Designing Integrators for User Flexibility: Interface Design in the SUNDIALS Suite of Nonlinear and Differential/Algebraic Solvers

Integrating the Integrators Workshop, Banff, Canada

CASC
Center for Applied Scientific Computing

Dec. 3, 2018

Carol S. Woodward, Daniel R. Reynolds, David J. Gardner, Cody J. Balos, and Slaven Peles

LLNL-PRES-763462

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

Lawrence Livermore National Laboratory

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.

LLNL: IBM/Nvidia P9/Volta, 2018

LBNL: Cray/Intel Xeon/KNL, 2016

ORNL: IBM/NVidia P9/Volta, 2018

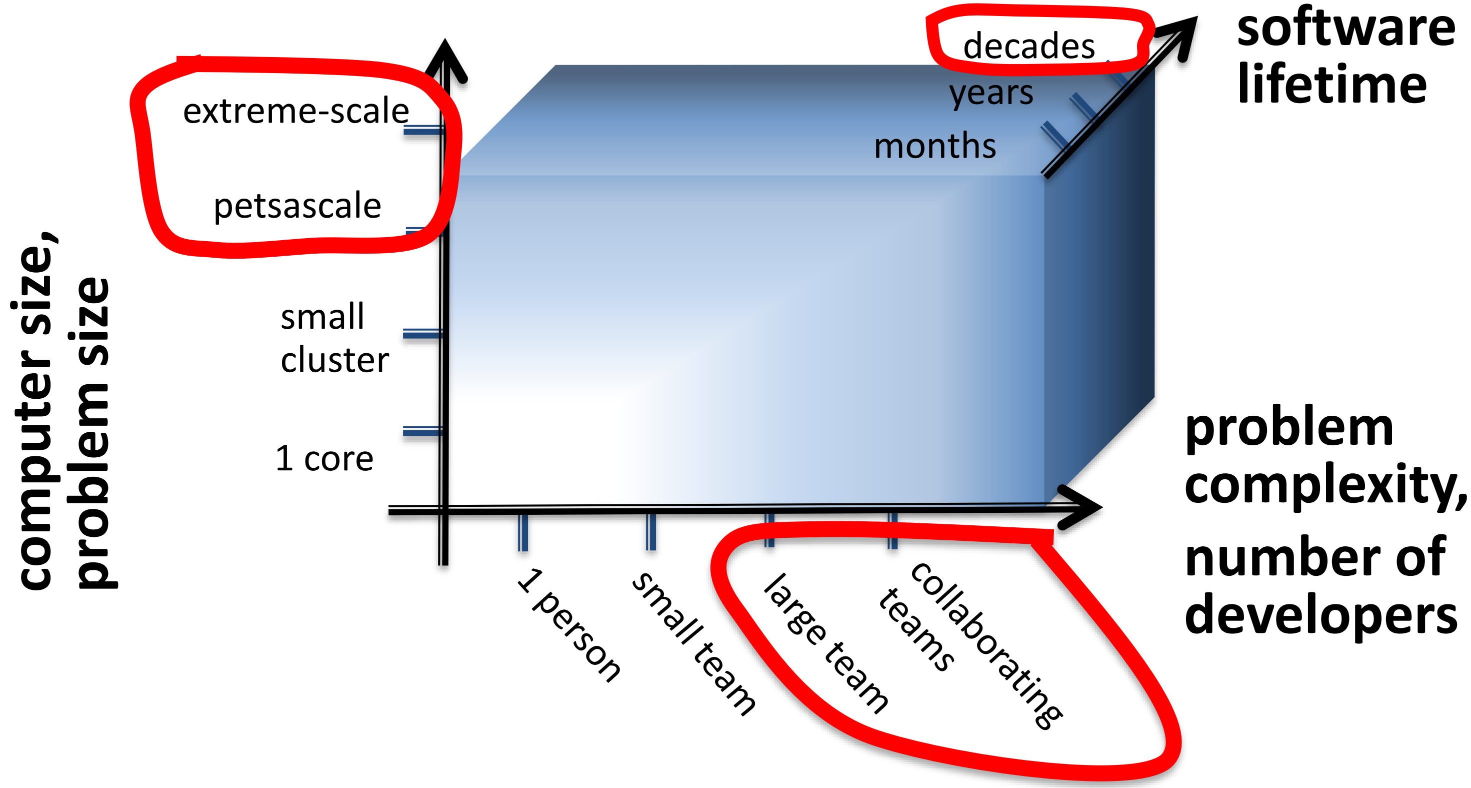
New capabilities will enable new computational science opportunities

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
- 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!

Slide courtesy of L. McInnes (ANL)

Despite challenges, opportunities abound for CSE software development improvements

- Better design, software practices, and tools are available
- Better software architectures: toolkits, libraries, frameworks
- Open-source software, community collaboration

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

Slide 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

- **References:**
  - https://en.wikipedia.org/wiki/Library_(computing)
  - What are Interoperable Software Libraries? Introducing the xSDK

Slide courtesy of L. McInnes (ANL)

Why is reusable scientific software important for you?

User perspective:

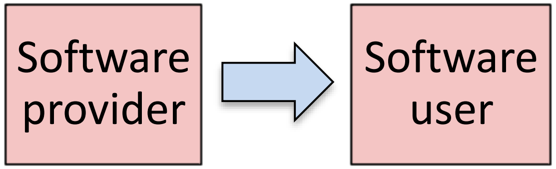
Focus on primary interests

- Reuse algorithms and data structures developed by experts
- Customize and extend to exploit application-specific knowledge
- Cope with complexity and changes over time

Provider perspective:

Share your capabilities

- Broader impact of your work
- Motivate new directions of research



- **More efficient, robust, reliable, sustainable software**
- **Improve developer productivity**
- **Better science**

We are developing the SUNDIALS SUite of Nonlinear and DIfferential-ALgebraic Solvers

- SUNDIALS is a software library consisting of ODE and DAE integrators and nonlinear solvers
  - 6 packages: CVODE(S), IDA(S), ARKode, and KINSOL
- Written in C with interfaces to Fortran
- Designed to be incorporated into existing codes
- Data use is fully encapsulated into vectors (and optionally matrices) which can be user-supplied
- Freely available released with BSD license (>17,000 downloads in 2017)
- Active user community supported by sundials-users email list
- Detailed user manuals are included with each package

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

CVODE(S) and IDA(S) employ variable order and step BDF methods for integration

- CVODE solves ODEs, $\dot{y} = f(t, y)$
- IDA solves $F(t, y, \dot{y}) = 0$
  - Targets: implicit ODEs, index-1 DAEs, and Hessenberg index-2 DAEs
  - Optional routine solves for consistent values of y_0 and $\dot{y}_0$ for some cases
- Variable order and variable step size Linear Multistep Methods

$$\sum_{j=0}^{K_1} \alpha_{n,j} y_{n-j} + \Delta t_n \sum_{j=0}^{K_2} \beta_{n,j} \dot{y}_{n-j} = 0$$

- Both packages include stiff BDF; $K_1 = k$, $K_2 = 0$, $k = 1,\ldots,5$
- CVODE includes nonstiff: Adams-Moulton; $K_1 = 1$, $K_2 = k$, $k = 1,\ldots,12$
- CVODES and IDAS include both forward and adjoint (user supplies the adjoint operator) sensitivity analysis

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

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

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

ARKode is the newest package in SUNDIALS

- Multistage embedded methods (as opposed to multistep):
  - High order without solution history (enables spatial adaptivity)
  - Sharp estimates of solution error even for stiff problems
  - But, implicit and additive multistage methods require multiple implicit solves per step
- ARKODE supports up to two split components with explicit and implicit methods $$M\dot{y} = f_E(t,y) + f_I(t,y)$$
- Split system into stiff, f_I, and nonstiff, f_E, components
- ARKODE includes the capability for multirate integration, currently two-rate explicit/explicit (more to come very soon)
- M may be the identity or any nonsingular mass matrix (e.g. FEM)

See presentation by Dan Reynolds on Tuesday

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

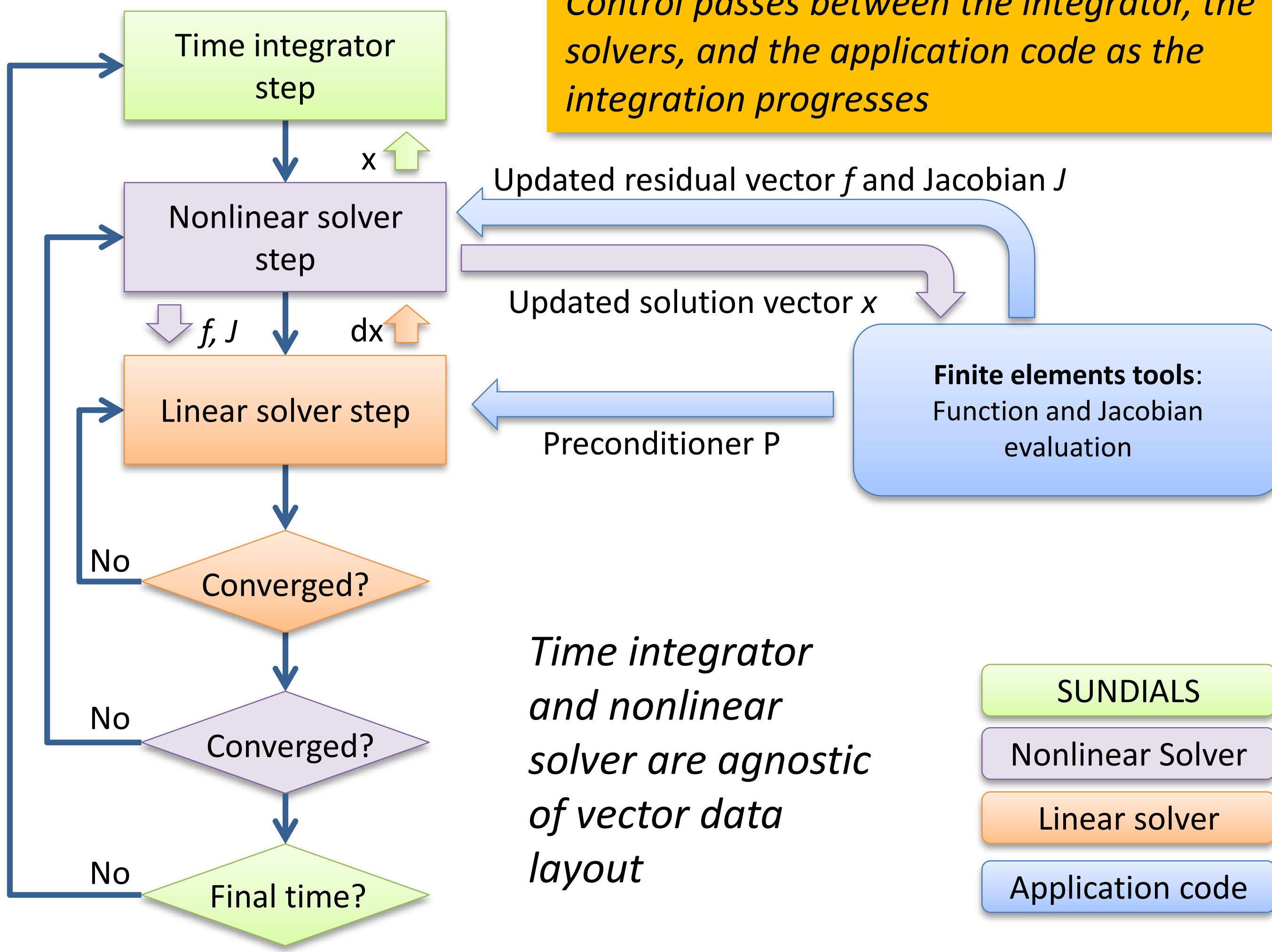
- Nonlinear solvers can be made more efficient through control of properties of the subsidiary linear solver, but these properties can be encapsulated

- 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

SUNDIALS uses Control Inversion to interoperate with other solvers and applications

Use case: implicit integration with iterative linear solver and finite element (FEM) application

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



Numerical integrators and nonlinear solvers may invoke fairly complex step size control logic

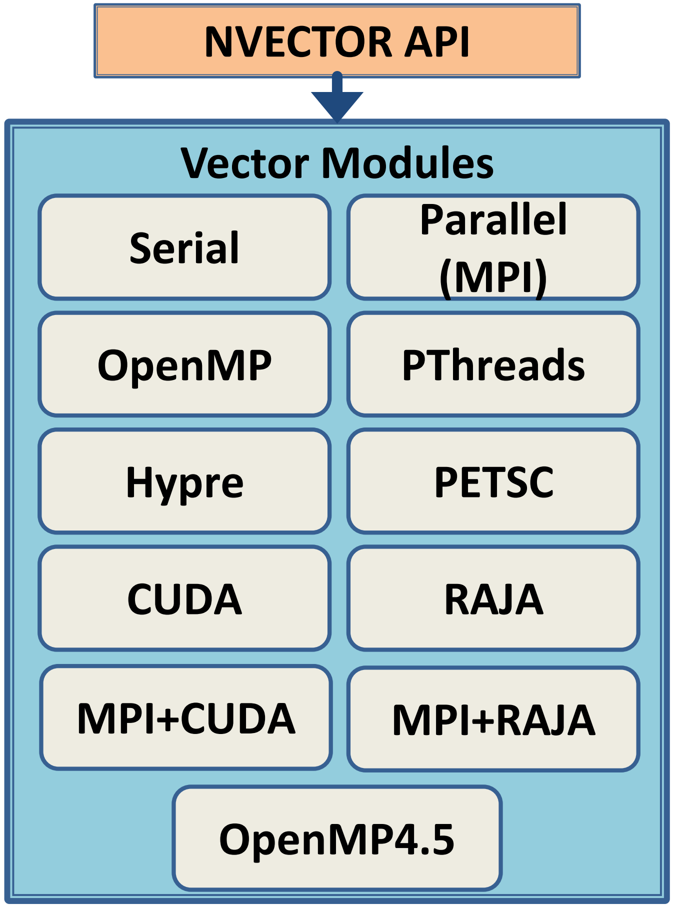
Time integrator and nonlinear solver are agnostic of vector data layout

In developing SUNDIALS we adhered to basic guiding principles in setting up interfaces between integrators and solvers and between the packages and the user

- 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, 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
  - 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

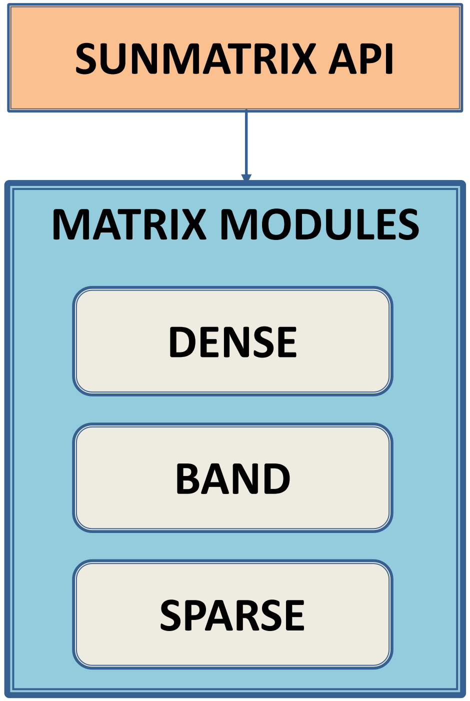
The SUNDIALS vector interface encapsulates interaction with application data

- Content is the vector data and information on its layout – depends on parallelism
- Ops includes
  - 3 constructors/destructors
  - 3 utility functions
  - 9 streaming operators (adding vectors, scaling, …)
  - 10 reduction operators (norms, dot products, etc.)
  - Several optional operators for efficiency
- Parallelism is reflected in the vector structure, not in SUNDIALS
- All ops are like level-1 BLAS operators
- Individual SUNDIALS packages require subsets of these



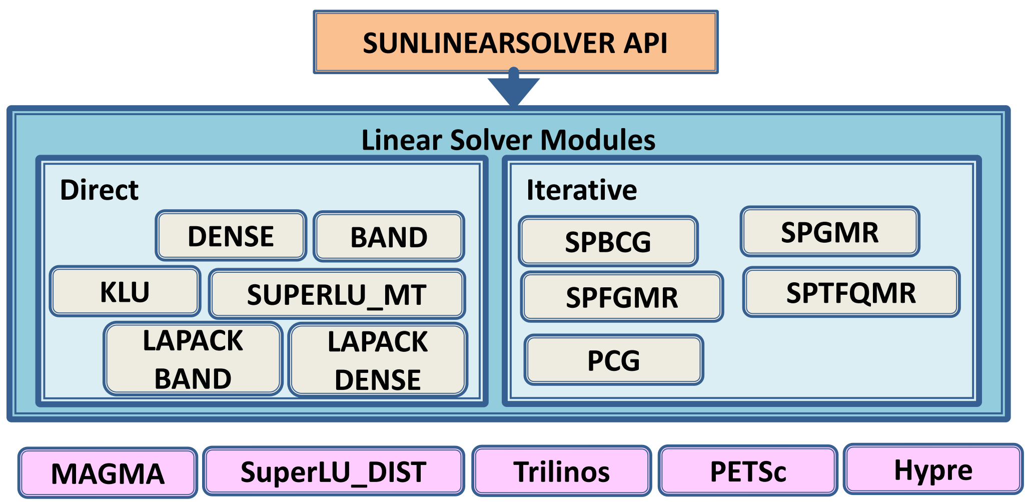
The SUNDIALS matrix interface encapsulates interaction with optionally used linear system matrices

- Content is the matrix data and information on its layout – depends on parallelism
- Ops includes
  - Constructor / Destructor
  - Scale
  - Copy
  - Add Identity
  - MatVec (when using a mass matrix)
- Matrices are needed with a direct linear solve is used
- Individual modules require subsets of these



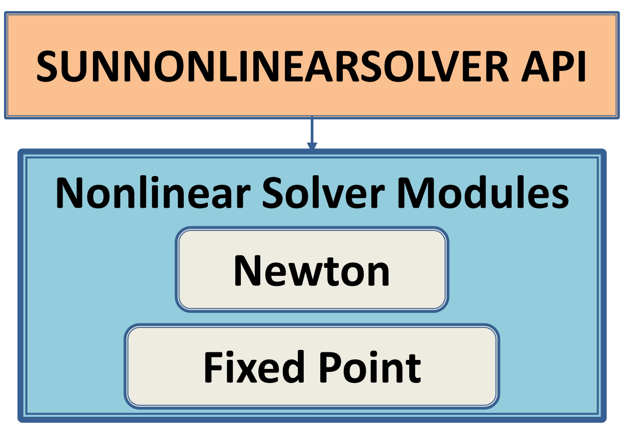
SUNDIALS time integration packages and nonlinear solvers are written in terms of generic linear solver operations

- Content is the solver data (iteration counters, needed work space, ptr to MatVec)
- Ops includes
  - Constructor / Destructor
  - Type identifier (direct, iterative, matrix-iterative)
  - Solve function
- Parallelism is reflected in
  - Vectors
  - How the matrix is handled
- Optional operations with linear solvers:
  - Solver setup
  - Set scaling
  - Preconditioner Set/Solve
  - Numerous iteration and solver statistics “Get” routines



SUNDIALS time integration packages are written in terms of generic nonlinear solver operations

- Content is the solver data (counters, update vectors, ptr to residual fcn)
- Ops includes
  - Constructor / Destructor
  - Type id (root-finding, fixed point)
  - Solve function
- Using the application-supplied problem-defining functions (f and optionally J), SUNDIALS packages form the nonlinear iteration function



- Optional operations:
  - Wrap linear solver setup and solve
  - User provided convergence tests
  - Numerous iteration and solver statistics “Get” routines

- When used with a direct linear solver, the SUNDIALS Newton solver holds the Jacobian matrix constant over many iterations resulting in Modified Newton
- With a matrix-free iterative linear solver, the iteration is an Inexact Newton method
- The SUNDIALS fixed point has an optional Anderson acceleration capability

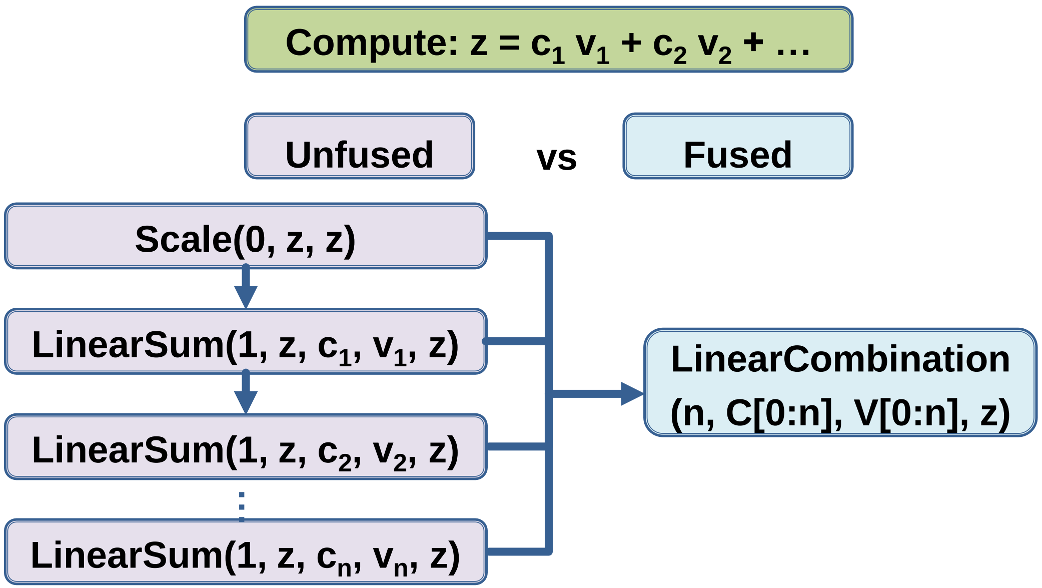
SUNDIALS package use: first instantiate the subsidiary structures and solvers then pass to the integrator

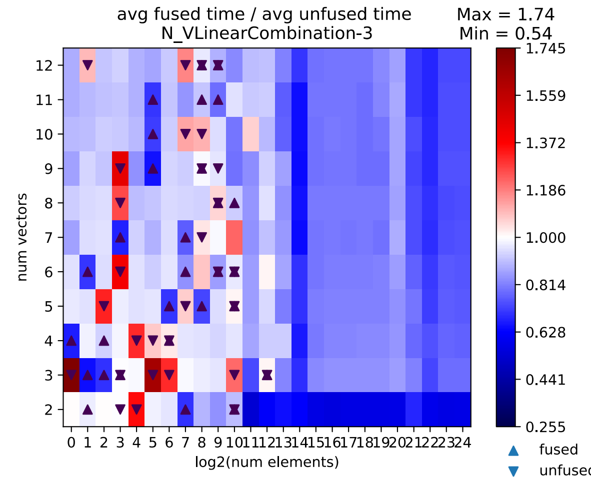
- Initialize parallelism if needed and Construct the initial state vector
- Call a Create function for the integrator – instantiates the integrator
- Call an Init function – specifies the problem (requires f fcn ptr) and initial state
- Set integration tolerances
- Create a matrix object if needed
- Create linear solver 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, the nonlinear solver, and the linear solver

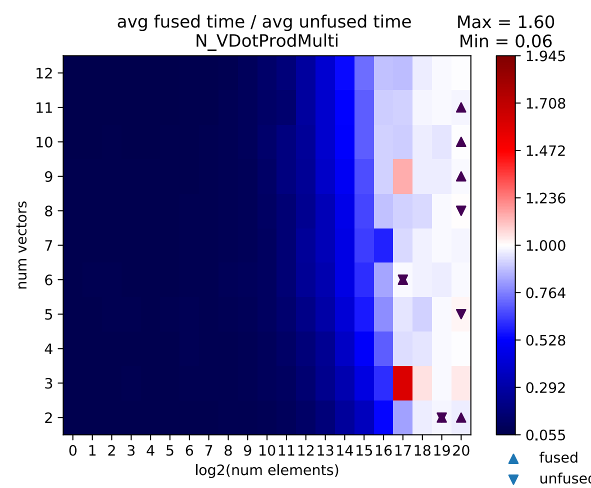
We are leveraging these interfaces to develop and deploy SUNDIALS for exascale systems

- We are adding new vectors that make efficient use of new architectures
  - OpenMP 4.5 + MPI
  - Kokkos
- Nine new vector kernels that rely on fused operations for decreased kernel calls and reduced communications
- We are adding interfaces to libraries that are optimized for fast linear solves on new architectures: SuperLU_DIST, PETSc, cuSOLVE, Trilinos, hypre
- Multiphysics simulations – we are developing and adding multirate methods to SUNDIALS
- Parallel bottleneck due to sequential time stepping – interfacing SUNDIALS integrator technology with multigrid reduction in time methods through the LLNL xBRAID software package

We added optional fused vector operations to the SUNDIALS vector API







- 9 new vector operations
- Greatest benefits when using long vectors and when fusing results in combined communication in parallel

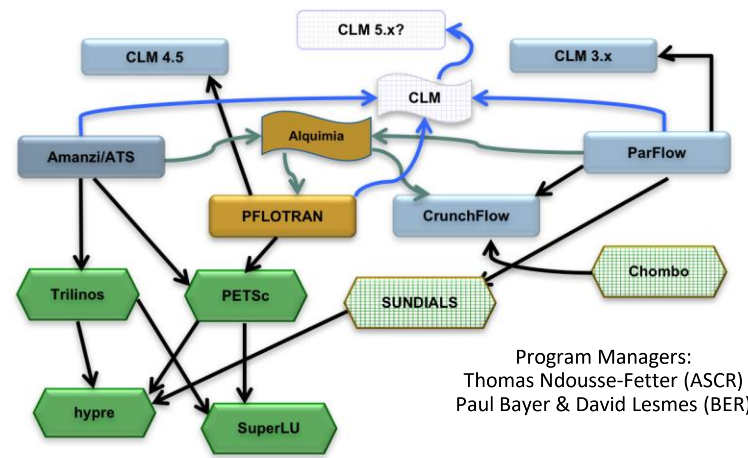
Software libraries are not enough: the xSDK effort was started to address challenges with using multiple libraries at once

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



Program Managers:
Thomas Ndousse-Fetter (ASCR)
Paul Bayer & David Lesmes (BER)

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

xSDK community policies: Help address challenges in interoperability and sustainability of software developed by diverse groups at different institutions

M1. Support xSDK community GNU Autoconf or CMake options.

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. The package must support production-quality installation compatible with the xSDK install tool and xSDK metapackage.

xSDK *recommended* community policies

Also **recommended policies**, which currently are encouraged but not required:

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.

https://xsdk.info

https://xsdk.info/policies

Through the Exascale Computing Project, the xSDK is also facilitating greater interoperability between member packages

- PETSc:
  - hypre, SuperLU, Trilinos linear solvers
  - SUNDIALS time integrators
- Trilinos: hypre, SuperLU, PETSc linear solvers
- Hypre:
  - SuperLU for coarse grid solves
  - Planned: interoperability with PETSc and Trilinos matrix structures

- SUNDIALS:
  - SuperLU_MT
  - Planned: Trilinos, hypre, PETSc, MAGMA, and more SuperLU linear solvers
- MFEM:
  - PETSc solvers
  - SUNDIALS time integrators

Summary and Future Directions

- Exascale systems are posing significant challenges to the scientific computing community and to numerical libraries
- Numerical libraries provide application developers with state-of-the-art numerical capabilities and the opportunity to easily take advantage of new algorithms
- Many time integration methods are easy to encapsulate in very flexible software
- By using control inversion and careful encapsulation of functionality, SUNDIALS provides flexible interfaces to high performance solvers and data structures
- We have made some progress in responding to the exascale challenge for SUNDIALS
  - New vector kernels (CUDA, RAJA, OpenMP4.5); fused vector kernel API
  - Redesigned solver interfaces for better encapsulation
  - Multirate methods – *See talks by Dan Reynolds and John Loffeld*

SUNDIALS v4.0.0 coming out *this week*:
https://computation.llnl.gov/casc/sundials

Acknowledgements

Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the U.S. Department of Energy Office of Advanced Scientific Computing Research and National Nuclear Security Administration.

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations – the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative.

CASC

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory