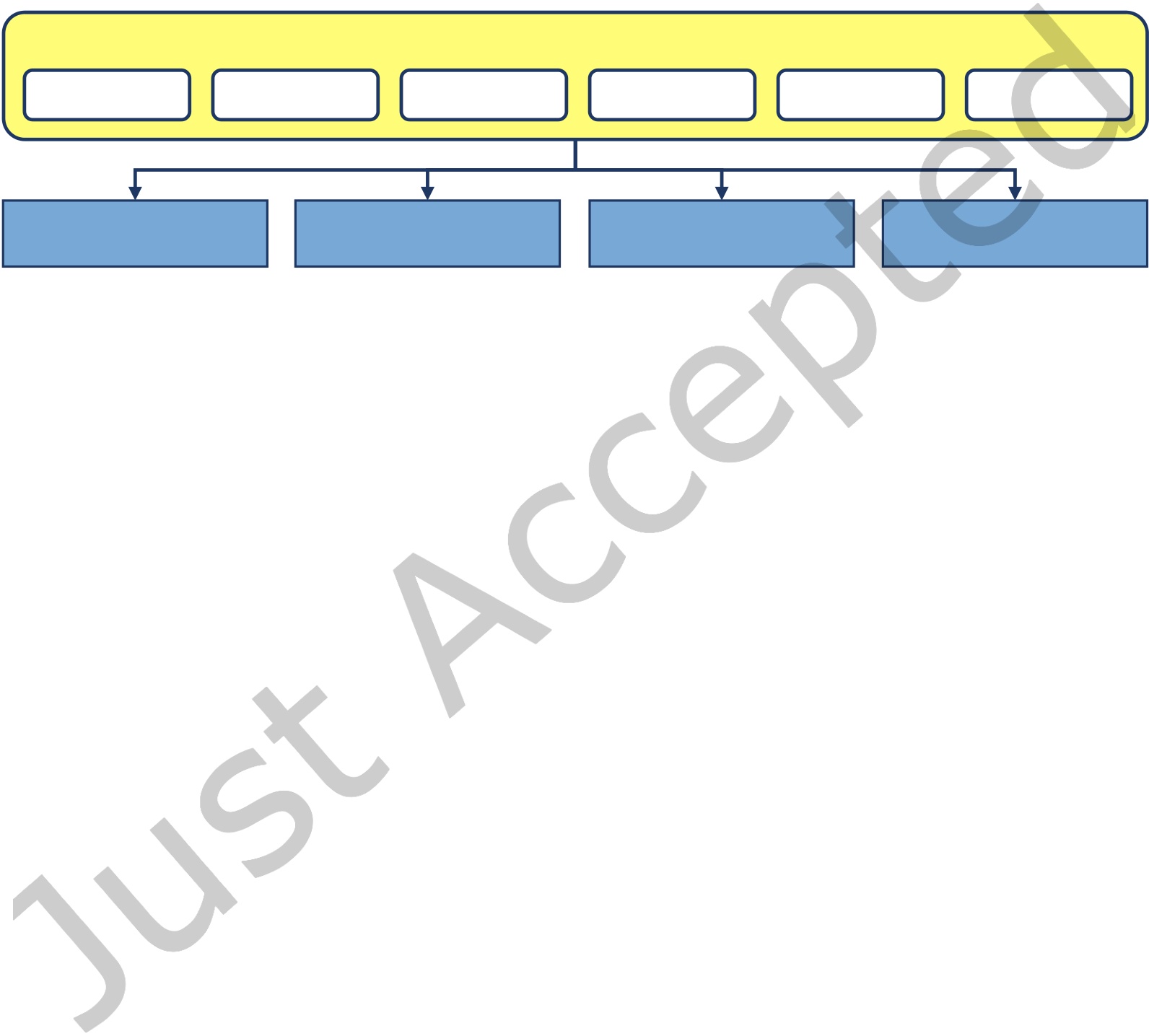
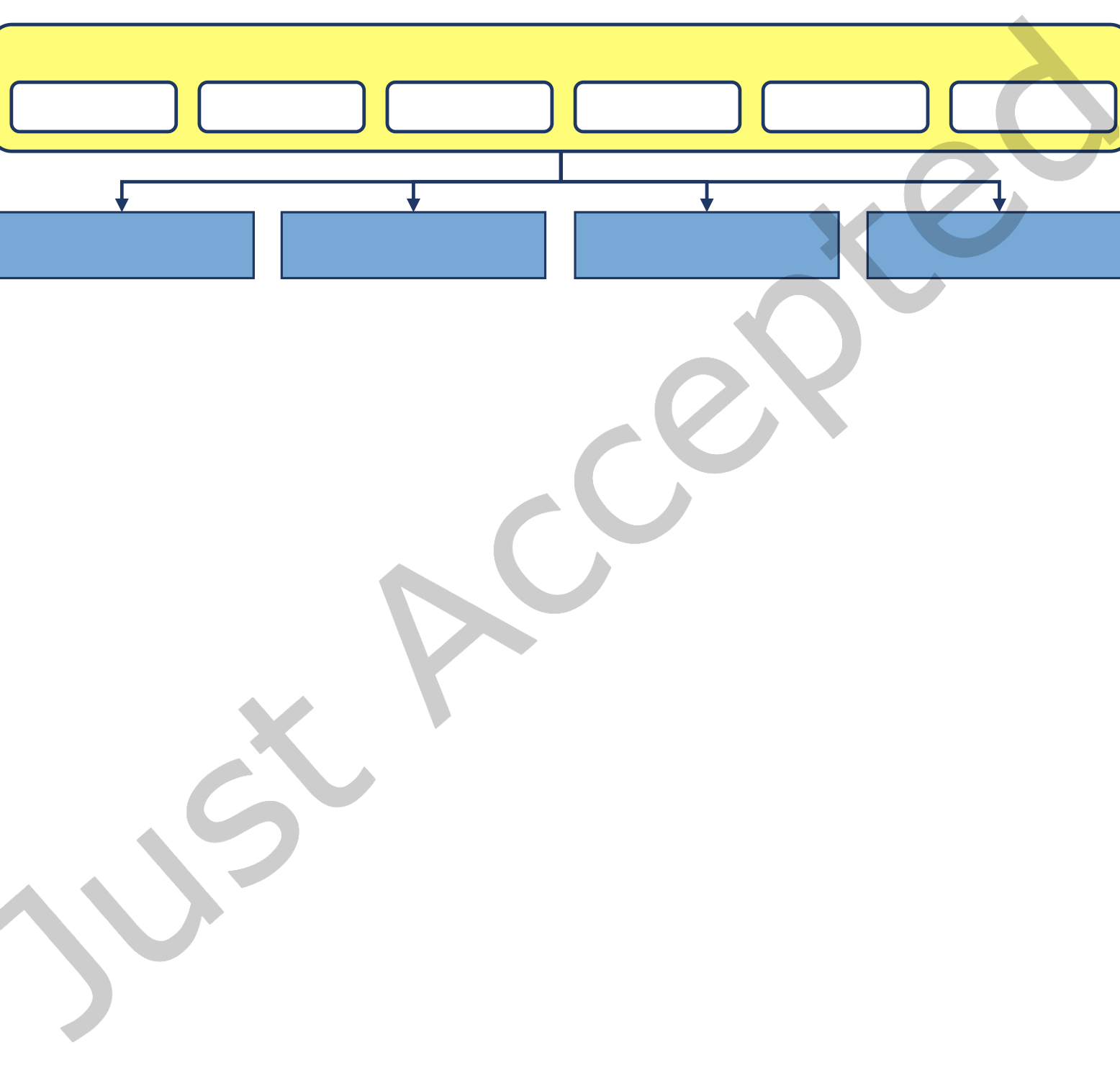
Note that in the forward sensitivity case CVODES and IDAS utilize the strategy discussed in [56] where a block diagonal approximation of the combined system Jacobian is used when solving the larger nonlinear system with Newton’s method. This decoupling enables reusing the ODE/DAE system Jacobian without additional matrix factorizations or preconditioner setups. CVODES and IDAS leverage this structure to utilize the same nonlinear solvers and linear solver interfaces employed by the non-sensitivity-enabled versions of the integrators.

3 SUNDIALS STRUCTURE

To implement the above methods for integrating ODEs and DAEs or solving nonlinear systems, one must be able to perform the following actions:

(1) Compute vector operations e.g., add vectors, compute norms, etc. (2) Solve the nonlinear systems that arise in implicit integration methods.

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| Enabling New Flexibility in SUNDIALS | • | 0:5 |

(3) Solve the linear systems that arise within nonlinear solvers or linearly-implicit ODEs (i.e., mass matrix linear systems).

(4) Compute matrix operations e.g., to explicitly construct the linear systems when necessary or to compute matrix-vector products.

These shared requirements across the packages lead directly to an updated SUNDIALS design, based on the existing N\_Vector class and the new SUNMatrix, SUNLinearSolver, and SUNNonlinearSolver classes shown in Figure 1. These classes deine abstract interfaces for performing vector and matrix operations as well as solving linear and nonlinear systems, respectively. All of the packages and the solver interfaces depend on the vector

**SUNDIALS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CVODE** | **CVODES** | **ARKODE** | **IDA** | **IDAS** | **KINSOL** |
| **N\_Vector** | **SUNMatrix** | **SUNLinearSolver** | **SUNNonlinearSolver** |
| **Interface** | **Interface** | **Interface** | **Interface** |

Fig. 1. An overview of the SUNDIALS packages (CVODE, CVODES, IDA, IDAS, ARKODE, and KINSOL) and the four base classes (N\_Vector, SUNMatrix, SUNLinearSolver, and SUNNonlinearSolver).

interface; however, connections between the other classes depend on the particular derived class and use case e.g., matrix-free linear solvers may accept a NULL matrix object.

This object-oriented design encapsulates implementation-speciic details behind a set of well deined abstract interfaces for performing mathematical operations. This encapsulation enables writing the SUNDIALS packages purely in terms of generic operations on objects, making them agnostic to the user’s particular choice of data structures/layout, method of parallelization, algebraic solvers, etc. Moreover, the design provides lexibility, eases the addition of new features, and maximizes the reuse of shared code. With the new class interfaces discussed in this paper, SUNDIALS packages more seamlessly work with other libraries that provide complementary capabilities for scientiic simulations. For example, SUNDIALS can interoperate with discretization and solver libraries as long as class implementations are provided for interfacing to other library’s data structures and, if necessary, solvers.

3.1 Previous SUNDIALS Interfaces

While the vector class has long been a feature of SUNDIALS, this object-oriented approach was not originally applied to matrices or algebraic solvers. Earlier versions of SUNDIALS did clearly separate the linear solvers and matrices from the integrators and nonlinear solvers. However, these interfaces sufered from three primary challenges. First, each SUNDIALS package deined its own unique interface to each linear solver implementation, leading to signiicant code duplication and an unnecessary maintenance burden. Second, since each package had a separate linear solver API, it was not possible to provide a custom or third-party linear solver that could be immediately utilized by all the SUNDIALS packages. For example, a linear solver written for CVODE could not be used by ARKODE without writing additional interface code. Lastly, providing the interface functions and performing some computations required directly accessing private data within the packages, which hindered the ability of the packages to evolve without breaking user code.

Unlike the linear solvers, earlier versions of SUNDIALS did not separate the nonlinear solvers from the integrators. While the integrator-speciic nonlinear systems could be written in a generic form, the nonlinear

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:6 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

solvers were embedded within the integrators. As such, each package implemented its own nonlinear solvers and did not provide a means for interfacing with external or user-deined nonlinear solvers leading to duplicated code and inconsistent solver options between the packages. Moreover, the lack of a shared nonlinear solver interface across the integration packages limited their lexibility and increased the software maintenance burden.

To address the challenges presented by the old linear solver interfaces and the absence of a nonlinear solver interface, we completely redesigned the handling of matrices, linear solvers, and nonlinear solvers throughout SUNDIALS. Our speciic goals in this redesign were to:

• Collapse the separate package-speciic linear solver interfaces into a single generic interface, thereby removing redundant code.

• Unify the package-speciic nonlinear solver implementations into generic implementations shared across the integrators.

• Maintain the ability to provide integrator-speciic controls over linear and nonlinear solvers for performance optimization.

• Provide streamlined interfaces with clearly deined base classes (and examples) so users can easily leverage application-speciic or external algebraic solvers.

3.2 SUNDIALS Class Implementations

All of the SUNDIALS base classes follow the same fundamental design and are deined as a C structure containing a void pointer to the derived class member data, a virtual method table (VMT) implemented as a C structure of function pointers, and a simulation context object which provides proiling and logging capabilities. For example, the new SUNLinearSolver class is deined in [include/sundials/sundials\_linearsolver.h](https://github.com/LLNL/sundials/blob/v6.2.0/include/sundials/sundials_linearsolver.h) as follows

typedef struct \_generic\_SUNLinearSolver \*SUNLinearSolver;   
typedef struct \_generic\_SUNLinearSolver\_Ops \*SUNLinearSolver\_Ops;

struct \_generic\_SUNLinearSolver {   
 void \*content; /\* pointer to member data \*/   
 SUNLinearSolver\_Ops ops; /\* virtual method table \*/   
 SUNContext sunctx; /\* simulation context   
 \*/ };

Note the base class VMT includes pure virtual functions for which a derived class must provide an implementation, virtual functions the derived class may override, and optional functions that, if non-NULL, the SUNDIALS packages will utilize to improve performance. Thus, the base class pure virtual functions deine the required minimum set of operations for derived classes.

Base class constructors are provided to aid in creating derived classes that are responsible for data allocations and other setup e.g., attaching method implementations to the VMT. In the linear solver case, a function to create a derived class could be implemented approximately as follows

SUNLinearSolver MySUNLinearSolver (...) {   
 SUNLinearSolver LS = SUNLinSolNewEmpty(sunctx); /\* create the base class \*/

|  |  |  |  |
| --- | --- | --- | --- |
| LS->content | = my\_member\_data; | /\* set member data ptr | \*/ |
| LS->ops ->solve | = my\_solve\_function; | /\* set method pointers | \*/ |

/\* ... \*/   
 return LS;   
}

Additionally, base class methods are provided to copy the VMT when cloning objects thus ensuring the method pointers are set correctly. The base constructors and utility methods enable the introduction of new specialized virtual and optional methods aimed at increasing the capability and/or performance of the class with minimal impact on current users. Furthermore, when combined with the new Fortran interfaces discussed in Section 7, these methods enable the creation of custom SUNDIALS class implementations from Fortran.

ACM Trans. Math. Softw.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | ENHANCED VECTOR STRUCTURES | Enabling New Flexibility in SUNDIALS | • | 0:7 |

The SUNDIALS vector class has served an invaluable role in allowing existing codes to rapidly utilize SUNDIALS integrators and solvers, since users needed only to provide an N\_Vector implementation to wrap and operate on application-speciic data structures. Over time SUNDIALS has been progressively updated to provide a wider set of N\_Vector implementations than the original serial and MPI-parallel versions, and now includes vectors based on OpenMP, Pthreads, CUDA, HIP, SYCL, and RAJA for various forms of on-node parallelism. Additionally, implementations that wrap vector objects from PETSc, hypre, and Trilinos have been added to streamline interfacing with other mathematical libraries. However, as hardware has advanced and become more heterogeneous, the required set of vector operations remained static, and there was not a native vector implementation that supported collecting independent vectors into a single object for partitioning data. Thus we have extended the vector class in two distinct ways:

(1) Expanded the set of vector methods to include operations that reduce the number of kernel launches, reduce memory accesses, increase arithmetic intensity, and/or minimize parallel communication.

(2) Added a łmany-vectorž implementation of the vector base class that enables combining diferent indepen-dent vector instances, each using potentially diferent parallelization approaches targeting distinct pieces of hardware, into a single vector.

4.1 New Vector Operations

Throughout a simulation using SUNDIALS, there exist numerous locations that use the same sets of repeated vector operations. Each of these operations over a vector of length N requires O(N ) loating-point operations and memory accesses. Additionally, for accelerator-based implementations, each vector operation requires a separate kernel launch, and for MPI-based parallel vectors each dot product or norm-like operation requires one MPI\_Allreduce call. While the total loating-point work of these sequences of vector operations has remained unchanged, we have now reduced these auxiliary costs (memory accesses, kernel launches, or MPI reductions) by creating a set of łfusedž operations that each combine a speciic sequence of operations into a single routine.

Table 1. Subset of optional fused and vector array operations.

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Description | | |
| int N\_VLinearCombination(int n, realtype\* c, N\_Vector\* X, N\_Vector z) | n−1 | | |
| z = | � | ci Xi |
| int N\_VScaleAddMulti(int n, realtype\* a, N\_Vector x, N\_Vector\* Y, N\_Vector\* Z) | Zi = ai x + Yi  for i = 0, . . . ,n − 1 | | |
| int N\_VDotProdMulti(int n, N\_Vector x, N\_Vector\* Y, realtype\* d) | di = x · Yi  for i = 0, . . . ,n − 1 | | |
| int N\_VLinearSumVectorArray(int n, realtype a, N\_Vector\* X, realtype b, N\_Vector\* Y, N\_Vector\* Z) | Zi = a Xi + b Yi  for i = 0, . . . ,n − 1 | | |

We group these new operations into two categories. The irst set, shown in the irst three rows of Table 1, fuses standard vector operations into a single operation. The N\_VLinearCombination and N\_VScaleAddMulti operations replace multiple calls to N\_VLinearSum, which computes z = ax + by, with a single call. Similarly, N\_VDotProdMulti combines the reductions from several N\_VDotProd calls, which compute x · y, into a single reduction. The second category is composed of operations on collections of vectors that arise in sensitivity

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:8 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

analysis simulations. An example is shown in the last row of Table 1 where the linear sum of multiple vectors are computed in one operation.

To minimize the requirements when supplying a custom N\_Vector implementation, all of these new operations are optional as the base class implementation calls the pre-existing sequence of required operations to achieve the same result. Thus, the operations can be enabled or disabled at runtime by setting the VMT function pointer to the implementation deined function or to NULL, thereby allowing the user to implement only the operations that are most impactful for their use case. More details on the fused operations and the full set of vector array operations that have been added to the class can be found in [45, Sections 6.1.2 and 6.1.3].

In standalone CPU-based vector tests, we have found that fused streaming operations are beneicial in cases that work on many vectors e.g., with high-order integrators and more than approximately 1,000 unknowns per MPI task. Fused reduction operations arise in orthogonalization methods in iterative methods and are beneicial for any number of vectors and with less than approximately 130K unknowns per MPI task. We note these numbers are machine dependent and will certainly change as architectures advance.

4.2 Many-vector

To enable SUNDIALS packages to leverage heterogeneous computational architectures, increase the ability for asynchronous computation, and enable greater lexibility for multiphysics applications, we have created two łmany-vectorž N\_Vector implementations. These vectors are a software layer to enable operating on a collection of distinct vector instances as if they were a single cohesive vector. The many-vector implementations do not directly manipulate vector data; all computations are carried out by the underlying vector instances. As such, the many-vectors are designed to provide users with more lexibility in the data layout or placement on parallel and heterogeneous architectures, and, in turn, over which resources perform calculations on this data. At a high-level, the design is based on the following objectives:

• Provide a clean mechanism for users to partition their simulation data among disparate computational resources (diferent nodes, diferent compute architectures, diferent types of memory, etc.), enabling łhybridž computations on heterogeneous architectures.

• Facilitate the use of a separate MPI communicator for the collection than is used for any of its subvectors, allowing the higher-level communicator to act as an MPI łintercommunicatorž enabling coupled simulations between codes that utilize disjoint MPI communicator groups. From a multiphysics perspective, these disjoint groups can correspond to fully-functioning MPI-based simulations, each using its own standard MPI intracommunicator ś the intercommunicator enables messages to be sent between these disjoint simulations.

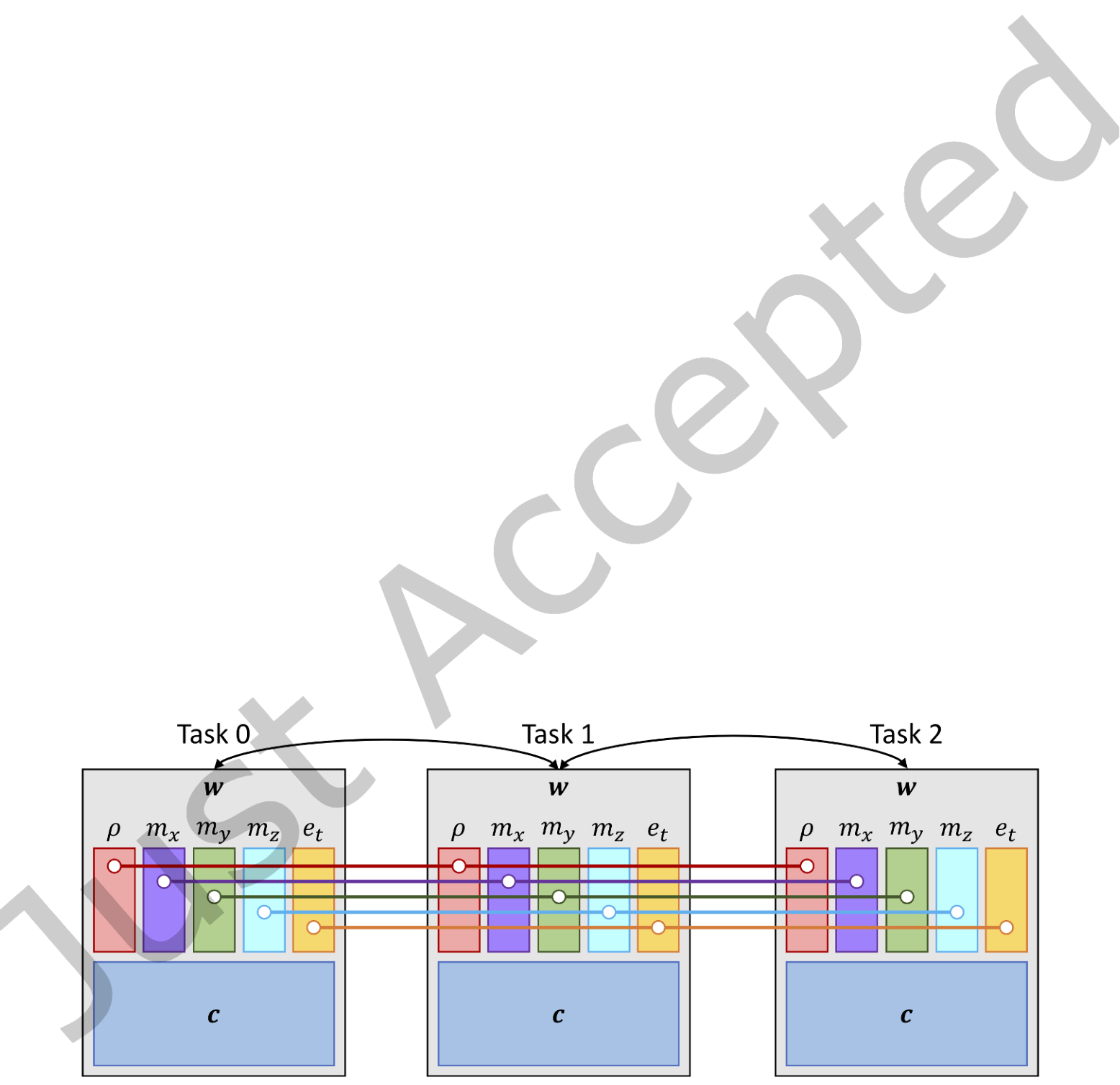
• Allow computationally eicient user-supplied routines to perform operations on the relevant subvectors.• Enable the creation of new SUNDIALS class implementations that leverage the data partitioning (e.g., partitioned integration methods, block (non)linear solvers, etc.) to improve performance for particular application types.

• Ensure seamless functionality with any existing SUNDIALS package or class.

We envision two non-mutually-exclusive scenarios where the many-vector capability will beneit users (though other use cases are possible): partitioning data between diferent computing resources and combining distinct MPI intracommunicators together into a multiphysics simulation.

For example, in the data partitioning case on a hybrid CPU+GPU system, a user would create subvectors using existing N\_Vector constructors, some of which may be CPU-speciic and others GPU-speciic. In this scenario, it is assumed that all MPI-aware subvectors have been constructed using the same MPI intercommunicator. We note that some of these vectors may represent łindependentž portions of the data (e.g., for processes that are not coupled between nodes); these can be constructed out of MPI-unaware N\_Vectors, thereby allowing corresponding

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| Enabling New Flexibility in SUNDIALS | • | 0:9 |

subvector operations to avoid communication altogether. Once all subvectors have been constructed, the user then combines them together into a single many-vector by passing the subvectors to the many-vector constructor. When performing problem-speciic routines (ODE RHS, DAE/nonlinear residual evaluation), the user may then access speciic subvectors to retrieve and ill data, potentially copying data between accelerator and host memory as needed. An illustration of this use case for reactive low is given in Figure 2.

The multiphysics use case corresponds to users who wish to combine separate simulations together (e.g., for luid-structure interaction or atmosphere-ocean coupling in climate simulations) where it is assumed that MPI-aware subvectors have diferent MPI communicators. Here, we expect that the user will (a) create the MPI intercommunicator that connects the distinct intracommunicators together, and (b) handle communication operations between intracommunicators to couple the physical models together in their problem-deining nonlinear residual or ODE right-hand side routines. As in the data partitioning case, the user creates each of the subvectors and the intercommunicator that couples these together. The unifying many-vector is then constructed by providing the subvectors and intercommunicator to the many-vector constructor.

We further note that combinations of these two use cases are also possible via the many-vector interface. Additionally, a many-vector may be constructed using many-vectors as subvectors, thereby allowing a hierarchy of communication patterns within a simulation.

To achieve the above objectives, the MPIManyVector implementation of an N\_Vector consists of a relatively lean data structure comprised primarily of an array of subvectors, the global vector length (the combined length of all subvectors for use in RMS-like norms), and an MPI communicator for global reduction operations. In support of these many-vector modules a new required operation, N\_VGetLength, and a new optional opera-tion, N\_VGetCommunicator, have been added to the set of vector operations. Additionally, to minimize parallel communication with the MPIManyVector, we have expanded the set of optional vector operations to include local reduction kernels so that norm-like calculations may be performed with only a single MPI\_Allreduce call (instead of one for each subvector). These methods compute only the portions of a reduction operation that are local to a given subvector and MPI rank, and their results are combined by the MPIManyVector into the overall result. As these local operations are optional, if they are not supplied by a user-provided subvector, then the MPIManyVector implementation will instead compute the inal result using the original (required) vector operations (with the corresponding increase in MPI reductions).

Fig. 2. Example illustration of the data partitioning many-vector use case for a reactive flow simulation. The full state, w, is composed of subvectors for the fluid variables and chemical species. The fluid quantities (density, ρ, momentum in the x, y, and z directions, m∗, and total energy, et ) are stored as parallel vectors each connected by their own communicator. The chemical species, c, for local reactions are stored in separate task-specific vectors e.g., one or more serial, OpenMP, or GPU vectors per task.

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:10 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

For example consider the weighted root-mean-square (WRMS) norm of a vector x with weight vector w, both of length N, given by

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ∥∥ | �� | 1 | N −1 | ()2. |
| � |
| xwrms = |  | N | � | wixi |
| In the many-vector case, x and w above are replaced with X and W comprised of s subvectors of length Nj with  the total length Ns =�s−1 j=0Nj. The WRMS norm is now given as  �  s−1 Nj−1  ∥X ∥wrms =�Ns 1  ���Wj,iXj,i�2. | | | | |

The MPIManyVector algorithm for computing the WRMS norm is given in Algorithm 1. Here we note the use of 1 versus s + 1 global reductions (each WRMS\_Norm call requires an MPI\_Allreduce). The full set of these optional local reduction operations can be found in [45, Section 6.1.4].

**Algorithm 1:** MPIManyVector weighted root-mean-square (WRMS) norm implementation

**1** local\_sum = 0

**2 foreach** subvector pair Xj, Wj **do**

|  |  |
| --- | --- |
| **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11** | **if** local squared sum method exists **then**   local\_sum += Local\_Squared\_Sum(Xj, Wj)  **else**   contrib = WRMS\_Norm(Xj, Wj)   **if** subvector root task **then**   subvec\_length = Get\_Length(Xj)   local\_sum += contrib \* contrib \* subvec\_length  **end**  **end** |

**12 end**   
**13** MPI\_Allreduce(&local\_sum, &global\_sum, 1, MPI\_DOUBLE, MPI\_SUM, inter\_comm)

**14 return** sqrt(global\_sum / global\_vec\_length)

|  |  |
| --- | --- |
| 4.3 | MPI+X |

An added beneit of the MPIManyVector implementation is the ability for users to easily create hybrid MPI+X vectors where łXž is a type of on-node parallelism. Given an MPI-unaware on-node vector, either one provided with SUNDIALS (e.g., OpenMP, CUDA, RAJA, etc.) or one deined by the user (e.g., OpenCL, OpenACC, etc.), an MPI+X version of the vector can be simply created by supplying an MPI communicator and the on-node N\_Vector to the MPIManyVector constructor. To streamline this particularly beneicial use case we have created an łMPIPlusXž vector, which is a thin wrapper of the MPIManyVector to simplify construction and provide some helpful utility functions for working with the wrapped on-node vector. See [11] for a performance comparison of various GPU on-node vectors beneath the MPIPlusX vector.

ACM Trans. Math. Softw.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | LINEAR SOLVERS | Enabling New Flexibility in SUNDIALS | • | 0:11 |

The new SUNLinearSolver class serves to streamline the use of problem-speciic and third-party linear solvers while maintaining support for integrator-speciic linear solver optimizations within SUNDIALS. To implement a custom SUNLinearSolver, a minimum set of required functions must be implemented. Additionally, there are a variety of optional routines that, if provided, can be leveraged by the SUNDIALS integrators to improve eiciency. There are two required SUNLinearSolver methods: SUNLinSolGetType and SUNLinSolSolve (see Table 2). The irst of these allows the solver to self-identify its łtypež as being either: (a) matrix-based and

Table 2. Required SUNLinearSolver operations

|  |  |
| --- | --- |
| Function | Description |
| SUNLinearSolver\_Type SUNLinSolGetType(SUNLinearSolver LS) | Returns the solver type |
| int SUNLinSolSolve(SUNLinearSolver LS, SUNMatrix A, N\_Vector x, N\_Vector b, realtype tol) | Solves the linear system Ax = b |

direct, (b) matrix-based and iterative, (c) matrix-free and iterative, or (d) matrix-embedded. This identiication allows the SUNDIALS packages to exploit certain solver properties for eiciency. For example, with matrix-based linear solvers the SUNDIALS integrators will infrequently update Jacobian matrices and, for direct linear solvers, matrix factorizations, to amortize the high cost of these operations across multiple time steps or nonlinear solver iterations. Similarly, for iterative linear solvers all of the SUNDIALS packages select tolerances that are łjust tight enoughž to aid an outer Newton iteration without performing unnecessary linear iterations. The purpose of the second required routine, SUNLinSolSolve, is self-explanatory ś this performs the solve for a given linear system Ax = b.

To support integrator-speciic optimizations, the SUNLinearSolver class deines a number of additional łsetž routines that, if implemented, can be called to provide more ine-grained control over the SUNLinearSolver object. In addition, a set of łgetž routines for retrieving artifacts (statistics, residual vectors, etc.) from the linear solver can also be called. While the SUNDIALS documentation provides an exhaustive list of these set and get routines [45, Section 8.1], we discuss a few of these optional operations (see Table 3) here to highlight their beneits in the SUNLinearSolver class.

Table 3. Subset of optional SUNLinearSolver operations

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Description | |  |
| int SUNLinSolSetup(SUNLinearSolver LS, SUNMatrix A) | Performs | infrequent | solver |

setup actions

|  |  |
| --- | --- |
| int SUNLinSolSetATimes(SUNLinearSolver LS, void\* A\_data , ATimesFn ATimes) | Provides the solver with a rou-tine to compute the product |

Ax → z

|  |  |
| --- | --- |
| int SUNLinSolSetScalingVectors(SUNLinearSolver LS,   N\_Vector s1, N\_Vector s2) | Provides the solver with linear system scaling vectors |

The amortization of potentially high cost matrix factorizations or preconditioner setups is achieved by calling the optional SUNLinSolSetup function with an updated system matrix A (for matrix-based linear solvers). The

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:12 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

frequency of these calls depends on the type of integrator and/or nonlinear solver that uses the linear solver, and that frequency may, in many cases, be modiied by the user. However, if this setup routine is not implemented by the SUNLinearSolver, then these costs must instead be incurred at every call to SUNLinSolSolve, which happens much more frequently. To facilitate the linear solver abstraction for matrix-based linear solvers we have created a new SUNMatrix class. This matrix class provides continued support for our pre-existing dense, banded, and sparse-direct linear solvers, and it streamlines interfacing with external linear solver packages.

All matrix-based SUNLinearSolver implementations must have a compatible SUNMatrix implementation that provides routines to clone or destroy existing SUNMatrix objects, although some SUNDIALS packages require additional matrix operations for compatibility (see Table 4). The set of SUNDIALS-provided SUNMatrix implementations, as well as their compatibility with provided N\_Vector and SUNLinearSolver implementations, is discussed in detail in the SUNDIALS documentation [45, Section 7.1].

Table 4. Subset of SUNMatrix operations: (R) are required, (O) are optional

|  |  |
| --- | --- |
| Function | Description |

SUNMatrix SUNMatClone(SUNMatrix A)

(R) Allocates a new matrix with the same size and structure as A

|  |  |
| --- | --- |
| void SUNMatDestroy(SUNMatrix A) | (R) Frees all memory associated with A |

SUNMatrix\_ID SUNMatGetID(SUNMatrix A)

(O) Identiier used by SUNLinearSolvers to check for compatibility

|  |  |
| --- | --- |
| int SUNMatCopy(SUNMatrix A, SUNMatrix B) | (O) Copies all values from A into B |
| int SUNMatScaleAddI(realtype c, SUNMatrix A) | (O) Performs the combination A ← cA + I |

For matrix-free iterative linear solvers, the SUNLinearSolver object requires knowledge of the linear system that must be solved. As such, linear solvers having this type must implement the SUNLinSolSetATimes routine, that is called by SUNDIALS packages to provide a function and corresponding data structure for computing the matrix-vector product, Ax → z.

Lastly, for both matrix-based and matrix-free iterative solvers, the SUNLinearSolver class deines a set of routines to support both scaling and preconditioning, to balance the error between solution components, and to accelerate convergence of the linear solver. To this end, instead of solving a linear system Ax = b directly, iterative SUNLinearSolver implementations may consider the transformed system ˜A ˜x =˜b, where

˜A = S1P−1 1AP−1 2S−1 2, ˜b = S1P−1 1b, ˜x = S2P2x, (2)

and where P1 is the left preconditioner, P2 is the right preconditioner, S1 is a diagonal matrix of scale factors for P−1 1b, and S2 is a diagonal matrix of scale factors for P2x. The SUNDIALS integrators provide these scaling matrices such that S1P−1 1b and S2P2x have dimensionless components. Furthermore, SUNDIALS packages request that iterative linear solvers stop based on the 2-norm of the scaled preconditioned residual meeting a prescribed tolerance:  
 ��� ˜b − ˜A ˜x���2 < tol.

These preconditioning operators and scaling matrices are provided to iterative SUNLinearSolver objects through the optional SetPreconditioner and SetScalingVectors routines. However, if an iterative linear solver imple-mentation does not support scaling, the SUNDIALS packages will attempt to accommodate for the lack of scaling

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| Enabling New Flexibility in SUNDIALS | • | 0:13 |

by instead requesting that the iterative linear solver stop based on the criterion

|  |
| --- |
| ���P−1 1b − P−1 1Ax���2< tol�s1, (3)  where s1 is the RMS norm of the diagonal of S1. This tolerance adjustment is not a perfect replacement for both |

solution and residual scaling, in that it only accounts for the magnitude of the overall linear residual and does not balance error between speciic residual or solution components.

With this SUNLinearSolver class in place, our redesign operated on two levels. On the SUNDIALS package side, we converted each linear solver interface to utilize generic linear solvers and leverage the three aforementioned types. On the solver side, we converted each existing linear solver in SUNDIALS to an implementation of the SUNLinearSolver class. We note that through the deinition and use of these abstract classes alone, we were able to remove nine duplicate linear solver interfaces and three utility iles in each of the SUNDIALS packages, while simultaneously expanding the set of linear solvers available for each package and maintaining integrator-speciic eiciency enhancements. The current set of SUNDIALS-provided SUNLinearSolvers includes direct solvers for dense (native, LAPACK [2], MAGMA [73], and oneMKL [46]), banded (native and LAPACK), and sparse (KLU [23, 24], SuperLU\_MT [26], SuperLU\_Dist [54], and cuSOLVER [58]) matrix-based linear systems, as well as an ensemble of matrix-free scaled-preconditioned iterative linear solvers (GMRES, FGMRES, TFQMR, BiCG-Stab, and CG).

SUNLinearSolver Demonstration Problem

To demonstrate the ability for users to supply a custom linear solver for their applications, we consider the two-dimensional heat equation

∂  
∂t u(t,x,y) = ∇ · (D∇u(t,x,y)) + b(t,x,y).

Here,u is the temperature, D is the diagonal matrix of difusion coeicients (the identity matrix in the experiments below) and the external forcing term b(t,x,y) is given by

b(t,x,y) = − 2π sin2(πx) sin2(πy) sin(πt) cos(πt)

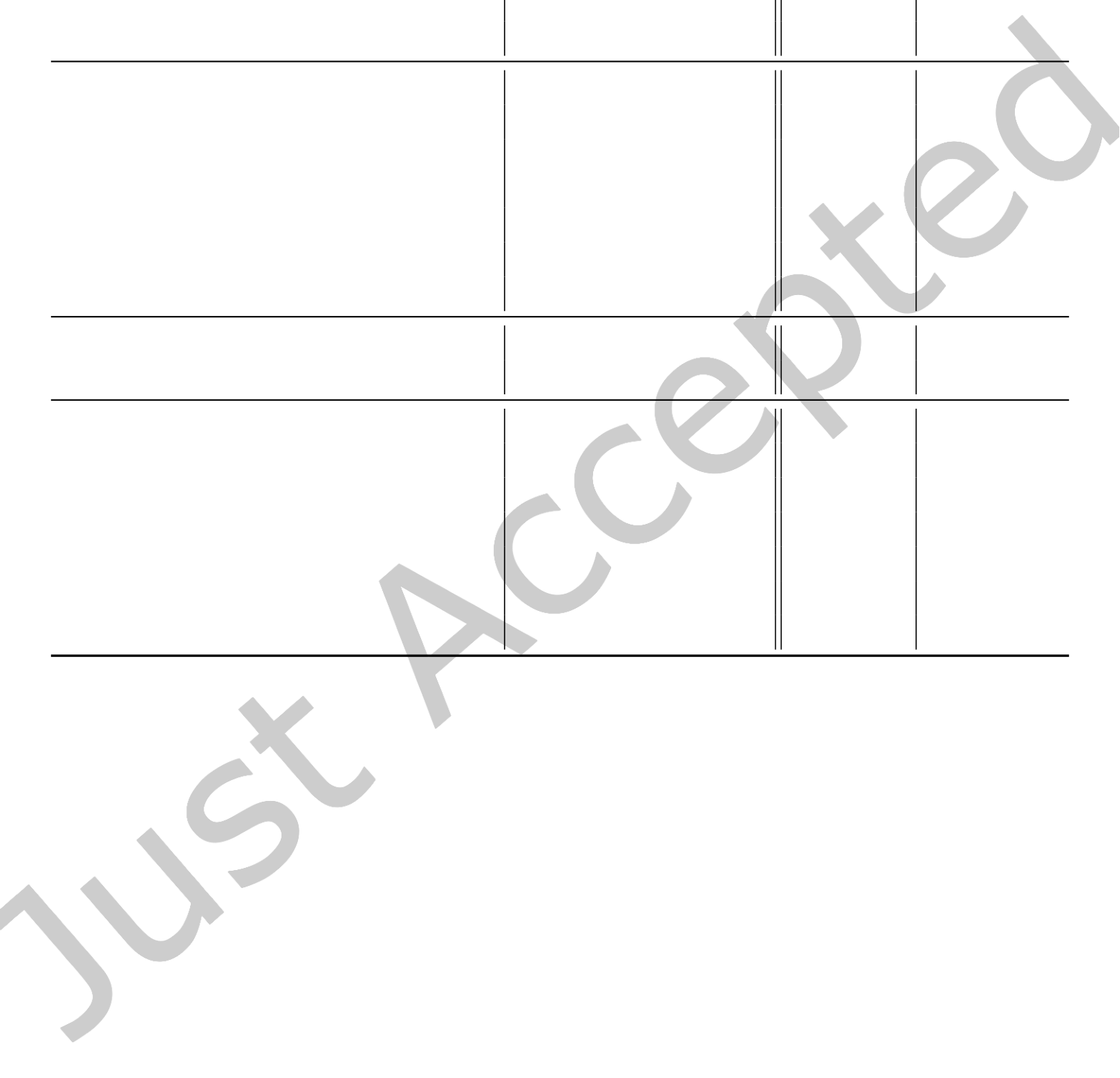
− D1,12π2(cos2(πx) − sin2(πx)) sin2(πy) cos2(πt)  
− D2,22π2(cos2(πy) − sin2(πy)) sin2(πx) cos2(πt).

The problem is solved on the unit square (x,y) ∈ [0, 1]2for t ∈ [0, 1] with u(t,x,y) = 0 on the boundary, and the initial condition is u(0,x,y) = sin2(πx) sin2(πy). With this coniguration, the problem has the analytical solution u(t,x,y) = sin2(πx) sin2(πy) cos2(πt).

Spatial derivatives are discretized using second order centered inite diferences with data distributed over an nx × ny uniform spatial grid decomposed across a two-dimensional Cartesian communicator with px × py MPI tasks. We integrate this problem in time using both CVODE and ARKODE; the ARKODE version of the problem uses the default third order diagonally implicit Runge-Kutta method in ARKODE, while the CVODE version uses adaptive order BDF methods. For both examples, the implicit systems(s) in each time step are solved using SUNDIALS’ default inexact Newton method with custom SUNMatrix and SUNLinearSolver implementations that wrap a structured grid matrix and iterative linear solvers from hypre, CG and GMRES. The linear solvers are preconditioned with hypre’s PFMG parallel semicoarsening multigrid solver [8, 31]. The source iles for the CVODE and ARKODE versions of this example problem are [cv\_heat2D\_hypre\_ls.cpp](https://github.com/LLNL/sundials/blob/v6.2.0/examples/cvode/CXX_parhyp/cv_heat2D_hypre_ls.cpp) and [ark\_heat2D\_hypre\_ ls.cpp](https://github.com/LLNL/sundials/blob/v6.2.0/examples/arkode/CXX_parhyp/ark_heat2D_hypre_ls.cpp) respectively.

The member data for the custom SUNMatrix object consists of the Hypre5ptMatrixContent structure that holds the hypre structured grid, stencil, and matrix objects as well as workspace arrays and other information

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:14 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

Table 5. Integrator statistics from the SUNLinearSolver demonstration problem. The MF and JV columns correspond to SUNDIALS solvers with a matrix-free or a user-defined Jacobian-vector product, respectively. The statistics given are the number of successful time steps (Steps), failed time steps (Step fails), total (integrator + linear solver) ODE right-hand side evaluations (RHS evals), nonlinear iterations (NLS iters), and linear iterations (LS iters). The max error at the final time (Error) and the total average run time across 100 runs (Run time) are also given.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | SUNDIALS v6.2.0 | | | | | |  | SUNDIALS v2.7 | |
| CVODE | | | ARKODE | | | CVODEa | ARKODEb |
| CG | hypre | SUN | | SUN | hypre | | SUN | SUN | SUN | SUN |
| MF | | JV | MF | JV | MF | MF |
| Steps | 84 | 84 | 84 | | 148 | 144 | | 144 | ś | 144 |
| Step fails | 4 | 4 | 4 | | 2 | 1 | | 1 | ś | 1 |
| RHS evals | 109 | 609 | 109 | | 1,053 | 5,253 | | 1,018 | ś | 5,250 |
| NLS iters | 106 | 106 | 106 | | 451 | 435 | | 435 | ś | 435 |
| LS iters | 449 | 500 | 500 | | 4,276 | 4,235 | | 4,236 | ś | 4,231 |
| Error (×10−7) | 3.2 | 3.2 | 3.2 | | 4.0 | 3.5 | | 3.5 | ś | 3.7 |
| Run time (sec) | 8.9 | 13.0 | 9.4 | | 67.6 | 104.4 | | 71.5 | ś | 106.5 |
| GMRES | hypre | SUN | | SUN | hypre | | SUN | SUN | SUN | SUN |
| MF | | JV | MF | JV | MF | MF |
| Steps | 84 | 80 | 80 | | 147 | 144 | | 144 | 80 | 144 |
| Step fails | 4 | 4 | 4 | | 2 | 1 | | 1 | 4 | 1 |
| RHS evals | 109 | 531 | 104 | | 1,042 | 5,193 | | 1,018 | 532 | 5,196 |
| NLS iters | 106 | 101 | 101 | | 446 | 435 | | 435 | 101 | 435 |
| LS iters | 394 | 427 | 427 | | 4,014 | 4,175 | | 4,174 | 427 | 4,177 |
| Error (×10−7) | 3.2 | 11.1 | 11.1 | | 3.8 | 3.6 | | 3.6 | 11.1 | 3.7 |
| Run time (sec) | 8.3 | 12.5 | 9.4 | | 68.0 | 110.4 | | 79.7 | 12.5 | 109.7 |

a An interface to CG from CVODE is not available in SUNDIALS v2.7.0   
b A patch is required to correct the norm used to test for CG convergence in SUNDIALS v2.7.0.

for updating the matrix entries. A similar structure (HypreLSContent) deines the member data for the SUN-LinearSolver. Since the linear solver is an iterative method that uses a matrix object, the custom SUNMatrix implementation only needs to deine the GetID, Clone, Destroy, Copy, and ScaleAddI methods from Table 4. The custom linear solver deines the required GetType and Solve methods as well as the optional Setup method to enable reusing the preconditioner setup. Additionally, the custom linear solver implementation provides the optional NumIters and Free methods for retrieving the number of iterations from a solve and deallocating the SUNLinearSolver and its content.

Since this custom solver does not provide a function to set scaling vectors, the integrators adjust the linear solve tolerance, as shown in (3). The main function for each example creates SUNMatrix and SUNLinearSolver instances using the custom constructors, Hypre5ptMatrix and HypreLS, respectively. Both of these constructors setup the object as outlined in Section 3. Once the matrix and linear solver objects are created, the example programs attach the objects to the integrators with the ARKODE and CVODE SetLinearSolver functions.

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| Enabling New Flexibility in SUNDIALS | • | 0:15 |

The updated linear solver interfaces make it easy to quickly compare multiple solvers and integrators on a problem. Results comparing the CG and GMRES linear solvers from SUNDIALS (matrix-free or with a user-supplied Jacobian-vector product function) and hypre (matrix-based), each preconditioned with PFMG, underneath a linear multistep integrator (CVODE) and a multistage integrator (ARKODE) are given in Table 5. See the iles [ark\_heat2D\_hypre\_pfmg.cpp](https://github.com/LLNL/sundials/blob/v6.2.0/examples/arkode/CXX_parhyp/ark_heat2D_hypre_pfmg.cpp) and [cv\_heat2D\_hypre\_pfmg.cpp](https://github.com/LLNL/sundials/blob/v6.2.0/examples/cvode/CXX_parhyp/cv_heat2D_hypre_pfmg.cpp) for the versions of this example that use the SUNDIALS linear solvers. Additionally, Table 5 includes results from SUNDIALS v2.7.01before the introduction of the new SUNLinearSolver class to illustrate that solver performance is unafected by the new infrastructure (i.e., run times are within the observed run time variability of ∼2% on the machine). As only ARKODE had an interface to the SUNDIALS CG solver before the introduction of the linear solver class, results with CVODE from SUNDIALS v2.7.0 only utilize the SUNDIALS matrix-free GMRES implementation.

The tests use a 4,096 × 4,096 spatial grid distributed across 256 MPI tasks (16 tasks each in the x- and y-directions) and are run on 8 nodes of the Quartz cluster at Lawrence Livermore National Laboratory. Each compute node is composed of two Intel Xeon E5-2695 v4 chips, each with 18 cores. MPI tasks are distributed equally across the compute nodes so that 32 of the 36 cores available on a node are utilized. All integrator and solver options are set to the default values and have not been optimized for this example demonstrating the re-usability of class implementations.

In this example, as seen from Table 5, the combination of CVODE and a matrix-based linear solver from hypre produces the most eicient results for the desired accuracy (the integrator relative and absolute tolerances are 10−5and 10−10respectively). CVODE is well suited to this problem as it only requires one nonlinear solve per step, regardless of the method order, and adapts the order in addition to the time step size to maximize eiciency. ARKODE also adapts the step size, but uses a ixed order method which requires multiple implicit solves per step (for a problem better suited to the multistage IMEX methods in ARKODE see the 1D advection-reaction problem in Section 6). With the default parameters, the matrix-based hypre solvers lead to faster run times than the SUNDIALS solvers. The increased cost in the matrix-free case is a result of the additional right-hand side evaluations (and the corresponding MPI communication) required to approximate the Jacobian-vector product. Supplying a user-deined Jacobian-vector product function eliminates this cost. However, the SUNDIALS solvers still require approximately 0.5 more linear iterations per nonlinear iteration. The lower average number of iterations with hypre solvers is, in part, due to the tolerance adjustment (3) for linear solvers that do not support scaling. The performance of the SUNDIALS linear solvers can be improved for this problem by relaxing the linear solve tolerance.

Prior to the introduction of the new matrix and linear solver classes, this comparison across diferent integrators, solver options, and parameters would have required the user to implement unique interfaces to hypre solvers for both CVODE and ARKODE. With the new SUNDIALS infrastructure such interfaces can easily be reused between the integrators. Moreover, the user’s calling program to setup and evolve the problem in each of these cases is essentially identical. To switch from a native SUNDIALS linear solver to using a hypre solver, the user only needs to swap the SUNDIALS constructor call with a call to their matrix and linear solver constructors and update the SetLinearSolver call to also attach a matrix object. To enable the user-deined Jacobian-vector product, only one additional function call is necessary to attach the user’s function. All other linear solver related functions (e.g., adjusting the preconditioner/linear system update frequency or linear solve tolerance) are independent of the choice of SUNLinearSolver object.

1The implementation of CG in SUNDIALS v2.7.0 incorrectly used a WRMS norm rather than a weighted 2-norm for checking convergence. As such, the results presented are with a patched version of v2.7.0 updated to use the correct norm.

ACM Trans. Math. Softw.



0:16 • D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos

6 NONLINEAR SOLVERS

The new SUNNonlinearSolver class provides an interface for supplying SUNDIALS integrators with problem-speciic and third-party nonlinear solvers. Like the other SUNDIALS base classes, this class deines a set of required and optional methods. Complete descriptions of all SUNNonlinearSolver methods can be found in the SUNDIALS documentation [45, Section 9.1].

Table 6. Required SUNNonlinearSolver operations

|  |  |
| --- | --- |
| Function | Description |
| SUNNonlinearSolver\_Type | Returns the solver type |

SUNNonlinSolGetType(SUNNonlinearSolver NLS)

|  |  |
| --- | --- |
| int SUNNonlinSolSetSysFn(SUNNonlinearSolver NLS , SUNNonlinSolSysFn SysFn) | Set a function pointer to the nonlinear system function |
| int SUNNonlinSolSolve(SUNNonlinearSolver NLS , | Solves the nonlinear system |

N\_Vector y0, N\_Vector ycor ,   
N\_Vector w, realtype tol ,   
booleantype callLSetup , void \*mem)

The three required nonlinear solver methods are listed in Table 6. These include the GetType method for querying a solver object about the nonlinear system formulation it expects: root-inding, F (u) = 0, or ixed-point, G(u) = 0. Based on this information, an integrator will use the SetSysFn method to supply the solver with a function to evaluate the appropriate nonlinear system. As nonlinear systems in SUNDIALS integrators are formulated in terms of a correction to the predicted solution, the Solve method is passed the initial guess for the state vector, y0, as well as the correction vector, ycor, which contains the initial correction on input and the inal correction on output. The solution is expected to satisfy the tolerance, tol, in the WRMS norm with weight vector, w. As discussed in Section 5, nonlinear solvers like Newton’s method may reuse linear system information between solves to reduce the cost of setting up the linear solver. The input lag, callLSetup, signals whether the linear system matrix and/or preconditioner should be updated.

Rather than relying directly on the SUNLinearSolver class, the optional SetLSetupFn and SetLSolveFn methods listed in Table 7 allow the integrators to provide the nonlinear solver with functions handling integrator-speciic actions for setting up and solving linear systems. These functions in turn utilize SUNLinearSolver objects. For nonlinear solvers that do not require a linear solver, or the solve is done within the nonlinear solver implementation, these optional methods do not need to be provided. Additionally, the optional SetConvTestFn method listed in Table 7 allows an integrator-speciic convergence test function to be provided. The default stopping test for nonlinear iterations is related to the subsequent local error test for a time step, with the goal of keeping the nonlinear iteration errors from interfering with local temporal error control. In each of the integrators, this test is based on an estimate of the solver convergence rate, and incorporates several heuristics for estimating the rate of convergence and detecting solver divergence. As the default convergence test may not be optimal for all nonlinear solvers or problems, this optional method also allows a user to attach a custom convergence test that is better-suited to the solver or application. Alternately, if the nonlinear solver includes its own internal convergence test, this optional function need not be provided.

With the new SUNNonlinearSolver class in place, the integrators were updated to utilize generic nonlinear solvers, and the existing solver implementations were combined into shared derived classes. This change greatly

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| Enabling New Flexibility in SUNDIALS | • | 0:17 |

Table 7. Subset of optional SUNNonlinearSolver operations

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Description | | |
| int SUNNonlinSolSetLSetupFn(SUNNonlinearSolver NLS ,   SUNNonlinSolLSetupFn SetupFn); | Set linear solver setup function | | |
| int SUNNonlinSolSetLSolveFn(SUNNonlinearSolver NLS ,   SUNNonlinSolLSolveFn SolveFn); | Set linear solver solve function | | |
| int SUNNonlinSolSetConvTestFn(SUNNonlinearSolver NLS ,  SUNNonlinSolConvTestFn CTestFn , void\* ctest\_data); | Set | convergence | test |
| function | | |

improves the lexibility of the time integrators, as third party nonlinear solver interfaces (e.g., the SUNNonlinear-Solver\_PetscSNES class utilizing PETSc’s SNES solver [9, 10]), or custom nonlinear solvers (as illustrated below) can now be created.

SUNNonlinearSolver Demonstration Problem

To demonstrate the lexibility provided by the new nonlinear solver class we consider a one-dimensional advection-reaction problem using a stif variation of the Brusselator model from chemical kinetics [37] utilizing a custom SUNNonlinearSolver. The system is given by

|  |  |
| --- | --- |
| ∂t= −c ∂u∂x+ A − (w + 1)u + vu2  ∂t= −c ∂v∂x+ wu − vu2 | |
| ∂w ∂t= −c ∂w∂x+ B − w | − wu |

where u, v, and w are chemical concentrations, t ∈ [0, 10] is time, x ∈ [0,b] is the spatial variable, c = 0.01 is the advection speed, A = 1 and B = 3.5 are the concentrations of chemical species that remain constant over space and time, and ϵ = 5 × 10−6is a parameter that determines the stifness of the system. The problem uses periodic boundary conditions and the initial condition is

u(0,x) = A + p(x), v(0,x) = B/A + p(x), w(0,x) = 3.0 + p(x),

p(x) = 0.1 exp�−2 (2x/b − 1)2�.

Spatial derivatives are discretized with irst order upwind inite diferences on a uniform mesh with nx points divided across px MPI tasks. The state is stored in a task-local serial vector wrapped by the MPIPlusX vector. The default third order IMEX method from ARKODE is used to evolve the system in time with the advection terms treated explicitly and the reaction terms implicitly.

As the reactions are purely local in space, the implicit system that must be solved at each stage can be decomposed into independent nonlinear systems at each grid point. To exploit this locality, the user code implements a custom problem-speciic SUNNonlinearSolver to perform MPI task-local solves with the local serial vectors. This eliminates nearly all parallel communication within the implicit solves i.e., global reductions for norms and orthogonalization in GMRES. The only global communication necessary is at the end of the solve phase to check if all the task-local nonlinear solves were successful. The source ile for this example is [ark\_ brusselator1D\_task\_local\_nls.c.](https://github.com/LLNL/sundials/blob/v6.2.0/examples/arkode/C_parallel/ark_brusselator1D_task_local_nls.c)

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:18 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

Here, the custom nonlinear solver member data is deined as the TaskLocalNewton\_Content structure con-taining an instance of the native SUNDIALS Newton solver and other data for solving the local nonlinear problem. The custom solver deines the required GetType, SetSysFn, and Solve methods as well as the optional methods: SetConvTestFn to use the integrator-provided convergence test function, Free to deallocate the solver, and GetNumConvFails to retrieve the number of solver failures. Additionally, the optional Initialize method is used, in this case, to provide the task-local instance of the SUNDIALS Newton solver with user-deined functions to evaluate the local residual (TaskLocalNlsResidual) and solve the local linear systems (TaskLocalLSolve). The main function creates a solver instance using the custom constructor, TaskLocalNewton, to setup the object as outlined in Section 3. Once the custom solver object is created, the example program attaches it to the integrator with ARKStepSetNonlinearSolver.

To compare this solution approach with what was possible before the addition of the SUNNonlinearSolver class, the example code provides an option to use the SUNDIALS Newton method to perform a global solve with GMRES as the linear solver and a user-deined, task-local preconditioner. Multiple tests are run with a ixed spatial resolution, b/nx = 10−3, and the domain size is increased with the number of MPI tasks; speciically we test with b ∈ {1, 2, 4, 8, 16, 32, 64, 128, 256} and px ∈ {40, 80, 160, 320, 640, 1280, 2560, 5120, 10240} with 40 MPI tasks per compute node on the Lassen supercomputer at Lawrence Livermore National Laboratory. Note the results below utilize only the CPUs (two IBM Power9 processors with 44 cores total) on Lassen. For results utilizing GPU acceleration and the MPIPlusX vector with various GPU task-local vectors see [11].

Fig. 3. Speedup results for various problem sizes for the SUNNonlinearSolver demonstration problem using a custom task-local nonlinear solver (red squares) and a global Newton solve (blue circles). Execution times are the average of 20 runs and each task-local solver data point is annotated with the approximate speedup achieved compared to the global solver.

Execution times for the global and task-local solvers are given in Figure 3. Although the reduction in synchro-nizations does not signiicantly impact the problem scaling, the custom solver results in speedups of 4x to more than 7x. As such, it is clear that the new nonlinear solver class provides important new capabilities that were not possible before the lexibility enhancements and can lead to signiicantly more eicient solver approaches by exploiting problem structure.

7 FORTRAN 2003 INTERFACES

Fortran has historically been heavily used in scientiic computing and remains widespread today. As such, although SUNDIALS is written in C, Fortran interfaces have been provided for many years. These Fortran 77 standard compliant interfaces have allowed Fortran users to access a subset of features from CVODE, IDA, ARKODE, and KINSOL and a limited number of the vector, matrix, and solver class implementations available to C/C++ users. While the Fortran 77 interfaces provided access to some SUNDIALS capabilities they also sufered from limited lexibility and sustainability. Notably, the lack of derived types in Fortran 77 was incompatible with SUNDIALS classes, and circumventing this limitation required accessing SUNDIALS classes using global memory, rendering the Fortran 77 interfaces non-threadsafe. As a result, it was diicult for Fortran users to supply application-speciic data layouts or custom solvers, and it was impossible to use multiple instances of SUNDIALS packages simultaneously in a multithreaded environment. Furthermore, maintaining these interfaces required signiicant manual upkeep of code to łgluež the interfaces to C; thus, the Fortran 77 interfaces were not sustainable in the long-term.

To address these limitations, we have introduced new Fortran 2003 standard compliant interfaces that provide access to all of the features in all six of the SUNDIALS packages (access to CVODES and IDAS were previously unsupported from Fortran 77) as well as most of the core class implementations. These new interfaces are

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| Enabling New Flexibility in SUNDIALS | • | 0:19 |

generated automatically using the SWIG-Fortran [48] tool, which only requires minimal additional interface code, much of which is common across SUNDIALS, to create bindings using the Fortran 2003 features for interoperability with ISO C code. When the C API changes in any package or class with an existing Fortran 2003 interface, the SWIG-Fortran tool is run, and the interface łgluež code is regenerated before package release. This process its nicely into a modern software development worklow and could be further automated to run as part of a continuous integration/continuous delivery pipeline. Thus, the Fortran 2003 interfaces have a low-maintenance cost and can quickly be extended to support new features and class implementations as they are added i.e., the Fortran 2003 interfaces are sustainable.

Since the Fortran 2003 interfaces are based on features of the Fortran 2003 standard for interopability with ISO C code, they allow Fortran users to continue to write idiomatic Fortran with few caveats, but to then interact with SUNDIALS in a way that closely resembles the C API. In fact, the Fortran 2003 interface API is close enough to the C API that the C API documentation serves dual-purpose as the Fortran 2003 documentation, modulo some information on data types and subtleties attributed to inherent diferences of C and Fortran. Function signatures match the C API with an łFž prepended to the function name e.g., FN\_VConst instead of N\_VConst. The generic SUNDIALS classes, such as the N\_Vector, are interfaced as Fortran derived-types of the same name. This convention means that users can easily provide custom, application-speciic implementations of the generic SUNDIALS classes that are written entirely in Fortran (as illustrated below). Data arrays (e.g., real arrays ) are interfaced as Fortran arrays of the corresponding type deined in the iso\_c\_binding intrinsic module. The exact mapping of SUNDIALS C types to Fortran 2003 types, as well as discussion of subtle usage diferences between the C and Fortran 2003 APIs, is given in the SUNDIALS documentation [45, Section 5.1].

Fortran 2003 Interface Demonstration Problem

A second version of the demonstration problem from Section 6 utilizing a custom nonlinear solver is also implemented in Fortran. The source code for this version of the problem is [ark\_brusselator1D\_task\_local\_ nls\_f2003.f90](https://github.com/LLNL/sundials/blob/v6.2.0/examples/arkode/F2003_parallel/ark_brusselator1D_task_local_nls_f2003.f90). As the Fortran interface closely mirrors the C API, the same steps to create a derived class implementation apply to the Fortran use case.

In this example, all of the data needed by the custom task-local nonlinear solver is contained in a user-deined module, nls\_mod. As such, the data is efectively global (albeit namespace protected). An alternative, thread safe approach would be to deine a new type for the solver data and attach it as the class member data. In this case, the module includes an instance of the SUNDIALS Newton method and other data needed for solving the task-local nonlinear systems as well as functions implementing required and optional nonlinear solver methods. An instance of the nonlinear solver object is created using the constructor subroutine TaskLocalNewton which leverages the corresponding base class constructor to create a SUNNonlinearSolver object and c\_funloc to set the function pointers in the VMT. In this case, the object’s content ield does not need to be set as all the data needed by the solver is accessible through the nls\_mod module. Once the solver and all its required data are created, the nonlinear solver object is attached to the integrator with the FARKStepSetNonlinearSolver function. As expected, the C and Fortran versions of the problem have the same integrator and solver statistics and nearly identical run times.

8 CONCLUSIONS

The growing complexity of scientiic simulations and recent advances in high performance computing systems have resulted in increased demand for utilizing new programming models and more varied algebraic solver options with the SUNDIALS time integrators and nonlinear solvers. To meet these demands, we have created new matrix, linear solver, and nonlinear solver classes following the same object-oriented design previously utilized by the SUNDIALS vector. These new additions adhere to a number of guiding principles that provide

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:20 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

greater lexibility while increasing the sustainability of the SUNDIALS packages, requiring minimal changes to user codes, and resulting in signiicant performance beneits in many situations.

The new classes allow for easier interoperability with and reuse of external and application-speciic data structures and solvers across SUNDIALS packages. In particular, the encapsulation of the nonlinear solvers from the time integrators provides lexibility in nonlinear solution approaches that was not previously available in SUNDIALS. Additionally, we have expanded the SUNDIALS vector class implementations to include a new many-vector capability, allowing for a vector comprised of various subvectors. This many-vector construct provides an easy way to accommodate multiphysics simulations as well as heterogeneous architectures where diferent vectors operate on data residing in potentially diferent memory spaces. An MPI+X vector was further added on top of the many-vector to streamline the introduction of new vectors supporting various approaches to on-node parallelism with heterogeneous computing platforms. New fused operations better utilize available hardware and enable development of reduced communication algorithms. Lastly, modern Fortran interfaces allow full access to the lexibility provided with the SUNDIALS classes.

Numerical tests have shown that the new infrastructure retains the existing behavior of the time integrators and does not increase the cost of simulations. Moreover, the results demonstrate the power of being able to easily compare the performance of fundamentally diferent integrators and solver approaches within the same problem. Signiicant speedups were also shown when leveraging the new nonlinear solver class to provide a problem-speciic solver exploiting the structure and locality in the nonlinear system.

With these new capabilities in place, future work includes implementing new vectors and matrices to accom-modate more types of on-node parallelism, investigating algorithms that leverage fused operations for greater performance, and expanding the set of native solver interfaces provided with SUNDIALS. Additionally, we plan to leverage the new many-vector implementation within SUNDIALS to exploit the structural features of diferent applications.

ACKNOWLEDGMENTS

The authors thank Alan Hindmarsh for numerous discussions and insights into details of integrator controls on solvers and design of integrator codes.

This research was supported by the Exascale Computing Project, (17-SC-20-SC), a collaborative efort of the U.S. Department of Energy Oice of Science and the National Nuclear Security Administration.

Support for this work was provided in part by the U.S. Department of Energy, Oice of Science, Oice of Advanced Scientiic Computing Research, Scientiic Discovery through Advanced Computing (SciDAC) Program through the Frameworks, Algorithms, and Scalable Technologies for Mathematics (FASTMath) Institute.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC. LLNL-JRNL-816631.

REFERENCES

[1] Donald G. Anderson. 1965. Iterative procedures for nonlinear integral equations. J. ACM 12, 4 (1965), 547ś560. [https://doi.org/10.1145/](https://doi.org/10.1145/321296.321305)  [321296.321305](https://doi.org/10.1145/321296.321305)   
[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. 1999. LAPACK Users’ Guide (third ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA. [https:](https://doi.org/10.1137/1.9780898719604)  [//doi.org/10.1137/1.9780898719604](https://doi.org/10.1137/1.9780898719604)   
[3] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, et al. 2021. MFEM: A modular inite element methods library. Comput. Math. with Appl. 81 (2021), 42ś74.

<https://doi.org/10.1016/j.camwa.2020.06.009>  
[4] Joel A. E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. 2019. CasADi ś A software framework for nonlinear optimization and optimal control. Math. Program. Comput. 11, 1 (2019), 1ś36. <https://doi.org/10.1007/s12532-018-0139-4>

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| Enabling New Flexibility in SUNDIALS | • | 0:21 |

[5] Daniel Arndt, Wolfgang Bangerth, Bruno Blais, Thomas C. Clevenger, Marc Fehling, Alexander V. Grayver, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Reza Rastak, Ignacio Thomas, Bruno Turcksin, Zhuoran Wang, and David Wells. 2020. The deal.II Library, Version 9.2. J. Numer. Math. 28, 3 (2020), 131ś146. <https://doi.org/10.1515/jnma-2020-0043>[6] Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. 1997. Implicit-Explicit Runge-Kutta Methods for Time-Dependent Partial Diferential Equations. Appl. Numer. Math. 25, 2-3 (Nov 1997), 151ś167. <https://doi.org/10.1016/S0168-9274(97)00056-1>  
[7] Uri M. Ascher, Steven J. Ruuth, and Brian T. R. Wetton. 1995. Implicit-Explicit Methods for Time-Dependent Partial Diferential Equations.

SIAM J. Numer. Anal. 32, 3 (1995), 797ś823. <https://doi.org/10.1137/0732037>  
[8] Steven F. Ashby and Robert D. Falgout. 1996. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater low simulations. Nucl. Sci. Eng. 124, 1 (1996), 145ś159. <https://doi.org/10.13182/NSE96-A24230>  
[9] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. 2021. PETSc Web page. https://petsc.org/. [10] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. 2021. PETSc Users Manual. Technical Report ANL-95/11 - Revision 3.15. Argonne National Laboratory.

[11] Cody J. Balos, David J. Gardner, Carol S. Woodward, and Daniel R. Reynolds. 2021. Enabling GPU Accelerated Computing in the SUNDIALS Time Integration Library. Parallel Comput. 108 (2021), 102836. <https://doi.org/10.1016/j.parco.2021.102836>  
[12] Kathryn Eleda Brenan, Stephen L. Campbell, and Linda Ruth Petzold. 1995. Numerical solution of initial-value problems in diferential- algebraic equations. SIAM Publications, Philadelphia, PA. <https://doi.org/10.1137/1.9781611971224>  
[13] Peter N. Brown, George D. Byrne, and Alan C. Hindmarsh. 1989. VODE: A variable-coeicient ODE solver. SIAM J. Sci. Statist. Comput.

10, 5 (1989), 1038ś1051. <https://doi.org/10.1137/0910062>  
[14] Peter N. Brown and Alan C. Hindmarsh. 1986. Matrix-free methods for stif systems of ODE’s. SIAM J. Numer. Anal. 23, 3 (1986), 610ś638.

<https://doi.org/10.1137/0723039>  
[15] Peter N. Brown and Alan C. Hindmarsh. 1989. Reduced storage matrix methods in stif ODE systems. Appl. Math. Comput. 31 (1989), 40ś91. <https://doi.org/10.1016/0096-3003(89)90110-0>  
[16] Peter N. Brown, Alan C. Hindmarsh, and Linda R. Petzold. 1994. Using Krylov methods in the solution of large-scale diferential-algebraic systems. SIAM J. Sci. Comput. 15, 6 (1994), 1467ś1488. <https://doi.org/10.1137/0915088>  
[17] Peter N. Brown, Alan C. Hindmarsh, and Linda R. Petzold. 1998. Consistent initial condition calculation for diferential-algebraic systems.

SIAM J. Sci. Comput. 19, 5 (1998), 1495ś1512. <https://doi.org/10.1137/S1064827595289996>  
[18] Peter N. Brown and Youcef Saad. 1990. Hybrid Krylov methods for nonlinear systems of equations. SIAM J. Sci. Statist. Comput. 11, 3 (1990), 450ś481. <https://doi.org/10.1137/0911026>  
[19] George D. Byrne and Alan C. Hindmarsh. 1975. A polyalgorithm for the numerical solution of ordinary diferential equations. ACM Trans. Math. Software 1, 1 (1975), 71ś96. <https://doi.org/10.1145/355626.355636>  
[20] George D. Byrne and Alan C. Hindmarsh. 1987. Stif ODE solvers: A review of current and coming attractions. J. Comput. Phys. 70, 1 (1987), 1ś62. <https://doi.org/10.1016/0021-9991(87)90001-5>  
[21] Yang Cao, Shengtai Li, Linda Petzold, and Radu Serban. 2003. Adjoint sensitivity analysis for diferential-algebraic equations: The adjoint DAE system and its numerical solution. SIAM J. Sci. Comput. 24, 3 (2003), 1076ś1089. <https://doi.org/10.1016/S0377-0427(02)00528-9>[22] Nicholas T. Carnevale and Michael L. Hines. 2006. The NEURON Book. Cambridge University Press, Cambridge, England. [https:](https://doi.org/10.1017/CBO9780511541612)  [//doi.org/10.1017/CBO9780511541612](https://doi.org/10.1017/CBO9780511541612)   
[23] Tim Davis. 2020. KLU Sparse Matrix Factorization Library. http://faculty.cse.tamu.edu/davis/suitesparse.html.

[24] Tim Davis and Ekanathan Palamadai Natarajan. 2010. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. ACM Trans. Math. Software 37, 3 (2010), 36:1ś36:17. <https://doi.org/10.1145/1824801.1824814>  
[25] Ron S. Dembo, Stanley C. Eisenstat, and Trond Steihaug. 1982. Inexact Newton methods. SIAM J. Numer. Anal. 19, 2 (1982), 400ś408.

<https://doi.org/10.1137/0719025>  
[26] James W. Demmel, John R. Gilbert, and Xiaoye S. Li. 1999. An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination. SIAM J. Matrix Anal. Appl. 20, 4 (1999), 915ś952. <https://doi.org/10.1137/S0895479897317685>  
[27] John E. Dennis, Jr and Robert B. Schnabel. 1996. Numerical methods for unconstrained optimization and nonlinear equations. SIAM Publications, Philadelphia, PA. <https://doi.org/10.1137/1.9781611971200>  
[28] Milo R. Dorr, J-L Fattebert, Michael E. Wickett, James F. Belak, and Patrice E.A. Turchi. 2010. A numerical algorithm for the solution of a phase-ield model of polycrystalline materials. J. Comput. Phys. 229, 3 (2010), 626ś641. <https://doi.org/10.1016/j.jcp.2009.09.041>[29] B.D. Dudson, M.V. Umansky, XQ Xu, P.B. Snyder, and H.R. Wilson. 2009. BOUT++: A framework for parallel plasma luid simulations.

Comput. Phys. Commun. 180, 9 (2009), 1467ś1480. <https://doi.org/10.1016/j.cpc.2009.03.008>  
[30] Edda Eich. 1993. Convergence results for a coordinate projection method applied to mechanical systems with algebraic constraints.

SIAM J. Numer. Anal. 30, 5 (1993), 1467ś1482. <https://doi.org/10.1137/0730076>

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| 0:22 | • | D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos |

[31] Robert D. Falgout and Jim E. Jones. 2000. Multigrid on massively parallel architectures. In Multigrid Methods VI. Springer, Berlin, Heidelberg, 101ś107. <https://doi.org/10.1007/978-3-642-58312-4_13>  
[32] William F. Feehery, John E. Tolsma, and Paul I. Barton. 1997. Eicient sensitivity analysis of large-scale diferential-algebraic systems.

Appl. Numer. Math. 25, 1 (1997), 41ś54. <https://doi.org/10.1016/S0168-9274(97)00050-0>  
[33] David J. Gardner, Jorge E. Guerra, François P. Hamon, Daniel R. Reynolds, Paul A. Ullrich, and Carol S. Woodward. 2018. Implicitś explicit (IMEX) RungeśKutta methods for non-hydrostatic atmospheric models. Geosci. Model Dev. 11, 4 (2018), 1497ś1515. [https:](https://doi.org/10.5194/gmd-11-1497-2018)  [//doi.org/10.5194/gmd-11-1497-2018](https://doi.org/10.5194/gmd-11-1497-2018)   
[34] David J. Gardner, Carol S. Woodward, Daniel R. Reynolds, Gregg Hommes, Sylvie Aubry, and A. Arsenlis. 2015. Implicit integration methods for dislocation dynamics. Modelling Simul. Mater. Sci. Eng. 23, 2 (2015), 025006. <https://doi.org/10.1088/0965-0393/23/2/025006>[35] David G. Goodwin, Raymond L. Speth, Harry K. Mofat, and Bryan W. Weber. 2018. Cantera: An Object-oriented Software Toolkit for Chemical Kinetics, Thermodynamics, and Transport Processes. [https://www.cantera.org.](https://www.cantera.org) <https://doi.org/10.5281/zenodo.1174508> Version 2.4.0.

[36] Ernst Hairer, Syvert P. Nùrsett, and Gerhard Wanner. 1993. Solving Ordinary Diferential Equations I: Nonstif Problems (2 ed.). Springer- Verlag, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-540-78862-1>  
[37] Ernst Hairer and Gerhard Wanner. 1996. Solving Ordinary Diferential Equations II: Stif and Diferential-Algebraic Problems (2 ed.).

Springer-Verlag, Berlin Heidelberg. <https://doi.org/10.1007/978-3-642-05221-7>  
[38] Alan C. Hindmarsh. 1980. LSODE and LSODI, two new initial value ordinary diferential equation solvers. ACM SIGNUM Newslett. 15, 4 (1980), 10ś11. <https://doi.org/10.1145/1218052.1218054>  
[39] Alan C. Hindmarsh. 1983. ODEPACK, a systematized collection of ODE solvers. In Scientiic Computing: Applications of Mathematics and Computing to the Physical Sciences (IMACS transactions on scientiic computation, Vol. 1), Robert S. Stepleman, M. Carver, William F. Ames, et al. (Eds.). North-Holland, Amsterdam, Netherlands, 55ś64.

[40] Alan C. Hindmarsh. 1992. Detecting stability barriers in BDF solvers. In Computational Ordinary Diferential Equations (Institute of Mathematics and its Applications Conference Series), J. R. Cash and I. Gladwell (Eds.). Oxford University Press, Oxford, England, 87ś96. [41] Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. 2005.

SUNDIALS: Suite of nonlinear and diferential/algebraic equation solvers. ACM Trans. Math. Software 31, 3 (2005), 363ś396. [https:](https://doi.org/10.1145/1089014.1089020)  [//doi.org/10.1145/1089014.1089020](https://doi.org/10.1145/1089014.1089020)   
[42] Alan C. Hindmarsh and Syvert P. Nùrsett. 1988. KRYSI, an ODE solver combining a semi-implicit Runge-Kutta method and a preconditioned Krylov method. Technical Report. Lawrence Livermore National Laboratory, Livermore, CA (USA); Trondheim Univ. (Norway). [43] Alan C. Hindmarsh and Linda R. Petzold. 1995. Algorithms and software for ordinary diferential equations and diferential-algebraic equations, Part I: Euler methods and error estimation. Comput. Phys. 9, 1 (1995), 34ś41. <https://doi.org/10.1063/1.168536>[44] Alan C. Hindmarsh and Linda R. Petzold. 1995. Algorithms and software for ordinary diferential equations and diferential-algebraic equations, Part II: Higher-order methods and software packages. Comput. Phys. 9, 2 (1995), 148ś155. <https://doi.org/10.1063/1.168540>[45] Alan C. Hindmarsh, Radu Serban, Cody J. Balos, David J. Gardner, Daniel R. Reynolds, and Carol S. Woodward. 2021. User Documentation for CVODE. v6.2.0.

[46] Intel. 2022. Developer Reference for Intel oneAPI Math Kernel Library. [https://www.intel.com/content/www/us/en/develop/documentation/](https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top.html)  [onemkl-developer-reference-c/top.html](https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top.html)   
[47] Kenneth R. Jackson and Ron Sacks-Davis. 1980. An alternative implementation of variable step-size multistep formulas for stif ODEs.

ACM Trans. Math. Software 6, 3 (1980), 295ś318. <https://doi.org/10.1145/355900.355903>  
[48] Seth Johnson, Andrey Prokopenko, and Katherine Evans. 2019. Automated Fortran-C++ Bindings for Large-Scale Scientiic Applications.

Comput. Sci. Eng. PP (2019), 1ś1. <https://doi.org/10.1109/MCSE.2019.2924204>  
[49] Ian Karlin et al. 2019. Preparation and Optimization of a Diverse Workload for a Large-Scale Heterogeneous System. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (Denver, Colorado) (SC ’19). Association for Computing Machinery, New York, NY, USA, Article 32, 17 pages. <https://doi.org/10.1145/3295500.3356192>  
[50] Christopher A. Kennedy and Mark H. Carpenter. 2003. Additive RungeśKutta schemes for convectionśdifusionśreaction equations.

Appl. Numer. Math. 44, 1-2 (Jan 2003), 139ś181. <https://doi.org/10.1016/S0168-9274(02)00138-1>  
[51] Christopher A. Kennedy and Mark H. Carpenter. 2019. Higher-Order Additive RungeśKutta Schemes for Ordinary Diferential Equations.

Appl. Numer. Math. 136 (Feb 2019), 183ś205. <https://doi.org/10.1016/j.apnum.2018.10.007>  
[52] Stefan J. Kollet, Reed M. Maxwell, Carol S. Woodward, Steve Smith, Jan Vanderborght, Harry Vereecken, and Clemens Simmer. 2010.

Proof of concept of regional scale hydrologic simulations at hydrologic resolution utilizing massively parallel computer resources. Water Resour. Res. 46, 4 (2010), 7 pages. <https://doi.org/10.1029/2009WR008730>  
[53] Shengtai Li and Linda Petzold. 2000. Software and algorithms for sensitivity analysis of large-scale diferential algebraic systems. J.

Comput. Appl. Math. 125, 1-2 (2000), 131ś145. <https://doi.org/10.1016/S0377-0427(00)00464-7>  
[54] Xiaoye S. Li and James W. Demmel. 2003. SuperLU\_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems. ACM Trans. Math. Software 29, 2 (June 2003), 110ś140. <https://doi.org/10.1145/779359.779361>

ACM Trans. Math. Softw.



|  |  |  |
| --- | --- | --- |
| Enabling New Flexibility in SUNDIALS | • | 0:23 |

[55] Benny Malengier, Pavol Kišon, James Tocknell, Claas Abert, Florian Bruckner, and Marc-Antonio Bisotti. 2018. ODES: a high level interface to ODE and DAE solvers. J. Open Source Softw. 3, 22 (2018), 165. <https://doi.org/10.21105/joss.00165>  
[56] Timothy Maly and Linda R. Petzold. 1996. Numerical methods and software for sensitivity analysis of diferential-algebraic systems.

Appl. Numer. Math. 20, 1-2 (1996), 57ś79. <https://doi.org/10.1016/0168-9274(95)00117-4>  
[57] Matthew J. McNenly, Russell A. Whitesides, and Daniel L. Flowers. 2015. Faster solvers for large kinetic mechanisms using adaptive preconditioners. Proc. Combust. Inst. 35, 1 (2015), 581ś587. <https://doi.org/10.1016/j.proci.2014.05.113>  
[58] NVIDIA. 2019. cuSOLVER library, release 10.1.243. [https://docs.nvidia.com/cuda/archive/10.1/pdf/CUSOLVER\_Library.pdf.](https://docs.nvidia.com/cuda/archive/10.1/pdf/CUSOLVER_Library.pdf) [59] James M. Ortega and Werner C. Rheinboldt. 2000. Iterative solution of nonlinear equations in several variables. SIAM Publications, Philadelphia, PA. <https://doi.org/10.1137/1.9780898719468>  
[60] Michael Pernice and Homer F. Walker. 1998. NITSOL: A Newton iterative solver for nonlinear systems. SIAM J. Sci. Comput. 19, 1 (1998), 302ś318. <https://doi.org/10.1137/S1064827596303843>  
[61] Linda R. Petzold. 1982. Description of DASSL: A diferential/algebraic system solver. Technical Report. Sandia National Laboratory, Livermore, CA (USA). <https://www.osti.gov/biblio/5882821>  
[62] Christopher Rackauckas and Qing Nie. 2017. DiferentialEquations.jl ś A performant and feature-rich ecosystem for solving diferential equations in Julia. J. Open Res. Softw. 5, 1 (2017), 10 pages. <https://doi.org/10.5334/jors.151>  
[63] Daniel R. Reynolds, David J. Gardner, Cody J. Balos, and Carol S. Woodward. 2019. SUNDIALS Multiphysics+MPIManyVector Performance Testing. Technical Report. Lawrence Livermore National Laboratory. <https://doi.org/10.2172/1567994>  
[64] Adrian Sandu. 2019. A Class of Multirate Ininitesimal GARK Methods. SIAM J. Numer. Anal. 57, 5 (Jan 2019), 2300ś2327. [https:](https://doi.org/10.1137/18M1205492)  [//doi.org/10.1137/18M1205492](https://doi.org/10.1137/18M1205492)   
[65] Martin Schlegel, Oswald Knoth, Martin Arnold, and Ralf Wolke. 2009. Multirate RungeśKutta schemes for advection equations. J.

Comput. Appl. Math. 226, 2 (Apr 2009), 345ś357. <https://doi.org/10.1016/j.cam.2008.08.009>  
[66] Martin Schlegel, Oswald Knoth, Martin Arnold, and Ralf Wolke. 2012. Implementation of multirate time integration methods for air pollution modelling. Geosci. Model Dev. 5, 6 (Nov 2012), 1395ś1405. <https://doi.org/10.5194/gmd-5-1395-2012>  
[67] Martin Schlegel, Oswald Knoth, Martin Arnold, and Ralf Wolke. 2012. Numerical solution of multiscale problems in atmospheric modeling. Appl. Numer. Math. 62, 10 (Oct 2012), 1531ś1543. <https://doi.org/10.1016/j.apnum.2012.06.023>  
[68] Henning Schmidt. 2007. SBaddon: High performance simulation for the Systems Biology Toolbox for MATLAB. Bioinformatics 23, 5 (2007), 646ś647. <https://doi.org/10.1093/bioinformatics/btl668>  
[69] Radu Serban and Alan C. Hindmarsh. 2005. CVODES: The sensitivity-enabled ODE solver in SUNDIALS. In Proceedings of the ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. (International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. Volume 6: 5th International Conference on Multibody Systems, Nonlinear Dynamics, and Control, Parts A, B, and C). American Society of Mechanical Engineers, 257ś269. <https://doi.org/10.1115/DETC2005-85597>  
[70] Lawrence F. Shampine. 1999. Conservation laws and the numerical solution of ODEs, II. Comput. Math. with Appl. 38, 2 (1999), 61ś72.

<https://doi.org/10.1016/S0898-1221(99)00183-2>  
[71] The SUNDIALS development team. 2022. SUNDIALS. <https://computing.llnl.gov/projects/sundials>  
[72] The SUNDIALS development team. 2022. SUNDIALS GitHub. <https://github.com/LLNL/sundials>  
[73] Stanimire Tomov, Jack Dongarra, and Marc Baboulin. 2010. Towards dense linear algebra for hybrid GPU accelerated manycore systems.

Parallel Comput. 36, 5-6 (June 2010), 232ś240. <https://doi.org/10.1016/j.parco.2009.12.005>  
[74] Christopher J. Vogl, Andrew Steyer, Daniel R. Reynolds, Paul A. Ullrich, and Carol S. Woodward. 2019. Evaluation of Implicit- Explicit Additive Runge-Kutta Integrators for the HOMME-NH Dynamical Core. J. Adv. Model. Earth Syst. 11, 12 (2019), 4228ś4244.

<https://doi.org/10.1029/2019MS001700>  
[75] Homer F. Walker and Peng Ni. 2011. Anderson acceleration for ixed-point iterations. SIAM J. Numer. Anal. 49, 4 (2011), 1715ś1735.

<https://doi.org/10.1137/10078356X>  
[76] Wolfram. 2020. IDA Method for NDSolve. <https://reference.wolfram.com/language/tutorial/NDSolveIDAMethod.html>  
[77] Weiqun Zhang, Andrew Myers, Kevin Gott, Ann Almgren, and John Bell. 2021. AMReX: Block-structured adaptive mesh reinement for multiphysics applications. Int. J. High Perform. Comput. Appl. 35, 6 (2021), 508ś526. <https://doi.org/10.1177/10943420211022811>

ACM Trans. Math. Softw.