The SUNDIALS Suite of Time Integrators and Nonlinear Solvers

Bout++ Workshop 2018

David J. Gardner, Carol S. Woodward, Daniel R. Reynolds,
Alan C. Hindmarsh, Slaven Peles, John Loffeld, and Cody Balos

LLNL-PRES-756675
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

- Overview of SUNDIALS packages
- Recent additions
  - CUDA and Raja vectors, fused vector operations
  - Revised linear solver API
- Current development
  - MPI+X and OpenMP 4.5 vectors
  - Nonlinear solver API
  - Multirate methods
- Future plans
  - Many-vector API
  - Parallel in Time

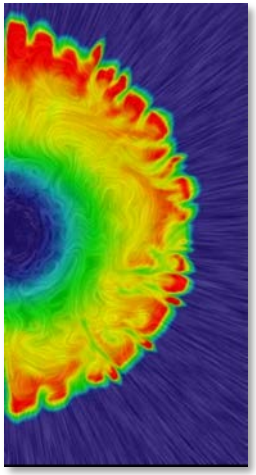
SUNDIALS: Suite of Nonlinear and Differential-Algebraic equation Solvers

- Suite of ODE and DAE time integrators and nonlinear solvers
  - Adaptive integrators with forward an adjoint sensitivity capabilities
  - Written in C with interfaces to Fortran
  - Designed to be incorporated into existing codes
- Modular implementation
  - Packages are built on shared vector, matrix, linear solver, preconditioner, and soon nonlinear solver APIs
  - Users can supply their own data structures/modules or use SUNDIALS provided versions
- Packages
  - **ODEs:** CVODE[1,3], CVODES[2], ARKode[1,3]
  - **DAEs:** IDA[1,3], IDAS[2]
  - **Nonlinear Systems:** KINSOL[3]

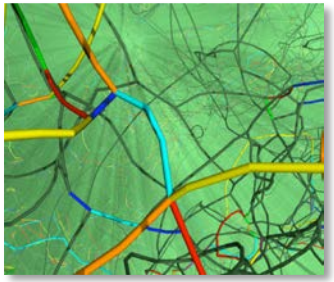
[1] Rootfinding capabilities
[2] Forward and adjoint sensitivity analysis capabilities
[3] Fortran interfaces

SUNDIALS: Suite of Nonlinear and Differential-Algebraic equation Solvers

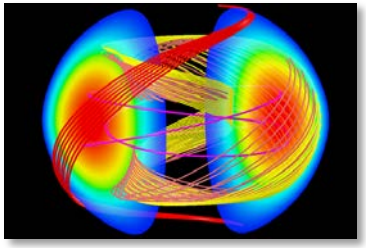
- Use and support
  - Used in application codes worldwide
  - More than 17,000 downloads in 2017
  - Extensive user documentation
  - User support email list, archive at nabble.com
- Freely available under BSD license
  - SUNDIALS webpage https://computation.llnl.gov/projects/sundials
  - GitHub https://github.com/LLNL/sundials
  - Spack https://github.com/spack/spack



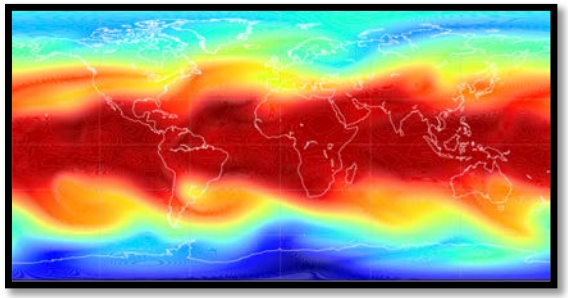
Core collapse supernovae



Dislocation Dynamics



Magnetic Reconnection



Atmospheric Dynamics

CVODE(S) and IDA(S): Adaptive step and order multistep integration methods

- CVODE solves $\dot{y} = f(t, y)$
- IDA solves $F(t, y, \dot{y}) = 0$
  - Targets: implicit ODEs, index-1 DAEs, and Hessenberg index-2 DAEs
  - Optional routines solve for consistent initial values for some cases
- Adaptive order and step linear multistep methods

$$\sum_{i=0}^{K_1} \alpha_{n,i} y^{n-i} + h_n \sum_{i=0}^{K_2} \beta_{n,i} \dot{y}^{n-i} = 0$$

- Both packages include BDF methods for stiff systems
- CVODE includes Adams-Moulton methods for non-stiff systems
- CVODES and IDAS include forward and adjoint sensitivity analysis capabilities (user supplies the adjoint operator)

ARKode: Adaptive step Runge-Kutta integration methods for ODEs

- ARKode solves $M\dot{y} = f^E(t,y) + f^I(t,y)$
  - M is any nonsingular linear operator (mass matrix, typically $M = I$)
  - $f^E(t,y)$ contains explicit terms and $f^I(t,y)$ implicit terms

- Adaptive multistage additive Runge-Kutta methods

$$Mz_i = My_n + h_n \sum_{j=1}^{i-1} A^E_{i,j} f^E(t^E_{n,j}, z_j) + h_n \sum_{j=1}^{i} A^I_{i,j} f^I(t^I_{n,j}, z_j), \quad i = 1, \ldots, s,$$

$$My_{n+1} = My_n + h_n \sum_{j=1}^{s} \left[b^E_j f^E(t^E_{n,j}, z_j) + b^I_j f^I(t^I_{n,j}, z_j) \right] \text{ (solution)}$$

$$M\tilde{y}_{n+1} = My_n + h_n \sum_{j=1}^{s} \left[\tilde{b}^E_j f^E(t^E_{n,j}, z_j) + \tilde{b}^I_j f^I(t^I_{n,j}, z_j) \right] \text{ (embedding)}$$

- ARKode includes adaptive explicit (ERK), implicit (DIRK), and IMEX (DIRK+ERK) methods, methods may also be user supplied

ARKode Flexibility Enhancements

- Variety of built-in Runge-Kutta methods; supports user-supplied
  - ERK order 3 – 8
  - DIRK order 2 – 5
  - ARK order 3 – 5
- Variety of built-in adaptivity functions; supports user-supplied
- Variety of built-in implicit predictor algorithms
- Ability to specify a problem is linearly implicit
- Ability to resize data structures based on changing problem size (e.g., adaptive mesh refinement)
- Set routines for internal package parameters

Time steps are selected to minimize local truncation error and maximize efficiency

- Time step selection
  - Based on the method, estimate the time step error
  - Accept the step if $\|LTE(h_n)\|_{\mathrm{wrms}} < 1$; otherwise reject the step

$$\|y\|_{\mathrm{wrms}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(w_i y_i)^2} \qquad w_i = \frac{1}{rtol|y_i| + atol_i}$$

  - Choose the next step h'_n so that $\|LTE(h'_n)\|_{\mathrm{wrms}} < 1$
  - CVODE(S) and IDA(S) will also choose the method order for the next step that has the largest step that meets the error condition

- Relative tolerance controls error relative to the size of the solution; rtol = 1e-4 means that errors are controlled to 0.01%

- Absolute tolerances control error when a solution may be small
  - e.g. solution starting at a nonzero value but decaying to a noise level, atol should bet set to the noise level

KINSOL: Solvers for systems of nonlinear algebraic equations

- Newton iterations for solving $F(u) = 0$
  - Computing iteration updates requires solving the linear system

$$J(u^k)\delta^k = -F(u^k) \qquad J(u) = \frac{\partial F(u)}{\partial u}$$

  - Direct solvers: modified Newton, Jacobian matrix is normally out of date
  - Iterative solvers: inexact Newton, linear system is solved approximately
- Dynamic linear tolerance selection with iterative linear solvers

$$\|F(u^k) + J(u^k)\delta^k\| \leq \eta^k \|F(u^k)\|$$

- Backtracking and line search options for robustness
- Can separately scale equations and/or unknowns

KINSOL: Solvers for systems of nonlinear algebraic equations

- KINSOL also includes Picard and fixed point iterations $G(u) = u$
- For the Picard iteration we consider $F(u) \equiv Lu - N(u)$

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

  this like Newton with L approximating J

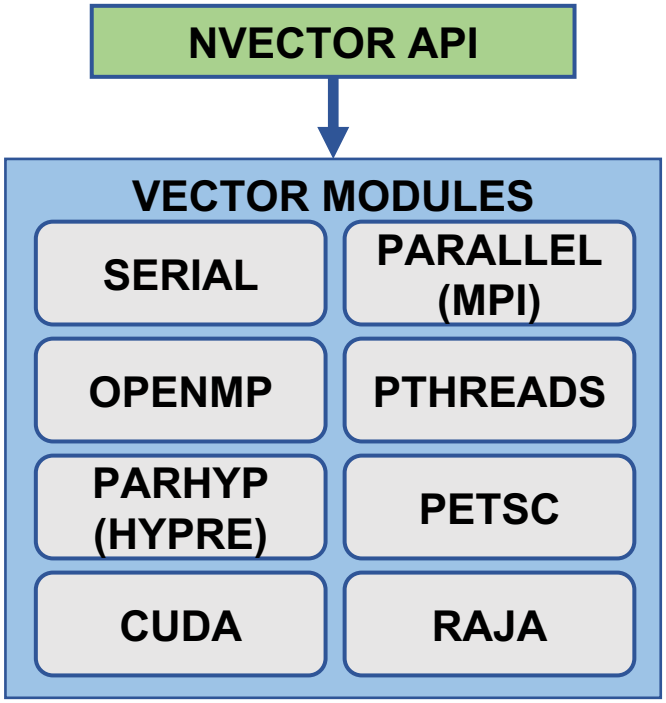
- Fixed point iteration does not require derivative information
- Both the Picard and fixed point iterations can utilized Anderson acceleration to increase the convergence rate
  - Requires the solution of a least squares problem the size of the number of prior residuals (not unknowns) each iteration
  - Least squares problem can be solved very efficiently

Flexible Design and Data Structures

- SUNDIALS packages are data structure agnostic
  - As long as basic vector kernels are supplied SUNDIALS packages will work with your native data structures
  - NVector API for vector operations
  - SUNMatrix API for matrix operations *(if needed)*
- SUNDIALS packages are built on shared solver APIs
  - SUNLinearSolver API for linear solvers
  - SUNNonlinearSolver API for nonlinear solvers *(in development)*
- Users can supply their own modules following the APIs or use implementations provided with SUNDIALS

NVector API

- A *content* structure specifies data and info to create new vectors
- An *operations* structure holds pointers to vector operations on the vector content data
- Routines to clone vectors for use within SUNDIALS
- All parallelism resides in vector operations: dot products, norms, etc.



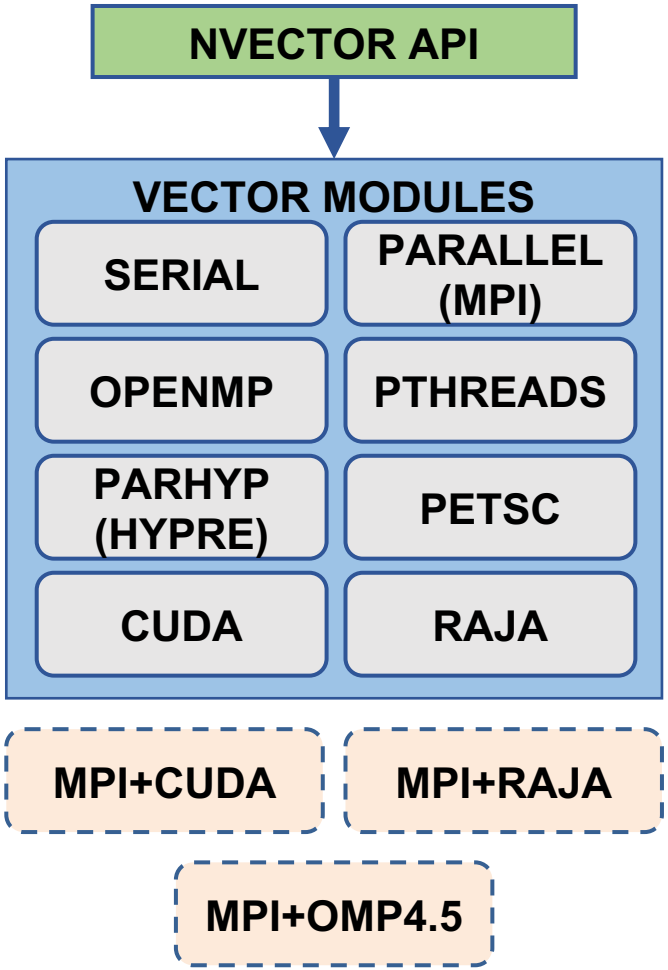
NVector modules provided with SUNDIALS

GPU vectors for a single GPU device

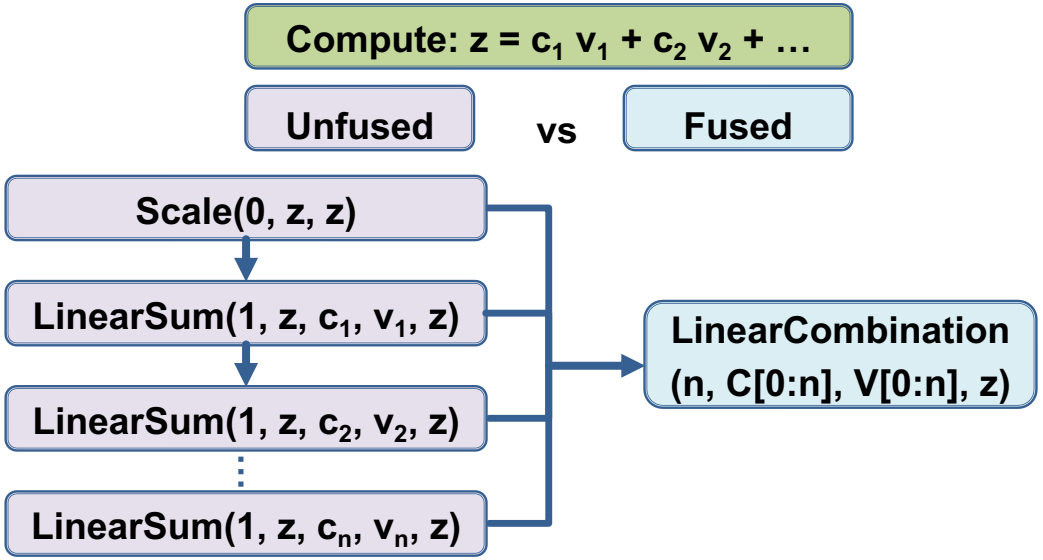
- NVector modules for use on GPU-based hardware
  - CUDA
  - RAJA hardware abstraction layer (https://software.llnl.gov/RAJA/)
  - OpenMP 4.5 *(currently testing)*
- Integrator logic is executed on the host; launches vector kernels on device
- Create the vector on the CPU and immediately move data to the GPU and leave there for duration of run
- Extra vector operations are provided to move data between GPU and CPU
- Results show best performance is achieved when model evaluation and solver workspace (vectors) are both in device memory during computation
- Users will need to write their problem-defining functions with CUDA, RAJA, or OpenMP 4.5+ to realize benefits

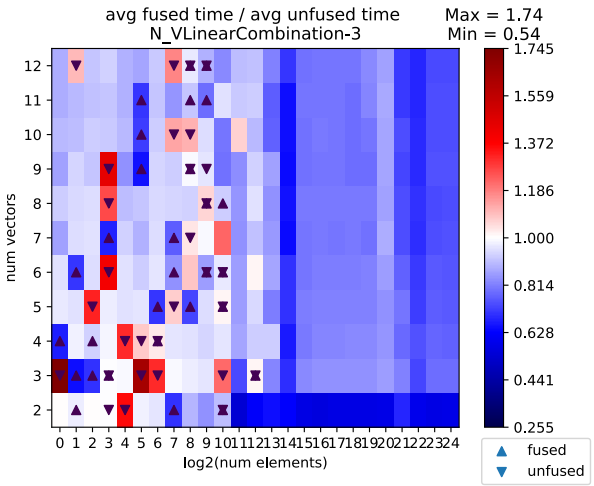
While these kernels are a start, we will need more sophisticated GPU-enabled vectors

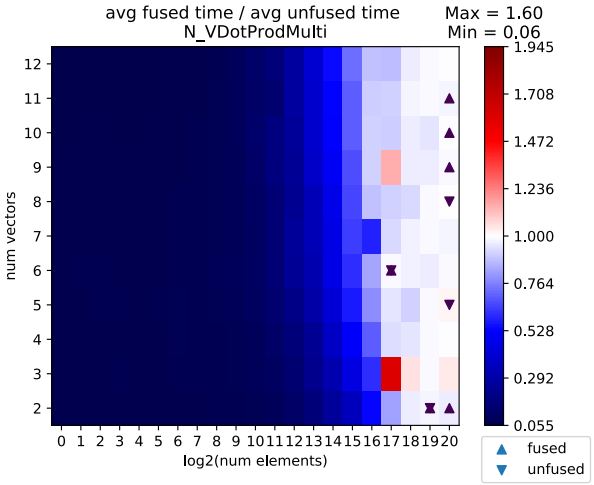
- Not reasonable to expect that all vector entries will fit in GPU memory at once
- MPI+X vectors coming soon
  - CUDA *(currently testing)*
  - RAJA *(currently testing)*
  - OpenMP 4.5 *(future development)*
- Will need to enrich our user interface to include information on data layout
- We are interfacing SUNDIALS with other LLNL codes (MFEM and *hypre*) and testing user API ideas



We added optional fused vector operations to the SUNDIALS vector API



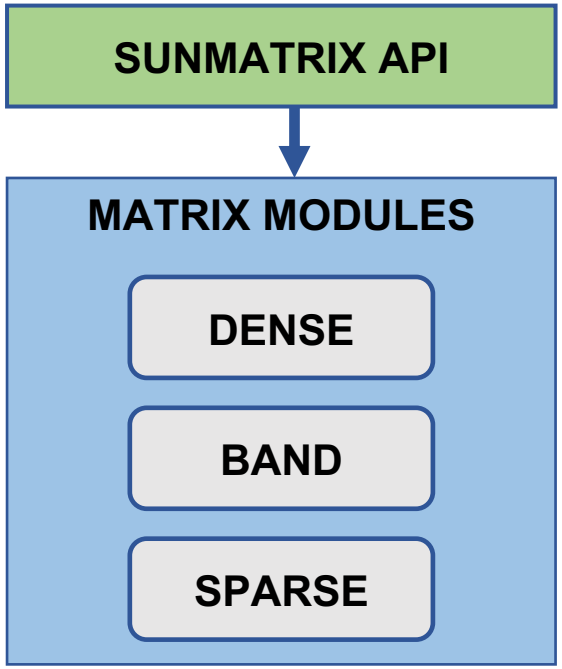




- Fused operations to increase data reuse and reduce communication
  - 9 new optional vector operations
- Greatest benefits when using long vectors and when fusing removes communication

SUNMatrix API

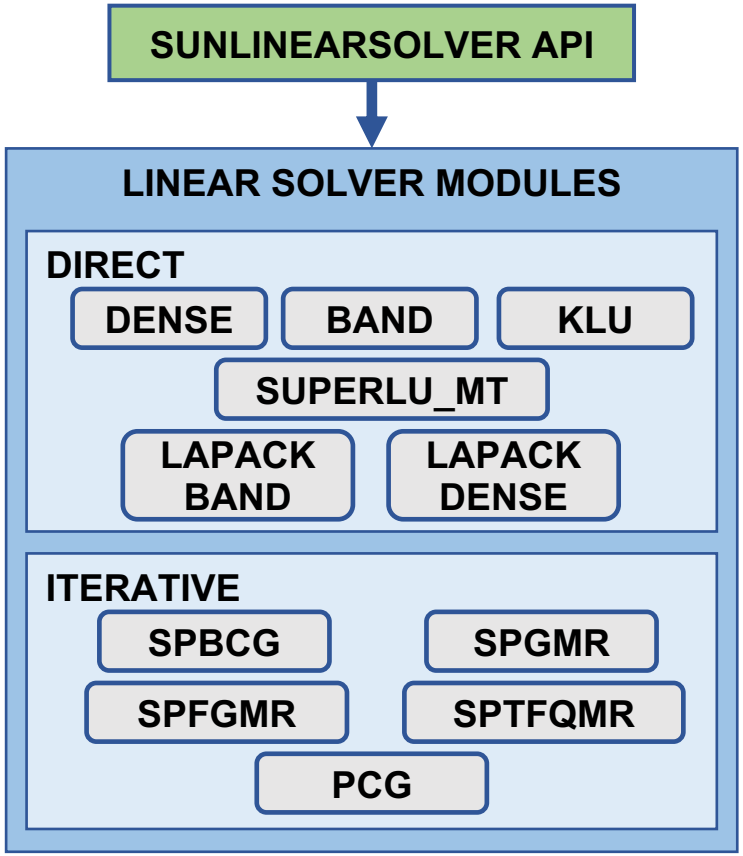
- Used with direct linear solvers
- Similar design to the NVector API
  - *content* structure specifies data and info to create new matrices
  - *operations* structure holds pointers to matrix operations on content data
- Matrix Operations
  - `GetID`
  - `Clone`
  - `Destroy`
  - `Zero`
  - `Copy`
  - `ScaleAdd`
  - `ScaleAddI`
  - `Matvec`
  - `Space`



SUNMatrix modules provided with SUNDIALS

SUNLinearSolver API

- Redesigned linear solver API for better encapsulation
  - Traded use of internal variables for accessor functions
  - Provided package-specific wrappers for user-defined functions to give uniformity in solver interfaces
- Direct and iterative linear solvers
- Similar design to the NVector API
  - *content* structure contains solver data
  - *operations* structure holds pointers to linear solver operations



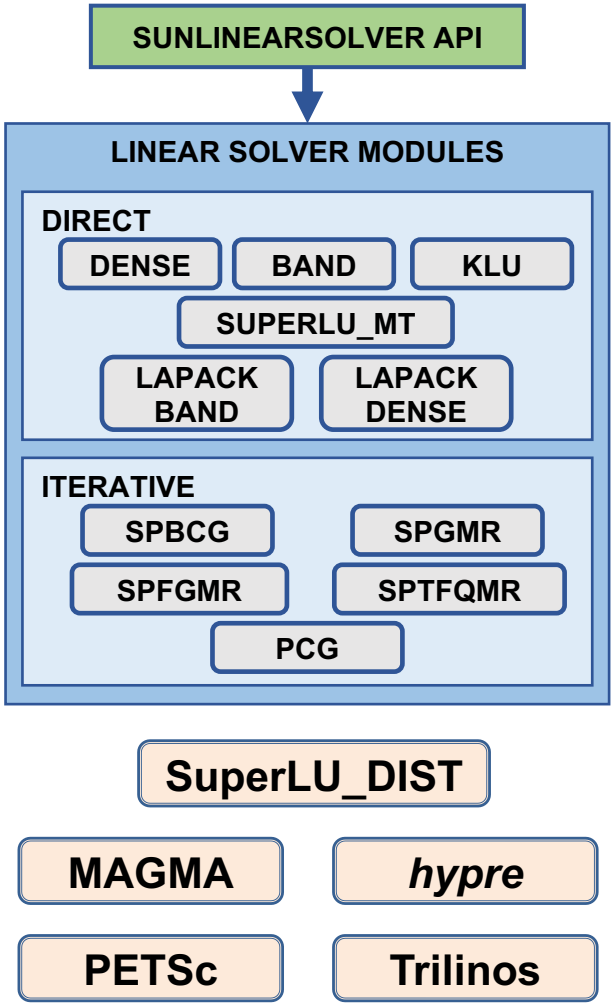
SUNLinearSolver modules provided with SUNDIALS

SUNLinearSolver Operations

- `GetType` – return linear solver type
- `Initialize` – initialize the linear solver object
- `Setup` – solver setup e.g., factorization or preconditioner update
- `Solve` – solve the linear system
- `Free` – deallocate solver memory
- `SetATimes` – set the Jacobian-vector product function (iterative)
- `SetPreconditioner` – set psetup and psolve functions (iterative)
- `SetScalingVectors` – set scaling vectors (iterative)
- `SolNumIters` – return the number of solver iterations (iterative)
- `SolResNorm` – return the residual norm from last solve (iterative)
- `SolResid` – return the initial residual vector (iterative)
- `LastFlag` – return last error flag encountered
- `Space` – return solver storage requirements

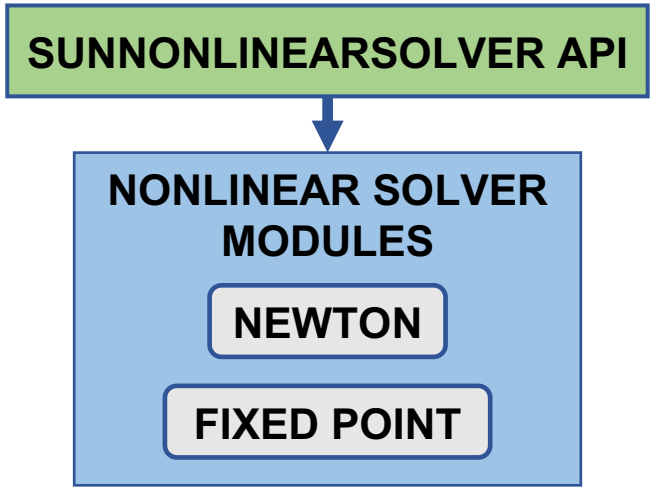
We are providing linear solvers on new architectures through external libraries

- New API eases interoperability with other packages
  - SuperLU_Dist linear solver and matrix modules are in development
  - Plan on adding additional modules in the future as part of the xSDK (https://xsdk.info/)
- Adding a matrix iterative linear solver type for iterative solvers that use matrix structures e.g., *hypre (in development)*
- One challenge is that SUNDIALS uses scaled iterative solvers – best use of libraries may be for preconditioners to the SUNDIALS scaled Krylov methods



SUNNonlinearSolver API

- Encapsulating nonlinear solvers in time integration packages *(in development)*
  - Uniformity in solver interfaces
  - Allow for external or user supplied nonlinear solvers
- Similar design to the NVector API
  - *content* structure contains solver data
  - *operations* structure holds pointers to nonlinear solver operations
- Newton will become the default nonlinear solver in CVODE(S)



SUNNonlinearSolver modules provided with SUNDIALS

SUNNonlinearSolver Operations

- `GetType` – return linear solver type
- `Initialize` – initialize the nonlinear solver object
- `Setup` – solver setup called infrequently
- `Solve` – solve the nonlinear system
- `Free` – deallocate solver memory
- `SetSysFn` – set the function defining the nonlinear system
- `SetLSetupFn` – set integrator linear solver setup function
- `SetLSolveFn` – set integrator linear solver solve function
- `SetConvTestFn` – set the convergence test function
- `GetNumIters` – return the number of iterations
- `GetCurIter` – return the current iteration number

We are investigating multirate integrators as an aid to applications coupling more physics

- Applications are becoming more multiphysics, multiscale, and multirate as they take advantage of greater computing power
- Multiphysics simulation challenges include:
  - Multirate processes, but too close to analytically reformulate
  - Optimal solvers may exist for some pieces, but not for the whole
  - Mixing of stiff and nonstiff processes
- Many commonly used schemes may suffer from low accuracy and poor/unknown stability.
- Multirate methods evolve distinct problem components with their own rate-specific time steps

Multirate integrators

- Existing multirate approaches:
  - O(h)-accurate subcycling
  - Fast/slow coupling via interpolation (typically $O(h^2)$, sometimes $O(h^3)$)
  - Extrapolation methods to "bootstrap" accuracy for low order methods
- New multirate methods should be:
  - High-order accuracy and stability, both within an between components
  - Flexible rate structure within integration
  - Robust temporal error estimation and step size adaptivity
  - Allow for problem-specific options (e.g., SSP or symplectic)
- Currently implementing 4th order two-rate explicit/explicit methods in ARKode
- Plan on adding two-rate IMEX and three-rate methods over the coming two years

Multirate Infinitesimal Step (MIS) Methods

- MIS/RFSMR is a highly efficient, up to O(h^3) method used in numerical weather prediction [Knoth and Wolke 1998, Schlegel et al. 2009]

$$y'(t) = f^{\{f\}}(t, y) + f^{\{s\}}(t, y)$$

- Fast terms $f^{\{f\}}(t, y)$ and slow terms $f^{\{s\}}(t, y)$ are separated by a factor m

- The MIS derivation assumes:
  - The slow component is integrated using an explicit "outer" Runge-Kutta method $T_o = \{A^o, b^o, c^o\}$ where $c_j^o \leq c_{j+1}^o$, $j = 1, \ldots, s^o - 1$
  - The fast component is advanced between slow stages as the exact solution of a modified ODE

- Practically, the fast solution is subcycled using an "inner" Runge-Kutta method (any type) with table T_i

MIS Properties

- MIS methods have a number of desirable properties
  - If both T_o and T_i are at least $O(h^2)$ then the MIS method is $O(h^2)$
  - If both T_o and T_i are at least $O(h^3)$ and To satisfies the condition

$$\sum_{j=2}^{s^o} \left(c_j^o - c_{j-1}^o\right) (e_j + e_{j-1})^\intercal A^o c^o + (1 - c_{s^o}^o) \left(\frac{1}{2} + e_{s^o}^\intercal A^o c^o\right) = \frac{1}{3}$$

    then the MIS method is $O(h^3)$.

- When T_i is a subcycled version of T_o the method is telescopic (may be used recursively to support n-rate problems)
- T_i and T_o can be problem-specific methods (SSP, symplectic, etc.)
- m can be varried between steps to adapt to changing rates
- Highly efficient – only a single traversal of the time interval is required to compute the solution

Relaxed Multirate Infinitesimal Step Methods

- RMIS [Sexton & Reynolds, 2018] is nearly identical to MIS, only changing how the fast stages contribute to y_{n+1}
- Inherits properties of MIS with minor changes:
  - The first stage of T_i must be explicit (the rest can be anything)
  - If both T_o and T_i are at least $O(h^3)$ then the RMIS method is $O(h^3)$
  - If T_i is at least $O(h^3)$, and if T_o is $O(h^4)$ and satisfies

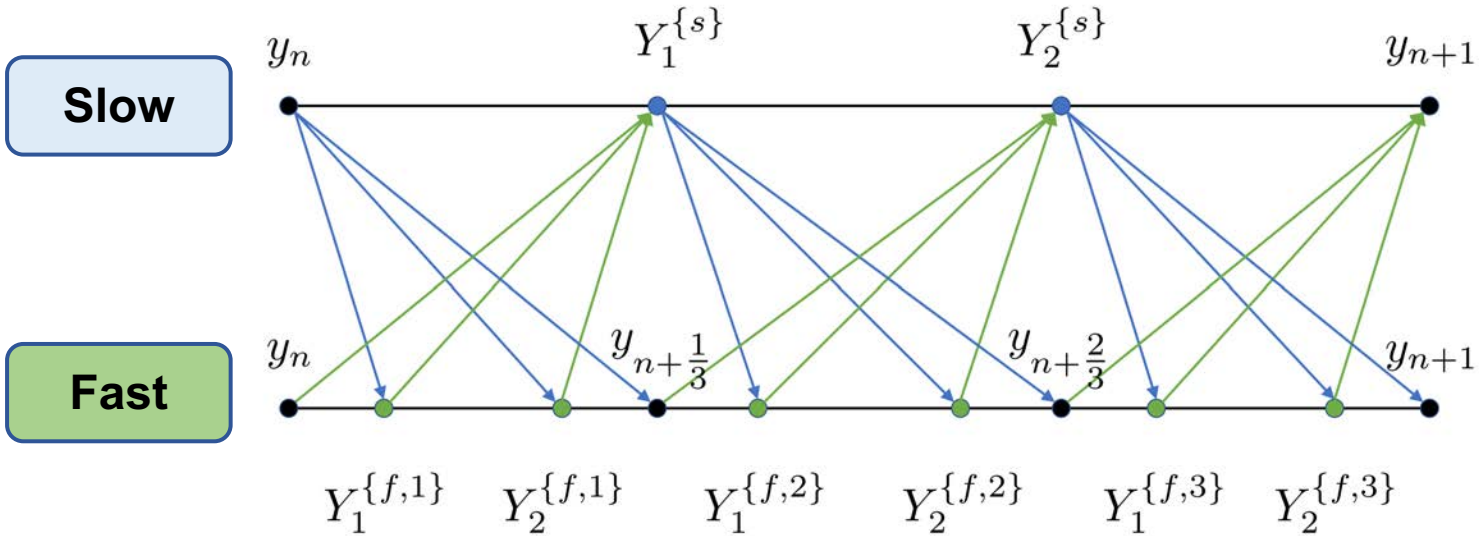
$$v^{o\intercal} A^o c^o = \frac{1}{12} \qquad v_j^o = \begin{cases} 0, & j = 1, \\ b_j^o \left(c_j^o - c_{j-1}^o\right) + \left(c_{j+1}^o - c_{j-1}^o\right) \sum_{k=j+1}^{s^o} b_k^o, & 1 < j < s^o, \\ b_{s^o}^o \left(c_{s^o}^o - c_{s^o-1}^o\right), & j = s^o, \end{cases}$$

  then the RMIS method is $O(h^4)$

- MIS can be used as an $O(h^3)$ embedding within $O(h^4)$ RIMS

RMIS Algorithm

- Illustration of the coupling between the slow and fast stages and steps in a third-order two-rate explicit RIMS method



- The most recent slow stage solution is held constant over the fast steps between the slow stages.

- Fast stage solutions are used to update the next slow stage value.

RMIS Algorithm

$$y'(t) = f^{\{f\}}(t, y) + f^{\{s\}}(t, y)$$

Set $z_1^{\{s\}} = y_n$.

For $j = 2, \ldots, s^o$:

Solve $v_j' = f^{\{f\}}(t, v_j) + \sum_{k=1}^{j-1} \frac{a_{j,k}^o - a_{j-1,k}^o}{c_j^o - c_{j-1}^o} \, f^{\{s\}}\Big(t_n + c_k^o h, z_k^{\{s\}}\Big)$,

where $t \in [t_n + c_{j-1}^o h, t_n + c_j^o h]$, with $v_j(t_n + c_{j-1}^o h) = z_{j-1}^{\{s\}}$.

Set $z_j^{\{s\}} = v_j(t_n + c_j^o h)$.

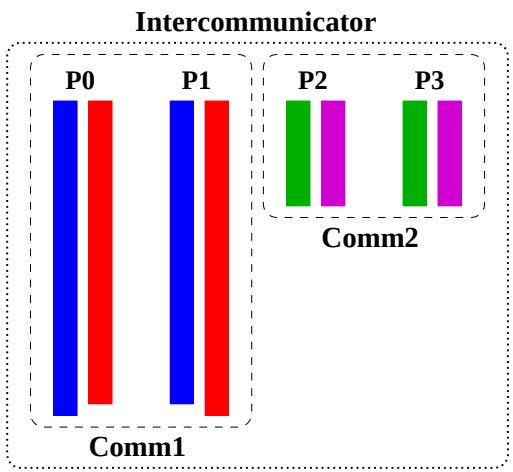
Set $y_{n+1} = y_n + h \sum_{k=1}^{s^o} b_k^o \left(f^{\{s\}}\Big(t_n + c_k^o h, z_k^{\{s\}}\Big) + f^{\{f\}}\Big(t_n + c_k^o h, z_k^{\{s\}}\Big) \right)$.

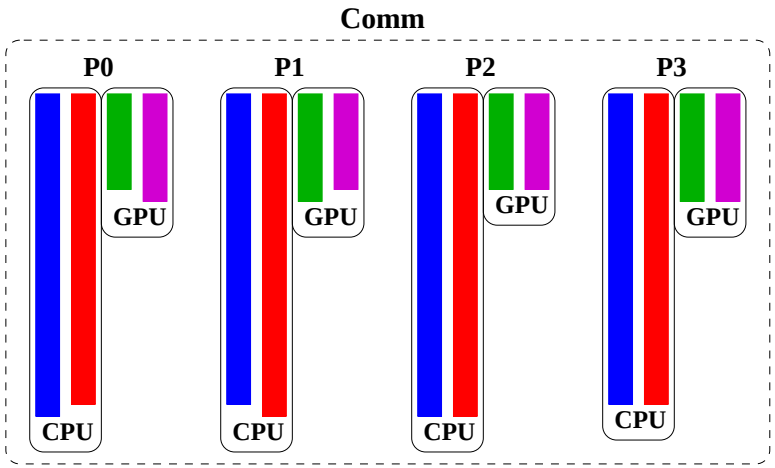
Multirate Summary

- Enhancements to ARKode to support multirate methods
- Addition of $O(h^3)$ and $O(h^4)$ multirate methods with a single traversal of the time interval
- Initial version will assume two rates, both evolved explicitly with a user defined step size (h)
- Follow on versions will support implicit methods for fast time scale, automated step size (h) and multirate (m) adaptivity
- Exploring extensions to three rates and possibly more rates

Many-vector API for multiphysics applications and heterogeneous architectures

- Enabling many-vector data structures more readily for users

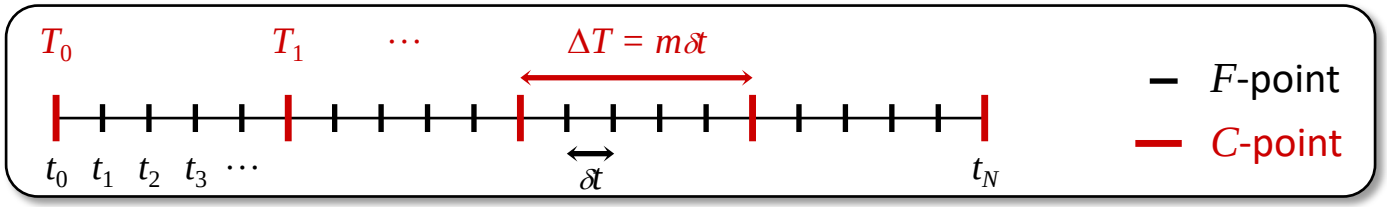




- Left: Process-based multi-physics decompositions
  - Comm1 connects processes 0 and 1 with physics operating on red/blue data,
  - Comm2 connects processes 2 and 3 with physics operating on green/magenta data,
  - MPI intercommunicator allows multiphysics coupling
- Right: multivector for multi-rate or data partitioning, allowing for each vector to utilize distinct processing elements

As total simulated time continues to grow, parallel in time methods may give speedup

- We are considering multigrid reduction in time (MGRIT) with SUNDIALS integration technologies
- Uses multiple coarse grids to update the original fine grid solution



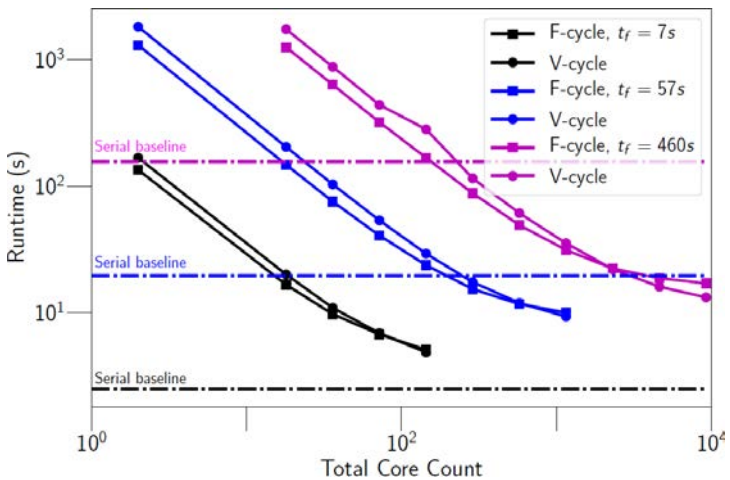
- The goal will be to enhance SUNDIALS with MGRIT capabilities for both BDF and RK packages
- Looking at using MGRIT in SUNDIALS by interfacing with the XBraid package which implements the MGRIT algorithm as a nonintrusive black box

"XBraid: Parallel multigrid in time" http://llnl.gov/casc/xbraid
R. Falgout *et al.*, "Parallel time integration with multigrid"

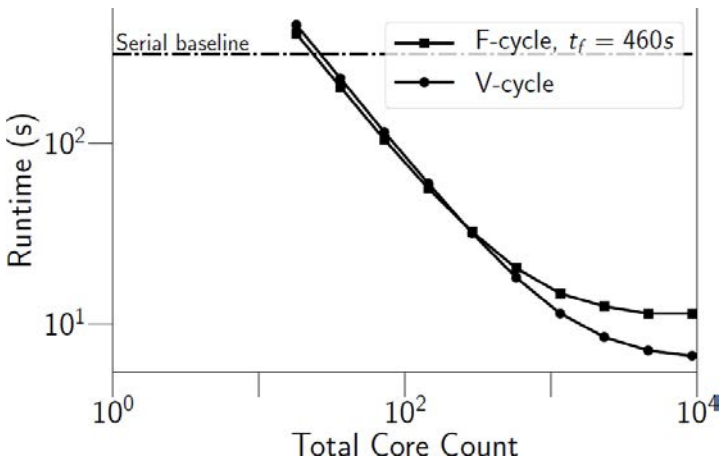
Results on a power grid test system based on the Western Interconnect show potential

- See speedups with both implicit Runge-Kutta and BDF methods
- Speedups can be attained with small core counts
- Don't see benefit with short time problems: Need enough steps for parallelization to be useful

Strong Scaling with BDF2



Strong Scaling with variable step RK4



IEEE 179 bus case
- 29 generators
- 793 unknowns
- DAE model
- Discontinuous change in load every second
- Coarsen order and grid

Courtesy of C. Woodward (LLNL), M. Lecouvez (CEA), J. Schroder (UNM), and R. Falgout (LLNL)

Summary

- Recent additions:
  - New GPU vectors (CUDA and RAJA); fused vector operations
  - Redesigned linear solver interfaces for better encapsulation and easier interfacing with external solver libraries
- Current work:
  - Nonlinear solver API to allow for external solver options
  - More vector options (OpenMP 4.5 and MPI+X)
  - Interfaces to external linear solvers and preconditioners
  - Multirate integration methods
- Future plans:
  - Enabling many-vector data structures more readily for users
  - Parallel-in-time methods with MGRIT and XBraid

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 (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

http://computation.llnl.gov/projects/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.