The SUNDIALS Suite of Time Integrators and Nonlinear Solvers: Its Capabilities, Design, and Role in Ushering New Methods Into Scientific Applications

Carol S. Woodward (LLNL), Daniel R. Reynolds(SMU), David J. Gardner (LLNL), and Cody J. Balos (LLNL)

ICERM Workshop on Holistic Design of Time-Dependent PDE Discretizations, Jan. 11, 2022

Lawrence Livermore National Laboratory

CASC
Center for Applied Scientific Computing

LLNL-PRES-830663

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

Outline

- Why numerical software libraries are important
- Overview of the SUNDIALS library capabilities
- Overview of the SUNDIALS library design and use
- Examples of use
- The xSDK extreme-scale software development kit
- Concluding remarks

As we look toward the future, we expect greater capability along with disruptive changes in high performance computing systems

- **Extreme levels of concurrency**
  - Very high node and core counts
  - Increasingly deep memory hierarchies
- **Additional complexities**
  - Hybrid architectures
  - Manycore, GPUs, multithreading
  - Relatively poor memory latency and bandwidth
  - Challenges with fault resilience
  - Must conserve power – limit data movement
  - New (not yet stabilized) programming models
  - Etc.

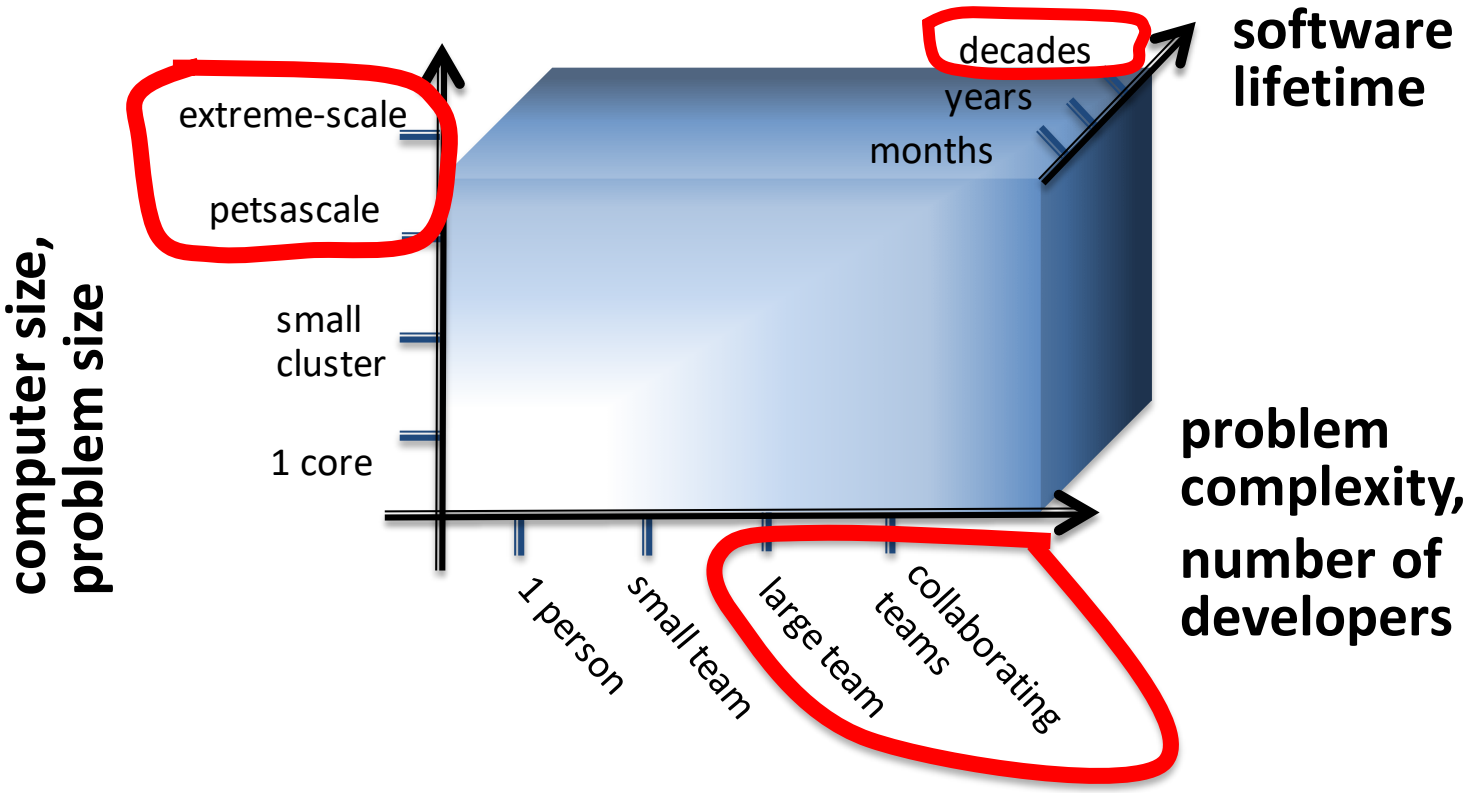
New capabilities will enable new computational science opportunities

Enough computational power to enable

- Multirate, multiscale, multicomponent, multiphysics simulations
- Uncertainty quantification and sensitivities for all simulations
- Simulations involving stochastic quantities
- Optimization over full-featured simulations
- Coupling of simulations and data analytics

Beyond interpretive simulations … working toward predictive science

Increasing complexity of future computational science problems leads to increasing complexity of software



Slide courtesy of L. McInnes (ANL)

Scientific software development encounters challenges from both the technical and sociological arenas

Technical

- All parts of the cycle can be under research
- Requirements change throughout the lifecycle as knowledge grows
- Importance of reproducibility
- Verification complicated by floating point representation
- Funding good software practices, like testing and documentation is hard to get
- The real world is messy, so is the software

Sociological

- Competing priorities and incentives
- Limited resources
- Perception of overhead with deferred benefit
- Need for interdisciplinary interactions

Science through computing is only as good as the software that produces it!

Some slide material courtesy of L. McInnes (ANL)

Despite challenges, opportunities abound for numerical software development improvements

- Better design, software practices, and tools are available
- Better software architectures: toolkits, libraries, frameworks
- Open-source software, community collaboration
- Greater recognition that good software practices can increase productivity

Working toward: **community software ecosystems for high-performance CSE**

Some slide material courtesy of L. McInnes (ANL)

Software libraries facilitate progress in computational science and engineering

Software library: a high-quality, encapsulated, documented, tested, and multiuse software collection that provides functionality commonly needed by application developers

- Organized for the purpose of being reused by independent (sub)programs
- User needs to know only
  - Library interface (not internal details)
  - When and how to use library functionality appropriately

Key advantages of software libraries

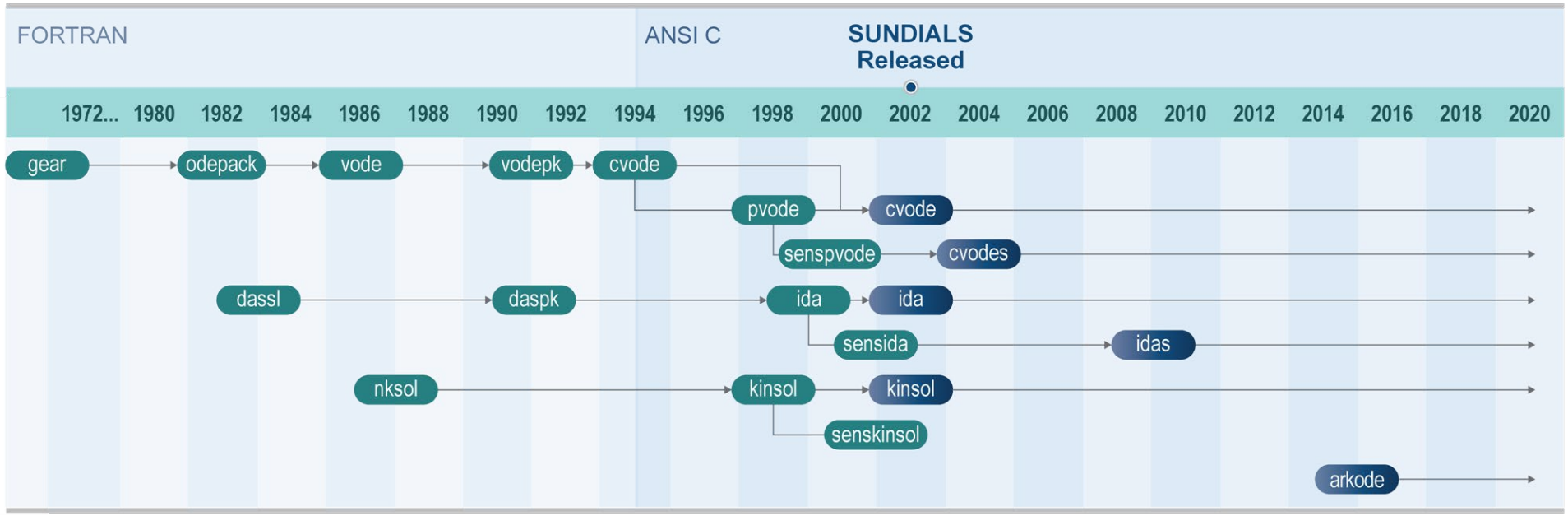
- Contain complexity
- Leverage library developer expertise
- Reduce application coding effort
- Encourage sharing of code, ease distribution of code

Slide courtesy of L. McInnes (ANL)

Why is reusable scientific software important for the math community?

- Allows us to verify algorithmic properties more easily
- Provides an insertion path for our work into scientific applications
- Broadens the impact of our work
- Can motivate new directions of research

LLNL has had a long history of software development for time integrators

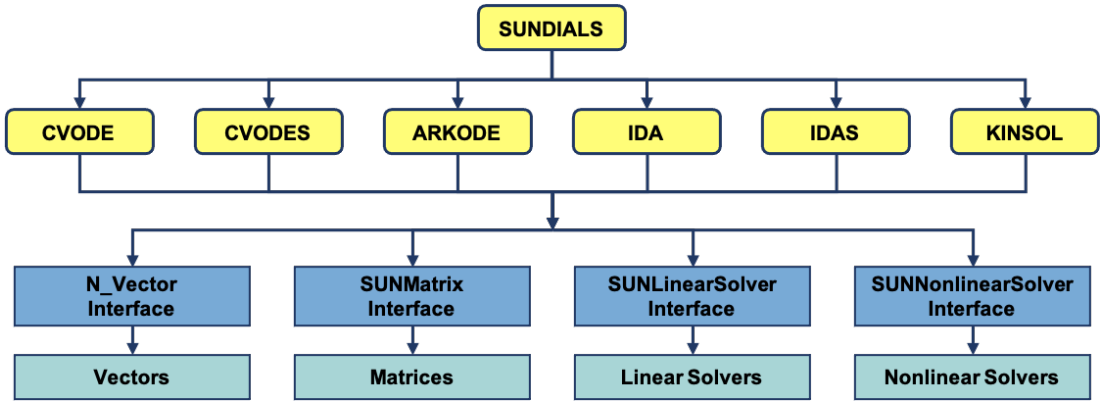


The 1980s and 1990s produced significant strides in the solution of ordinary differential equations and differential/algebraic equations:

- Stiff integrators that allow for changing step sizes
- Fully adaptive step and order integrators
- Novel Newton–Krylov methods allowing for matrix-free solvers
- Efficient and robust software implementations (ODEPACK -> CVODE, IDA -> SUNDIALS)

SUite of Nonlinear and DIfferential-ALgebraic Solvers

- SUNDIALS is a software library consisting of ODE and DAE integrators and nonlinear solvers
- Written in C with interfaces to Fortran
- Designed to be incorporated into existing codes
- Nonlinear and linear solvers and all data use is fully encapsulated from the integrators and can be user-supplied
- All parallelism is encapsulated in vector and solver modules and user-supplied functions
- Freely available; released under the BSD 3-Clause license (>120,000 downloads in 2021)
- Detailed user manuals are included with each package
- *Online documentation at readthedocs.org*



https://computing.llnl.gov/casc/sundials

SUNDIALS offers packages with linear multistep and multistage methods

- CVODE, IDA, and their sensitivity analysis variants (forward and adjoint), CVODES and IDAS, are based on linear multistep methods
  - CVODE solves ODEs, $\dot{y} = f(t, y)$
  - IDA solves DAEs, $F(t, y, \dot{y}) = 0$
  - Adaptive in both order and step sizes
  - Both packages include stiff BDF methods
  - CVODE includes nonstiff Adams-Moulton methods

- ARKODE is designed to work as an infrastructure for developing adaptive one-step, multistage time integration methods
  - Originally designed to solve $M\dot{y} = f_I(t, y) + f_E(t, y), \quad y(t_0) = y_0$

    $M(t)$ may be the identity or any nonsingular (and optionally time-dependent) mass matrix (e.g., FEM)
  - Includes multistage embedded methods which give rise to adaptive time steps
  - Three steppers: ARKStep (explicit, implicit, and additive IMEX Runge-Kutta methods), ERKStep (streamlined ERK implementation), and MRIStep (multirate infinitesimal step methods)
  - XBraid wrappers for methods from ARKStep which provide ARK methods to the XBraid parallel-in-time package

KINSOL solves systems of nonlinear algebraic equations, F(u) = 0

- Newton Solvers: update iterate via $u^{k+1} = u^k + s^k, k = 0, ..., 1$
  - Get update by solving: $J(u^k)s^k = -F(u^k) \qquad J(u) = \frac{\partial F(u)}{\partial u}$
  - Inexact method approximately solves this equation
- Dynamic linear tolerance selection for use with iterative linear solvers

$$\|F(x^k) + J(x^k)s^{k+1}\| \leq \eta^k \|F(x^k)\|$$

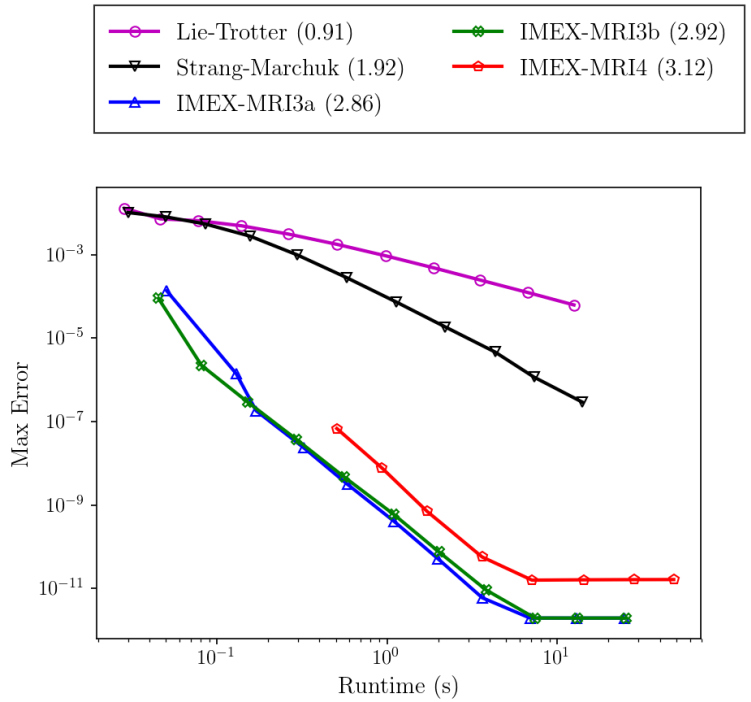
- Can separately scale equations and unknowns
- Backtracking and line search options for robustness
- KINSOL also solves fixed point and Picard iterations with acceleration

$$u^{k+1} = G(u^k), k = 0, 1, ...$$

$$F(u) \equiv Lu - N(u) \quad G(u) \equiv L^{-1}N(u) = u - L^{-1}F(u) \Rightarrow u^{k+1} = u^k - L^{-1}F(u^k)$$

The MRIStep (multirate infinitesimal step) module is our newest module in ARKODE and is still expanding in capability

- The new MRIStep module supports 2nd, 3rd, and 4th order multirate methods
- The slow time scale is integrated with implicit, explicit, or IMEX methods
- The slow time scale uses a user-defined Δt for the slow operator that can be varied between slow steps
- The fast time scale can call ARKStep and thus allows for explicit, implicit, or IMEX integration (user-supplied fast integrator is also supported)
- The fast time scale can use adaptive or fixed time step sizes
- Supports user-defined method tables for both time scales
- Currently available
  - 2nd and 3rd order multirate MIS methods
  - 4th order multirate MRI-GARK methods, explicit and solve decoupled implicit (Sandu, SINUM, 2019)
  - 3rd and 4th order multirate with IMEX splitting at the slow time scale (Chinomona and Reynolds, SISC, 2021)



Comparison of 3rd and 4th order IMEX-MRI methods in SUNDIALS with 1st and 2nd order splitting approaches on a 1D advection-diffusion-reaction test. The IMEX-MRI methods show greater accuracy and efficiency. Figure courtesy of R. Chinomona (SMU).

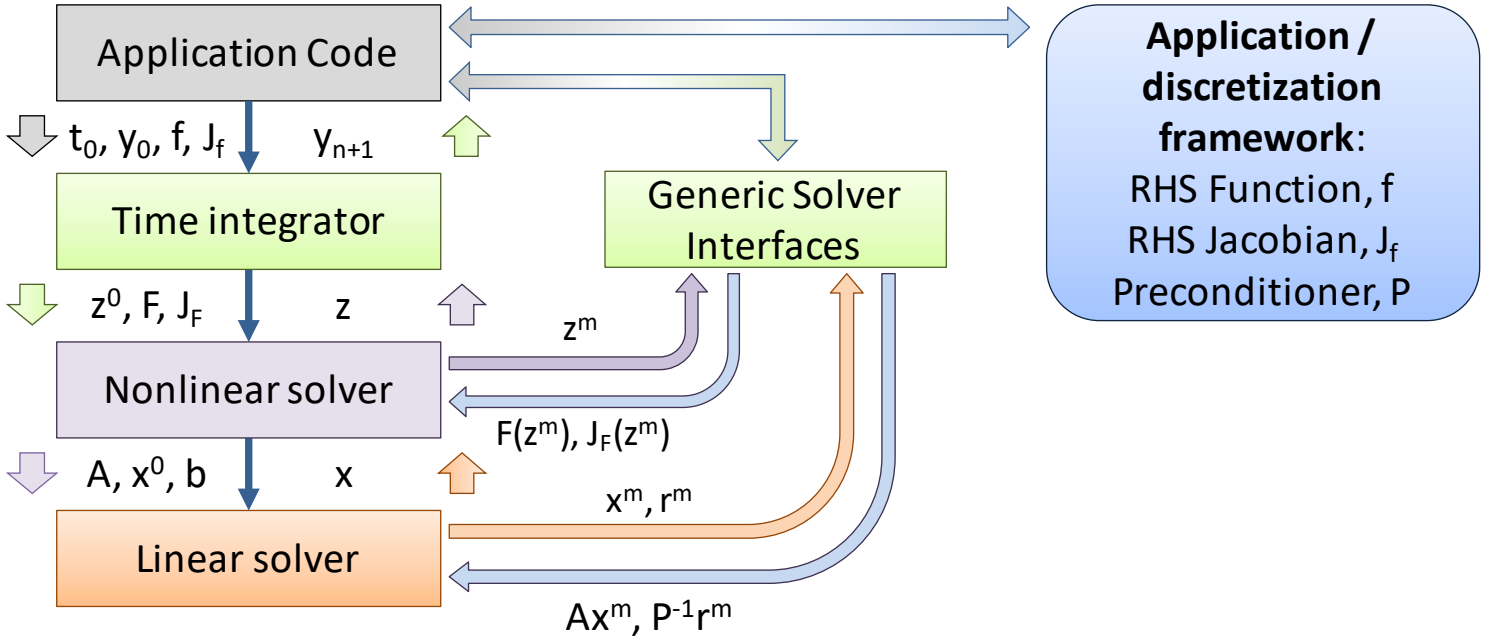
See Daniel Reynolds' talk from Monday

Many time integrators and nonlinear solvers can be implemented in ways that allow for very flexible software

- Most methods can be written in terms of operations on data, rather than assuming exactly what the data looks like and how it is laid out in memory
- Implicit time integrators can be made more efficient through control of properties of the nonlinear and linear solver, but these properties can be encapsulated away from the integrator
- Nonlinear solvers can be made more efficient through control of properties of the subsidiary linear solver, but these properties can be encapsulated
- Linear solvers may require detailed data information:
  - Iterative: only needs action of the linear operator on a matrix rather than the full matrix
  - Direct: Requires the matrix in specific formats
- Libraries can be designed to minimize need to interact with linear system data and thus allow for the application or linear solver to control the data layout

SUNDIALS uses modular design and control inversion to interface with application codes, external solvers, & encapsulate parallelism

- Control passes between the integrator, solvers, and application code as the integration progresses



Application / discretization framework:
RHS Function, f
RHS Jacobian, J_f
Preconditioner, P

Time integrator and nonlinear solver are agnostic of vector data layout and specific solvers used

- Nonlinear and linear solver modules are designed for generic systems

$$F(y) = 0 \qquad G(y) = y \qquad Ax = b$$

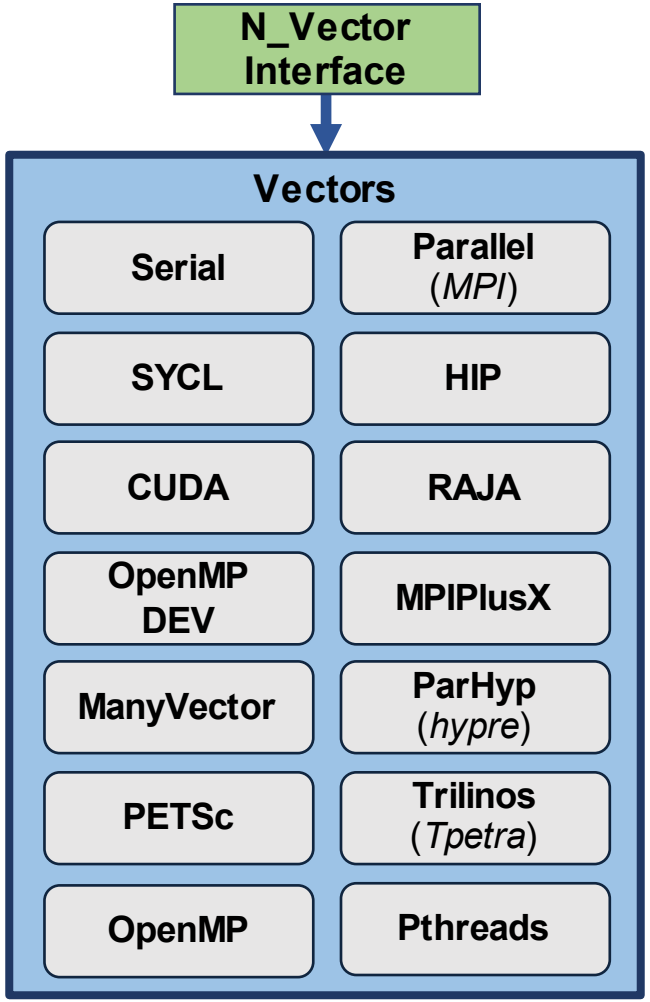
In developing SUNDIALS we adhered to basic guiding principles in setting up interfaces between integrators, solvers, and users

- Application Program Interfaces (APIs) for vectors, matrices, linear solvers, nonlinear solvers, and time integrators are based on the minimal required functionality; these encapsulate all parallelism
  - Although written in C, SUNDIALS is set up like C++ classes with a **content structure and a set of operations**
  - SUNDIALS allows users to supply custom versions of data structures and solvers
- Allow for the user to control as much as possible about the integrators and solvers
  - Include optional calls to allow for control of many features
  - Ensure the user controls specifics of third-party solvers
  - Assume as little information about parallelism as possible
- Keep the SUNDIALS packages easy to use
  - Intuitive interfaces
  - Detailed user documentation
  - User-friendly build system
  - Simple example programs
- Don't slow down the most common cases to cover all rare cases

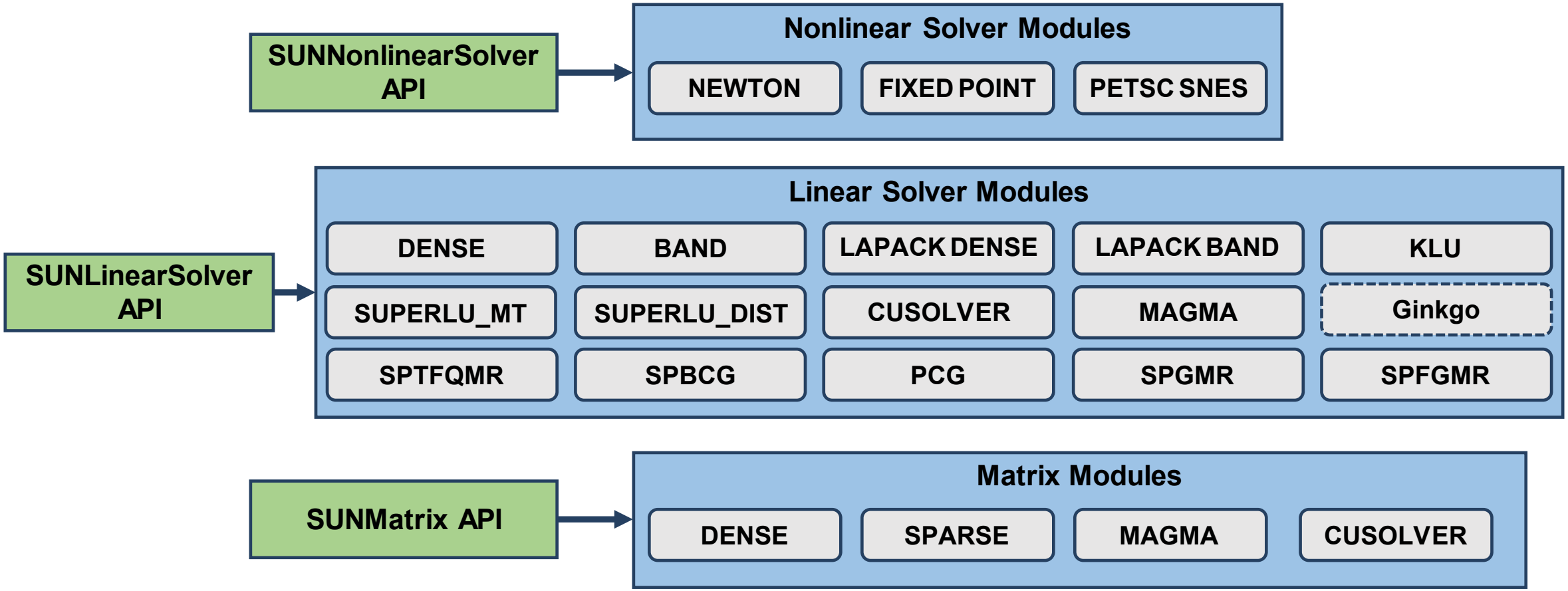
The SUNDIALS team gutted and redid the underlying software architecture over the last several years. While these guidelines seem obvious, reminders at many decision points were needed.

SUNDIALS packages modify solution data only through the methods in the NVector data class

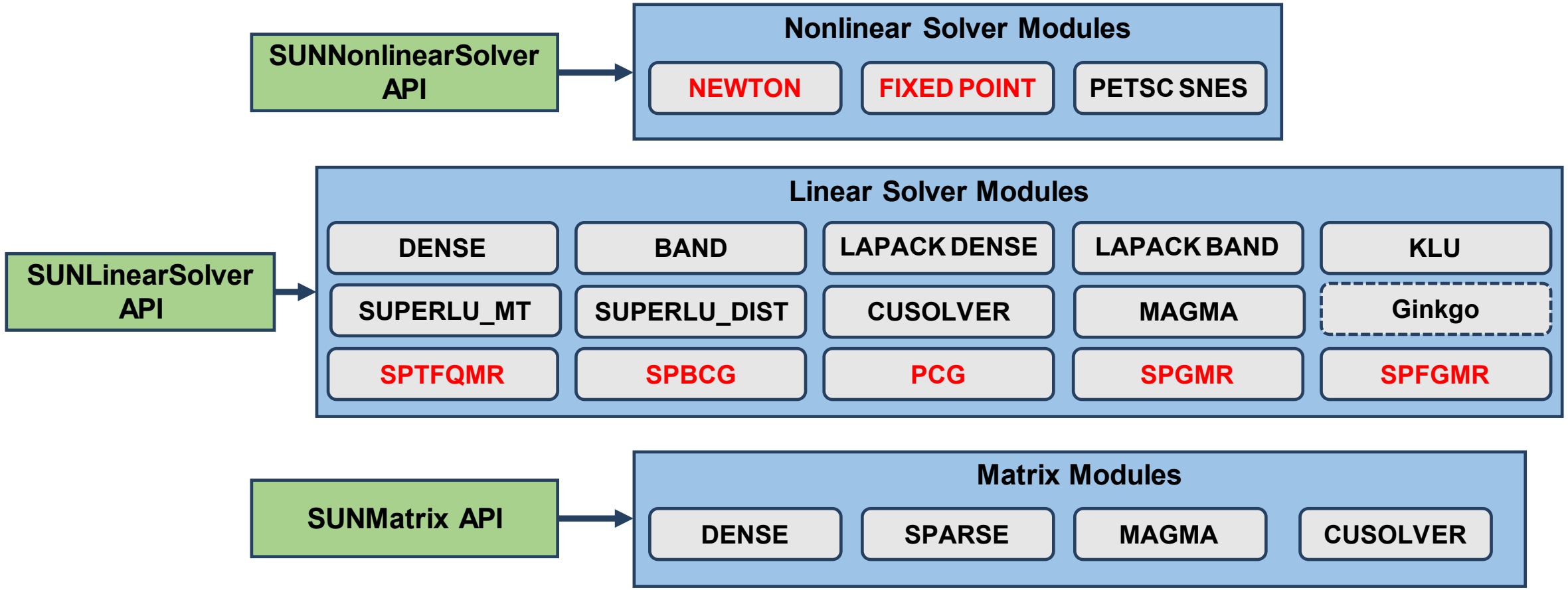
- Several NVector implementations are released with SUNDIALS:
  - CUDA, HIP, SYCL, RAJA (CUDA and HIP backends), and OpenMPDEV (target offload) vectors provide on-node GPU support
  - Parallel, ParHyp, PETSc, and Trilinos modules are MPI distributed
  - ManyVector and MPIPlusX modules provide support for hybrid computation
- Seems like a lot to support but…
  - vectors are simple and a lot of code can be reused
  - once the GPU platforms mature, can hopefully reduce options
- It is straightforward to implement a problem-specific NVector tailored to an application



Matrix and solver implementations supply a minimum set of functionality in order to be used underneath the time integrators



Matrix and solver implementations supply a minimum set of functionality in order to be used underneath the time integrators



Nonlinear solvers and matrix-free iterative linear solvers derive GPU support from vectors

SUNDIALS package use: first instantiate the subsidiary structures and solvers then pass to the integrator

- Initialize parallelism if needed
- Create simulation context
- Construct the initial state vector
- Call a Create function for the integrator – instantiates the integrator
- Call an Init function – specifies the problem (requires f function pointer(s)) and initial state
- Set integration tolerances
- Create a matrix object if needed
- Create linear solver, if needed, then set any linear solver optional inputs
- *Attach the linear solver module to the integrator*
- Create nonlinear solver
- *Attach the nonlinear solver* then set any nonlinear solver optional inputs
- Advance the solution in time – call to the integrator; this may be in a loop
- Get optional outputs
- Call relevant destructors for the solution vector, the integrator, and any algebraic solvers

Supplying the Initial Condition Vector(s)

- The user needs to construct an initial condition vector
- If using one of the SUNDIALS provided vectors, note that each provided module has a unique set of "constructors", e.g.
  - N_Vector N_VNew_Serial(sunindextype length, SUNContext sunctx);
  - N_Vector N_VMake_Cuda(sunindextype length, realtype *h_vdata, realtype *d_vdata , SUNContext sunctx);
  - N_Vector N_VMake_MPIPlusX(MPI_Comm comm, N_Vector x , SUNContext sunctx);
- Once an application creates a vector for their data, they fill it with the initial conditions for the problem and supply it to the integrator, who "clones" it to create its workspace.
- For PETSc, *hypre,* and Trilinos, the corresponding SUNDIALS NVector wrapper constructors take the native vector structure as their only input.

Supplying the IVP to the Integrator – RHS/Residual Functions

Once the problem data is encapsulated in a vector, all that remains for basic SUNDIALS usage is specification of the IVP itself:

- CVODE and ARKODE specify the IVP through right-hand side function(s):

```
int (*RhsFn)(realtype t, N_Vector y, N_Vector ydot, void *user_data)
```

- IDA specifies the IVP through a residual function:

```
int (*ResFn)(realtype t, N_Vector y, N_Vector ydot, N_Vector r,
  void *user_data)
```

- The *user_data pointer enables problem-specific data to be passed through the SUNDIALS integrator and back to the RHS/residual routine (i.e., no global memory).

CVODE/ARKODE RHS Functions

```c
/*
 * RHS function
 * The form of the RHS function is controlled by the flag passed as f_data:
 *   flag = RHS1 -> y' = -y
 *   flag = RHS2 -> y' = -5*y
 */

static int f(realtype t, N_Vector y, N_Vector ydot, void *f_data)
{
  int *flag;

  flag = (int *) f_data;

  switch(*flag) {
  case RHS1:
    NV_Ith_S(ydot,0) = -NV_Ith_S(y,0);
    break;
  case RHS2:
    NV_Ith_S(ydot,0) = -5.0*NV_Ith_S(y,0);
    break;
  }

  return(0);
}
```

Example:
cvDisc_dns.c

Initializing the Integrators – ARKODE

```c
/* Create fast solver memory structure and specify IMEX problem */
void *arkode_mem = ARKStepCreate(fe, fi, T0, y, sunctx);
if(check_retval((void *)arkode_mem, "ARKStepCreate", 0)) return(1);

/* Create solver memory structure and specify implicit problem */
void *arkode_mem = ARKStepCreate(NULL, f, T0, y, sunctx);
if(check_retval((void *)arkode_mem, "ARKStepCreate", 0)) return(1);

/* Create solver memory structure and specify explicit problem */
void *arkode_mem = ARKStepCreate(fe, NULL, T0, y, sunctx);
if(check_retval((void *)arkode_mem, "ARKStepCreate", 0)) return(1);
```

IMEX (top), implicit (middle), explicit (bottom)

```c
/* Create fast solver memory structure and specify IMEX problem */
void *inner_mem = ARKStepCreate(ffe, ffi, T0, y, sunctx);
if(check_retval((void *)inner_mem, "ARKStepCreate", 0)) return(1);

/* Set up fast integrator as normal */
int retval = ARKStepSet...(inner_mem, ...);
if(check_retval(&retval, "ARKStepSet...", 1)) return(1);

/* Create the inner stepper object wrapper */
int retval = ARKStepCreateMRIStepInnerStepper(inner_mem, stepper);
if(check_retval(&retval, "ARKStepCreateMRIStepInnerStepper", 1)) return(1);

/* Create slow solver memory structure and specify multirate problem */
void *arkode_mem = MRIStepCreate(fse, fsi, T0, y, *stepper, sunctx);
if(check_retval((void *)inner_mem, "ARKStepCreate", 0)) return(1);
```

Multirate with IMEX at fast time scale

Supplying Options to the Integrators

After constructing the integrator, additional options may be supplied through various "Set" routines (example from ark_heat1D_adapt.c):

```
/* Set routines */
int retval;
retval = ARKStepSetUserData(arkode_mem, (void *) udata);          /* Pass udata to user functions */
if (check_flag(&retval, "ARKStepSetUserData", 1))  return(1);

retval = ARKStepSetMaxNumSteps(arkode_mem, 1000);                 /* Increase max num steps */
if (check_flag(&retval, "ARKStepSetMaxNumSteps", 1))  return(1);

retval = ARKStepSStolerances(arkode_mem, rtol, atol);             /* Specify tolerances */
if (check_flag(&retval, "ARKStepSStolerances", 1))  return(1);

retval = ARKStepSetAdaptivityMethod(arkode_mem, 2, 1, 0, NULL);  /* Set adaptivity method */
if (check_flag(&retval, "ARKStepSetAdaptivityMethod", 1))  return(1);

retval = ARKStepSetPredictorMethod(arkode_mem, 0);                /* Set predictor method */
if (check_flag(&retval, "ARKStepSetPredictorMethod", 1))  return(1);
```

Advancing the Solution

Once all options have been set, the integrator is called to advance the solution toward t_{out}.

```
retval = IDASolve(ida_mem, tout, &tret, yy, yp, IDA_NORMAL);
if (check_retval(&retval, "IDASolve", 1))  return(1);
```

IDA

```
retval = CVode(cvode_mem, tout, y, &tret, CV_ONE_STEP);
if (check_retval(&retval, "CVode", 1))  return(1);
```

CVODE

```
retval = ARKStepEvolve(arkode_mem, tout, y, &tret, ARK_NORMAL);
if (check_retval(&retval, "ARKStepEvolve", 1))  return(1);
```

```
retval = MRIStepEvolve(arkode_mem, tout, y, &tret, ARK_ONE_STEP);
if (check_retval(&retval, "MRIStepEvolve", 1))  return(1);
```

ARKODE

Retrieving Output from the Integrators

```
long int nst, nfe, nsetups, netf, nni;
int retval;

retval = CVodeGetNumSteps(cvode_mem, &nst);
check_retval(&retval, "CVodeGetNumSteps", 1);

retval = CVodeGetNumRhsEvals(cvode_mem, &nfe);
check_retval(&retval, "CVodeGetNumRhsEvals", 1);

retval = CVodeGetNumLinSolvSetups(cvode_mem, &nsetups);
check_retval(&retval, "CVodeGetNumLinSolvSetups", 1);

retval = CVodeGetNumErrTestFails(cvode_mem, &netf);
check_retval(&retval, "CVodeGetNumErrTestFails", 1);

retval = CVodeGetNumNonlinSolvIters(cvode_mem, &nni);
check_retval(&retval, "CVodeGetNumNonlinSolvIters", 1);
```

Scalar-valued solver statistics from cvAdvDiffReac_kry.c

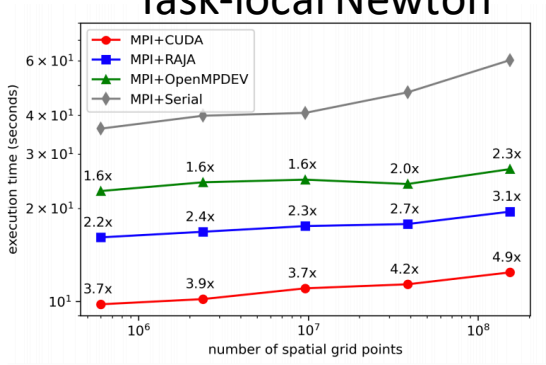
Software package flexibility allows for solver and programming model assessments

- 1D Advection-Reaction PDE solved with an IMEX method from ARKODE

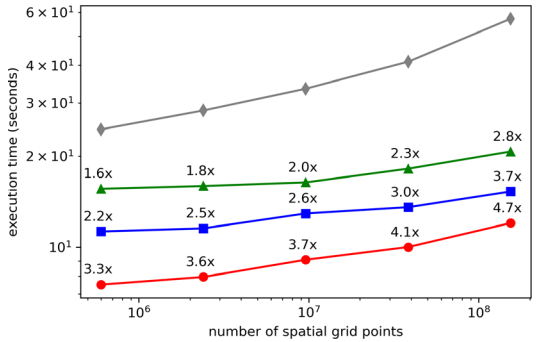
$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x} + A - (w+1)u + vu^2$$
$$\frac{\partial v}{\partial t} = -c\frac{\partial v}{\partial x} + wu - vu^2$$
$$\frac{\partial w}{\partial t} = -c\frac{\partial w}{\partial x} + \frac{B-w}{\epsilon} - wu$$

- Reactions treated implicitly; advection treated explicitly
- Nonlinear solver is either a "Task-local" Newton solve (solve per spatial node) + direct inversion or a Global Newton solve

Task-local Newton

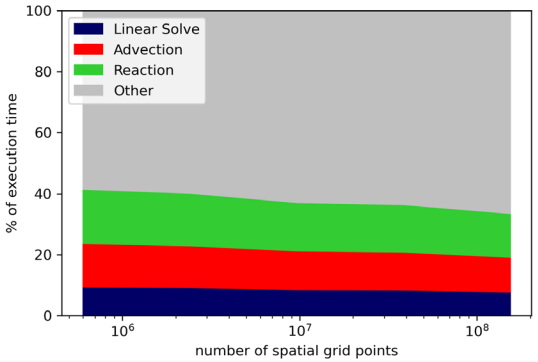


Global Newton

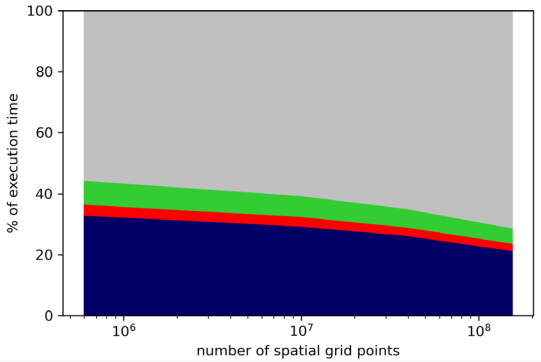


(Above) Weak scaling on 1, 4, 16, 64, and 256 Summit nodes, each with 6 MPI tasks per node (1 MPI task per GPU); MPI-only uses 40 MPI tasks per node. Annotations show speedup over the MPI+Serial vector. Greatest speedup achieved when using the CUDA vector and CUDA for the RHS

Task-local with CUDA



Task-local serial



Execution time breakdowns with and without CUDA with the Task-local solve; Choice of solver can have a significant impact on where time is spent within a simulation

Balos, Gardner, W., and Reynolds, 2021

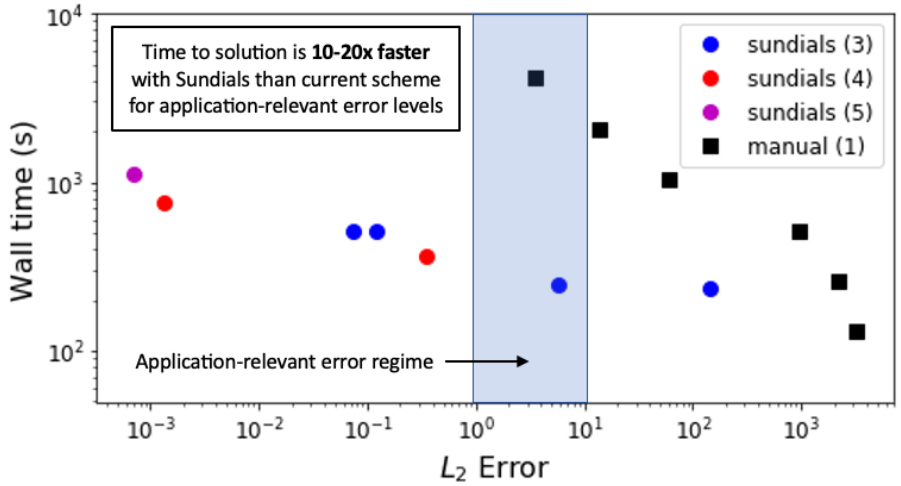
MEUMAPPS-SS phase field code was able to try new integrators easily once they had a single RHS function to call

- **MEUMAPPS-SS** is a phase-field code for simulating microstructure evolution during phase transformations
- Solves the Allen-Cahn and diffusion equations
  - Pseudo-spectral spatial discretization
  - **First order, fixed step implicit-explicit time integration**
- Utilizes Kokkos for performance portability
- Interfaced ARKODE with test application solving the Cahn-Hilliard benchmark problem:
  - Wrapped physics evaluations into a single function
  - Linear terms treated implicitly; nonlinear terms explicitly
  - Uses an application-specific linear solver
  - Uses SUNDIALS native vectors (CUDA and HIP)
- Adaptive 3rd order IMEX method in ARKODE provides a nice speedup over the native method in the relevant error regime

PFHub Benchmark 1a: Deterministic simulation of spinodal decomposition using the Cahn-Hilliard equation

$$\frac{\partial c}{\partial t} = \nabla \cdot \left(M \nabla \left(\frac{\partial f}{\partial c} - \kappa \nabla^2 c \right) \right)$$

$$\frac{\partial f}{\partial c} = 2\rho_s \left((c - c_\alpha)(c_\beta - c)^2 - (c - c_\alpha)^2 (c_\beta - c) \right)$$



Timing results on Ascent at ORNL (NVIDIA V100 GPUs) comparing the native fixed-step, first-order time integration method and adaptive, high-order methods from ARKODE with errors calculated using a small-time-step reference solution. In the application-relevant error regime, ARKODE is 10-20x faster at the same error levels. (Figure and results courtesy of Steve DeWitt of ORNL.)

DOE's climate nonhydrostatic dynamical core used ARKODE's flexibility to test method options

- **E3SM** is the US DOE's new climate code. In developing the nonhydrostatic dynamical core for the atmospheric model, it was initially unclear as to the best time integration approach.
- Non-hydrostatic models: compressible Navier Stokes supporting acoustic (sound) waves which have a negligible effect on climate, travel much faster than convection
- Spatial discretization is spectral elements in horizontal and second order mimetic FD in the vertical with vertical remaps; 2D parallel decomposition stores full vertical column(s) on single MPI task
- Application applies a hyperviscosity term to stabilize the discretization; first order split from dynamic system
- Applied horizontally explicit / vertically implicit (HEVI) splittings and IMEX integrators
- Developed accuracy criterion to determine "better" methods

Solve 5 hyperbolic equations:

$$\frac{\partial}{\partial t}\left(\frac{\partial \pi}{\partial \eta}\right) = -\nabla_\eta \cdot \left(\frac{\partial \pi}{\partial \eta}\mathbf{u}\right) - \frac{\partial}{\partial \eta}\left(\pi \frac{\mathrm{d}\eta}{\mathrm{d}t}\right)$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\nabla_\eta \times \mathbf{u} + 2\Omega) \times \mathbf{u} - \frac{1}{2}\nabla_\eta (\mathbf{u}\cdot\mathbf{u}) - \frac{\mathrm{d}\eta}{\mathrm{d}t}\frac{\partial \mathbf{u}}{\partial \eta} - \frac{1}{\rho}\nabla_\eta p$$

$$\frac{\partial w}{\partial t} = -\mathbf{u}\cdot\nabla_\eta w - \frac{\mathrm{d}\eta}{\mathrm{d}t}\frac{\partial w}{\partial \eta} - g(1-\mu), \quad \mu = \left(\frac{\partial p}{\partial \eta}\right) / \left(\frac{\partial \pi}{\partial \eta}\right),$$

$$\frac{\partial \Theta}{\partial t} = -\nabla_\eta \cdot (\Theta \mathbf{u}) - \frac{\partial}{\partial \eta}\left(\Theta \frac{\mathrm{d}\eta}{\mathrm{d}t}\right), \quad \Theta = \frac{\partial \pi}{\partial \eta}\theta,$$

$$\frac{\partial \phi}{\partial t} = -\mathbf{u}\cdot\nabla_\eta \phi - \frac{\mathrm{d}\eta}{\mathrm{d}t}\frac{\partial \phi}{\partial \eta} + gw,$$

where π is hydrostatic pressure, η is vertical coordinate, $\mathbf{u}$ and w are horizontal and vertical velocities, θ is potential temperature, and ϕ is geopotential.

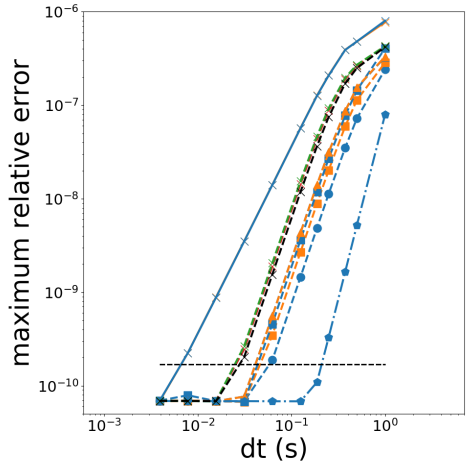
Key: hydrostatic model and nonhydrostatic terms.

We split the system so that if $y = \left(u, v, w, \phi, \Theta, \frac{\partial \pi}{\partial \eta}\right)^T$ then the stiff part of the operator is:

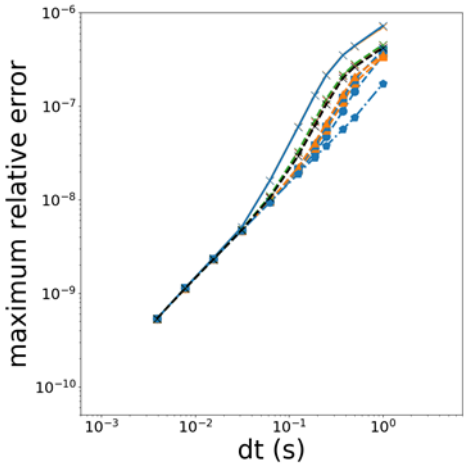
$$s(y) = (0, 0, -g(1-\mu), gw, 0, 0)^T$$

ARKODE was interfaced with the HOMME-NH dynamical core to test numerous integrator options

- "Taught" ARKODE how to perform vector arithmetic directly on pre-allocated vector objects
- Implemented a system for ARKODE to "check out" and "check in" temporary vectors, in lieu of standard allocation / deallocation.
- IMEX/HEVI Splittings: Repurposed existing physics routines to provide the IMEX splitting(s): fE and fI
- HOMME-NH:
  - Tridiagonal linear systems for vertical velocity on each MPI task
  - Post-process result for update
- Tried 22 IMEX methods including recently developed ARK methods
  - Steyer: IMKG methods using Kinnmark & Gray's 1984 results to increase explicit stability while limiting both tables to only 2 stored stages at any time (Steyer, Vogl, Taylor, Guba, 2019)
  - Reynolds: O(Dt^3), 5-stage method for max. imaginary axis coverage (Vogl, Steyer, Reynolds, Ullrich, and W., 2020)



All ARK methods converged at expected order on linear gravity wave test without hyperviscosity and first order with hyperviscosity

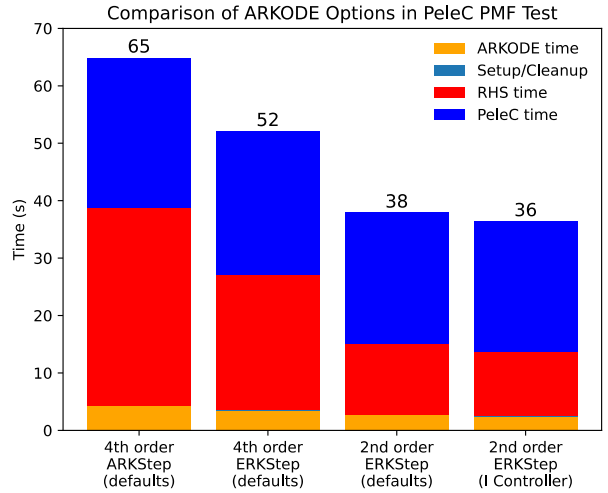


- Developed recommendation for methods to use in E3SM and metrics for testing new methods
  - Steyer's methods were fastest
  - Reynolds' method: most accuracy but 25% slower
- ARKODE was added to E3SM as a testing vehicle

Vogl, Steyer, Reynolds, Ullrich, and W., JAMES, 2020

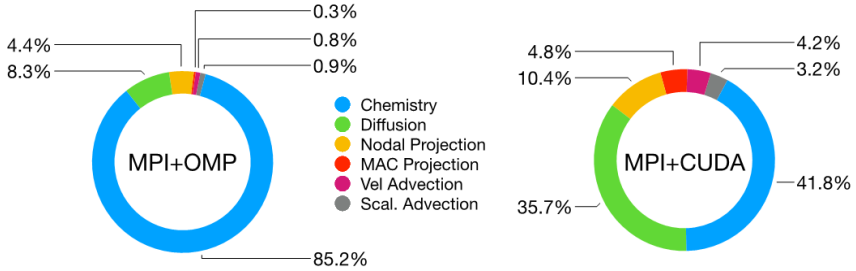
Work with Pele project shows role libraries have in helping applications transition between programming models

- The Pele codes use AMReX for structured grid adaptive mesh refinement within reacting flow simulations and SUNDIALS integrators to evolve the combustion mechanism w/in each grid cell
- **PelePhysics:** Interfaces to CVODE and ARKODE, batched solvers allow for integrating mechanisms with stiff integrators on an array of GPUs with differing programming models (CUDA, HIP, SYCL). This infrastructure is available to both PeleC and PeleLM
- **PeleLM:** Uses CVODE; solver options – iterative w/ and w/o preconditioners or direct batched
- **PeleC:** Less stiff problems; evaluating options with explicit integrator in ARKODE and CVODE

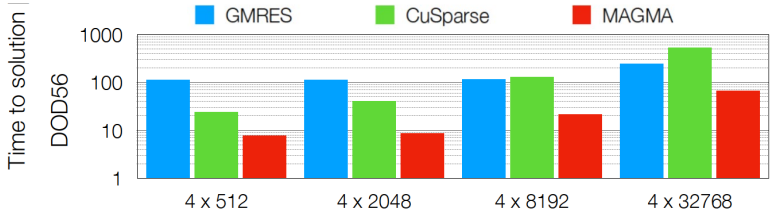


Runtimes for PeleC pre-mixed flame test on two Summit nodes (12 GPUs) using ARKODE. Saw 44% reduction in runtime w/ algorithm choices.

PeleLM Profiles



Through introduction of CUDA in AMReX and SUNDIALS, the lion's share of the PeleLM port to CUDA was completed. Once running with GPUs, the chemistry integration went from being significantly dominant to 41% of runtime. Speedups of ~6x observed on Summit.



Comparisons of linear solvers with PelePhysics using CVODE on Eagle (V100) system showing runtime benefits of MAGMA linear solvers.

Bottom figures and results courtesy of Lucas Esclapez, Marc Day, and Jon Rood

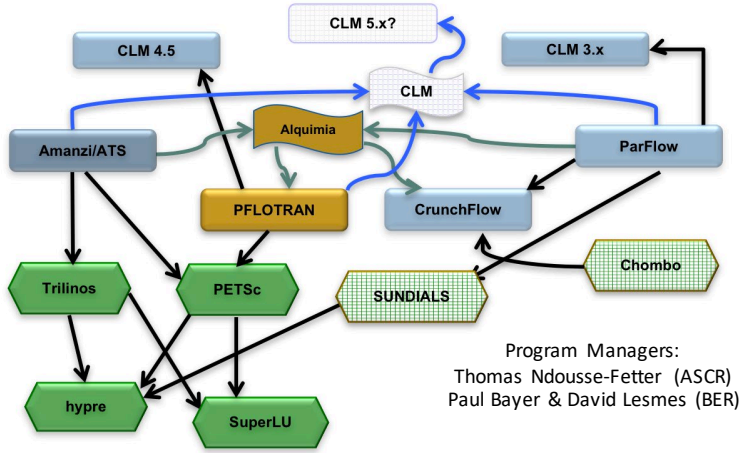
Software libraries are not enough: the xSDK effort was started to address challenges with using multiple libraries at once

Next-generation scientific simulations require combined use of packages

- Installing multiple independent software packages is error prone
  - Need consistency of compiler (+version, options), 3rd-party packages, etc.
  - Namespace and version conflicts make simultaneous build/link of packages difficult
- Multilayer interoperability requires careful design

xSDK history: Work began in ASCR/BER partnership, IDEAS project (Sept 2014)

Needed for multiscale, multiphysics integrated surface-subsurface hydrology models



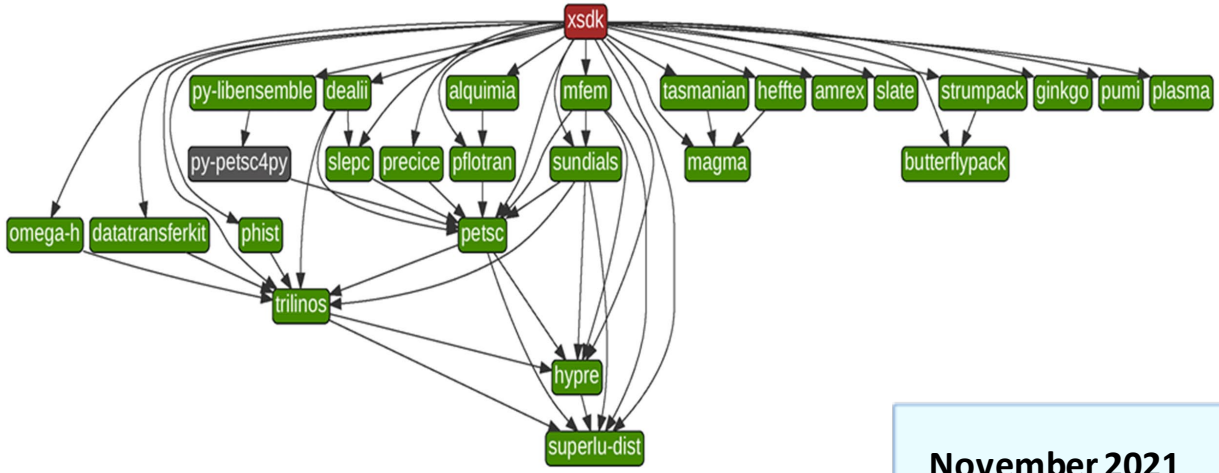
Prior to the xSDK effort, could not build required libraries into a single executable due to many incompatibilities

xSDK (Extreme-scale Scientific Software Kit) brings together many DOE libraries for improved cohesiveness

- Provides infrastructure and interoperability for independently developed mathematical libraries to support Exascale Computing Project (ECP) applications
- Achieves cohesiveness through a set of **community policies** adapted by all member libraries for seamless build, improved software quality, sustainability, and portability
- **Provide regular releases and documentation, including testing** on a variety of platforms, including key platforms at Argonne's Leadership Computing Facility (ALCF), NERSC at LBNL, and Oak Ridge's Leadership Computing Facility (OLCF)
- Each xSDK member package uses or can be used with one or more xSDK packages, and the connecting interface is regularly tested for regressions.
- Each member package maintains a spack package that is in turn used by the xSDK spack package to aid installation of the full xsdk. (Spack is a package manager: https://spack.io/).

xSDK Version 0.7.0 was released in November 2021

- Original xSDK math libraries: **hypre, PETSc, SuperLU, Trilinos**
- Added Dec 2017: **MAGMA, MFEM, SUNDIALS**
- Added Dec 2018: **AMReX, deal.II, DTK, Omega_h, PHIST, PLASMA, PUMI, SLEPc, STRUMPACK, TASMANIAN**
- Added Nov 2019: **ButterflyPACK, Ginkgo, libEnsemble, preCICE**
- Added Nov 2020: **heFFTe, SLATE**



November 2021
- 24 math libraries
- 2 domain components
- 16 mandatory xSDK community policies
- Spack xSDK installer

Tested on key machines at ALCF, NERSC, OLCF, also Linux, Mac OS X

xSDK: https://xsdk.info

Building the foundation of an extreme-scale scientific software ecosystem

xSDK community policies: Help address challenges in interoperability and sustainability of software developed by diverse groups at different institutions https://github.com/xsdk-project/xsdk-community-policies

xSDK compatible package: must satisfy the mandatory xSDK policies (M1, ..., M16)

Topics include: configuring, installing, testing, MPI usage, portability, contact and version information, open source licensing, namespacing, and repository access

Also specify **recommended policies**, which currently are encouraged but not required (R1, ..., R8)

Topics include: public repository access, error handling, freeing system resources, and library dependencies, documentation quality

xSDK member package:

(1) Must be an xSDK-compatible package, *and*

(2) it uses or can be used by another package in the xSDK, and the connecting interface is regularly tested for regressions.

xSDK policies 0.6.0: Oct 2020

- Facilitate combined use of independently developed packages

Impact:

- Improved code quality, usability, access, sustainability
- Foundation for work on deeper levels of interoperability and performance portability

We encourage feedback and contributions!

xSDK community policies

now also on github:

https://github.com/xsdk-project/xsdk-community-policies

We welcome feedback. What policies make sense for your software?

https://xsdk.info/policies

Mandatory xSDK policies:

M1. Support portable installation through Spack. (includes xSDK Spack variant guildelines)

M2. Provide a comprehensive test suite.

M3. Employ user-provided MPI communicator.

M4. Give best effort at portability to key architectures.

M5. Provide a documented, reliable way to contact the development team.

M6. Respect system resources and settings made by other previously called packages.

M7. Come with an open source license.

M8. Provide a runtime API to return the current version number of the software.

M9. Use a limited and well-defined symbol, macro, library, and include file name space.

M10. Provide an accessible repository (not necessarily publicly available).

M11. Have no hardwired print or IO statements.

M12. Allow installing, building, and linking against an outside copy of external software.

M13. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.

M14. Be buildable using 64-bit pointers. 32 bit is optional.

M15. All xSDK compatibility changes should be sustainable.

M16. Have a debug build option.

Recommended xSDK policies:

R1. Have a public repository.

R2. Possible to run test suite under valgrind in order to test for memory corruption issues.

R3. Adopt and document consistent system for error conditions/exceptions.

R4. Free all system resources it has acquired as soon as they are no longer needed.

R5. Provide a mechanism to export ordered list of library dependencies.

R6. Provide versions of dependencies.

R7. Have README, SUPPORT, LICENSE, and CHANGELOG file in top directory.

R8. Provide sufficient documentation to support use and further development.

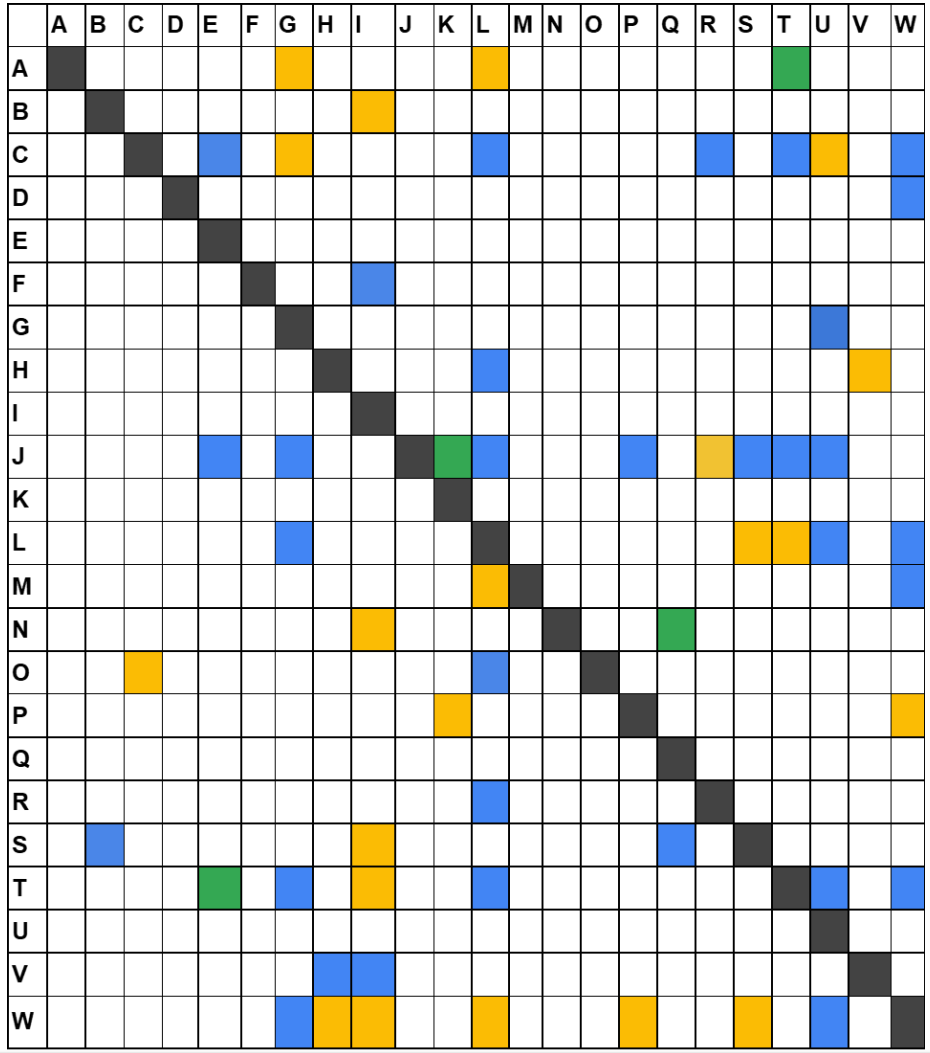
Several deeper interoperabilities exist and are maintained between member packages

Current and planned interoperabilities between xSDK libraries:

Legend:

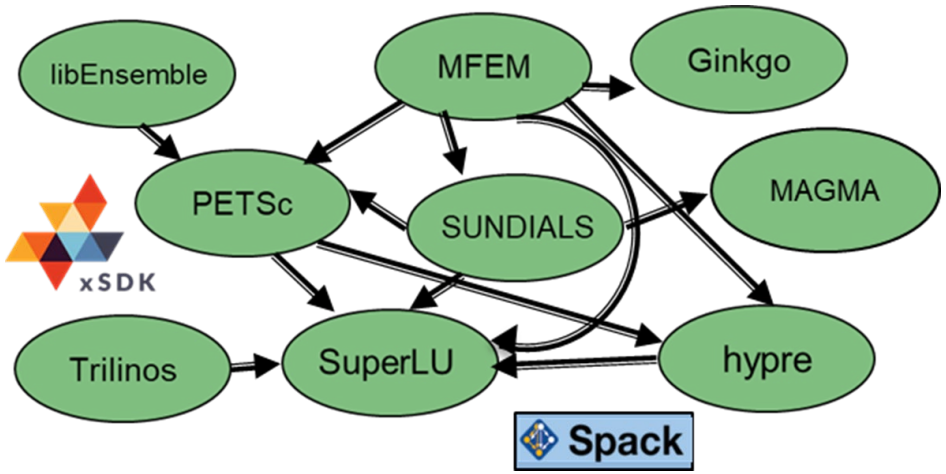
A = AMReX, **B** = ButterflyPACK, **C** = deal.II,
D = DataTransferKit, **E** = Ginkgo, **F** = heFFTe,
G = hypre, **H** = libEnsemble, **I** = MAGMA,
J = MFEM, **K** = Omega_h, **L** = PETSc,
M = PHIST, **N** = PLASMA, **O** = preCICE,
P = PUMI, **Q** = SLATE, **R** = SLEPc,
S = STRUMPACK, **T** = SUNDIALS, **U** = SuperLU
V = TASMANIAn, **W** = Trilinos

Key	
Yellow	Interoperability exists
Blue	Interoperability exists and is enabled in xSDK Spack
Green	Interoperability is planned
Black(ish)	Diagonal



The xSDK maintains several multi-library example codes demonstrating interoperability

- Second release of example codes (xsdk-examples v. 0.2.0) is available in a github repository: https://github.com/xsdk-project/xsdk-examples
- The example codes are a demonstration of interoperability between xSDK libraries and provide training for xSDK library users interested in using these capabilities.
- Building and running the example codes will enable better testing of the xSDK in future releases.
- Spack package and build systems were created and successfully tested for all example codes.
- Newest examples exhibit interoperabilities in GPU environments (CUDA for now)

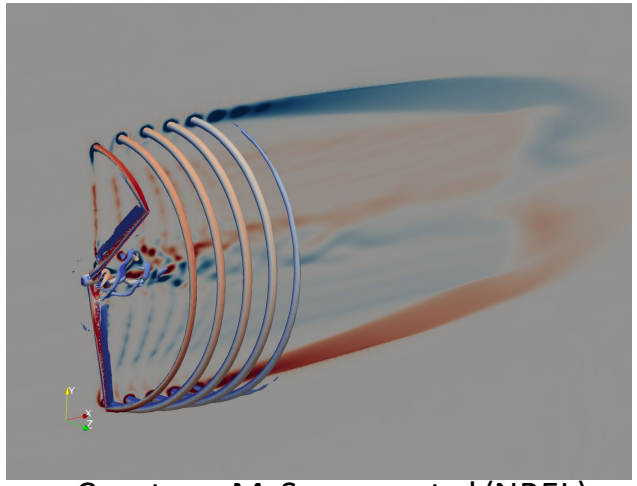


The xSDK uses a Gitlab continuous integration setup to regularly test interoperable compilation and deeper interoperabilities

- Using Gitlab CI (pipeline) infrastructure at https://gitlab.com/xsdk-project/spack-xsdk/-/pipelines
- Regularly run multiple tests per pipeline on latest Spack development branch: spack install xsdk
  - Linux with GNU and Intel compilers
  - Linux KNL with Intel compilers
  - MacOS with Clang/gfortran compilers
- Testing for xSDK release additionally includes key platforms at ALCF, NERSC, and OLCF
- Have started regular testing of development version of subsets of xSDK libraries
- Working towards a plan to improve xSDK testing with view towards sustainability

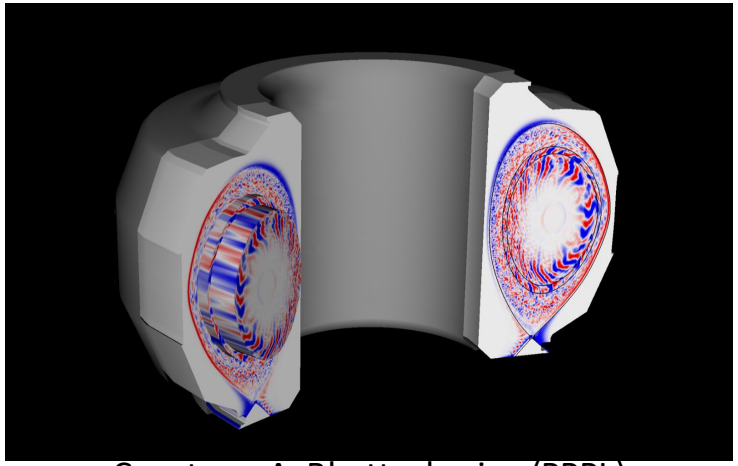
Several Exascale Computing Project applications use multiple numerical libraries in their simulation frameworks

- Simulations for better understanding complex flow physics in wind farms (ExaWind)
  - Uses AMReX, Trilinos, hypre
  - Plans to use SUNDIALS, MAGMA, SuperLU



Courtesy: M. Sprague et al (NREL)

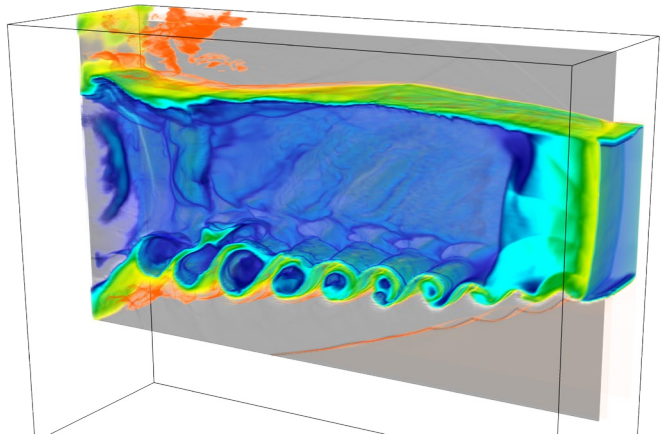
- High-Fidelity Whole Device Model of Magnetically Confined Fusion Plasma (WDMApp)
  - Core-edge coupled turbulence simulation in a realistic tokamak geometry
  - Color contours represent perturbed plasma density.
  - Uses hypre, PETSc, MAGMA, SLEPc, SuperLU



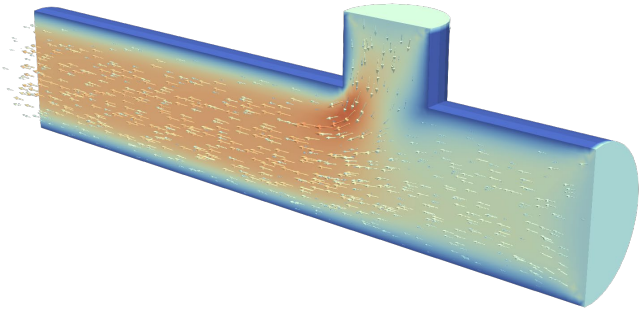
Courtesy: A. Bhattacharjee (PPPL)

Applications in Center for Efficient Exascale Discretizations ECP project also use multiple xSDK libraries

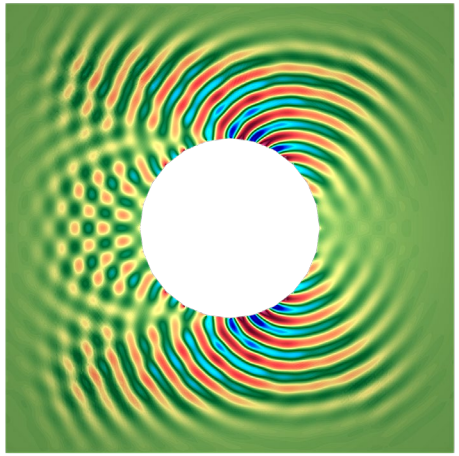
- xSDK highly relevant for CEED
- CEED uses or plans to use the following libraries in various applications:
  - MFEM, MAGMA, hypre, PETSc, SUNDIALS, SuperLU, Ginkgo, PUMI and STRUMPACK



Laser-driven radiating Kevin-Helmholtz instability using a high-order multi-material ALE radiation-hydrodynamics discretization



fully implicit mass transport



Electromagnetic wave scattering (tokamak fusion modeling)

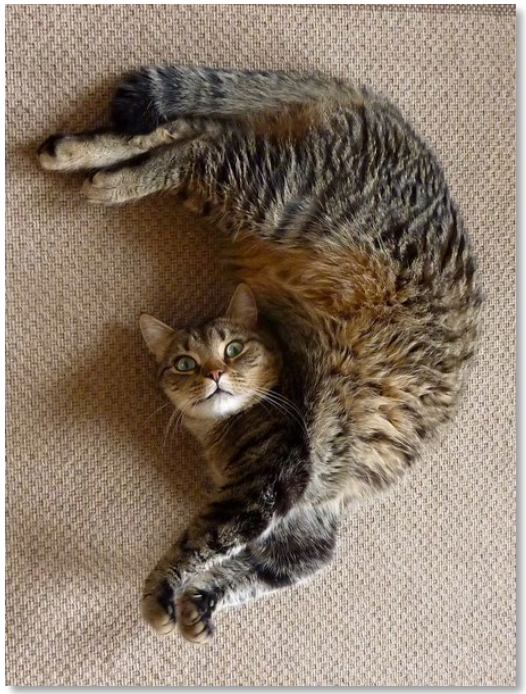
Courtesy: Tzanio Kolev (LLNL)

What's new in SUNDIALS?

- Multirate methods
  - IMEX-MRI (Chinomona and Reynolds)
  - MRI-GARK methods (Sandu)
  - Custom fast integrators in the multirate stepper, including an example that shows use of CVODE as the fast, inner stepper
- Support for SYCL-based direct linear solvers for use on Intel GPUs
- New performance assessment layer and performance assessments with Caliper
- Planned: greater interoperability to discretization and other solver packages
  - AMReX (structured adaptive mesh refinement package) – multifab-based vector for SUNDIALS
  - Chombo (structured adaptive mesh refinement package) – Chombo vector for SUNDIALS
  - MFEM (Modular finite element methods) – integrators already available from MFEM, new GPU-based examples
  - Gingko (sparse solvers for HPC systems) – interfaces to batched iterative methods in development
  - PETSc – updating interfaces to SUNDIALS integrators from PETSc

Concluding Remarks

- Software libraries are needed to help simulations achieve predictive capabilities
  - Complexity of many realistic models makes it essential for most work groups to adopt expertise from outside
  - Libraries can ease the transition to much more sophisticated and efficient methods and implementations
- Time integration falls in the middle of the scientific software stack – used by applications and uses solvers
  - Packages must be able to work with other packages, whether they be applications, discretizations, or solvers
  - Making as few assumptions as possible about data allows for a lot of flexibility but restricts scope
- Software practicalities are important considerations when developing new methods
  - Flexible software designs make incorporating more methods feasible
  - Complexity of implementations impacts maintenance efforts
- Community policies and regular testing help to ensure package compatibilities



Flexibility is essential!

Where to learn more and get the software

- Visit the SUNDIALS website
  https://computing.llnl.gov/projects/sundials
- SUNDIALS tutorial (from 2020): At top of presentations list at the SUNDIALS publications page:
  https://computing.llnl.gov/projects/sundials/publications
- Download from the SUNDIALS website:
  https://computing.llnl.gov/projects/sundials/sundials-software
- Download the tarball from the SUNDIALS GitHub page:
  https://github.com/LLNL/sundials/releases
- Install SUNDIALS using Spack "spack install sundials"
- View online documentation on all SUNDIALS packages at readthedocs.org:
  https://sundials.readthedocs.io/en/latest/
- Visit the xSDK website:
  https://xsdk.info
- xSDK Community Policies:
  https://github.com/xsdk-project/xsdk-community-policies
- xSDK examples github repository:
  https://github.com/xsdk-project/xsdk-examples
- xSDK Gitlab CI (pipeline) infrastructure:
  https://gitlab.com/xsdk-project/spack-xsdk/-/pipelines
- Install xSDK using Spack "spack install xsdk"
- Spack package manager Github:
  https://spack.io/
- Spack tutorial on readthedocs.org:
  https://spack-tutorial.readthedocs.io/en/latest/

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.

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

computing.llnl.gov/sundials

CASC

Center for Applied
Scientific Computing

Lawrence Livermore
National Laboratory

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.