4-7-2021

Lecture 12: Recent Advances in Time Integration Methods and How They Can Enable Exascale Simulations

Carol S. Woodward
Lawrence Livermore National Laboratory

Recent Advances in Time Integration Methods and How They Can Enable Exascale Simulations

Carol S. Woodward

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

46th Annual Spring Lecture Series
April 7, 2021

Lawrence Livermore National Laboratory

CASC
Center for Applied Scientific Computing

LLNL-PRES-821168

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

Thank you Tulin and the Univ. of Arkansas SIAM Student Chapter!

- Great to see such an active SIAM Student chapter
- Participation in SIAM is important for our profession
  - Increases the visibility of applied math on campuses and in companies
  - Provides leadership opportunities
  - Provides a significant boost to building your network
- Student chapters provide an opportunity to build connections between math and other campus departments

siam® Society for Industrial and Applied Mathematics

https://kaman.uark.edu/siam/

- Great to also see an active AWM chapter
- Participation is open to all and helps support women in the mathematics profession through a greater understanding of the contributions of women in the mathematical sciences
- The chapter promotes mentoring and encouraging women and girls as they prepare for careers in the mathematical sciences

AWM ASSOCIATION FOR WOMEN IN MATHEMATICS

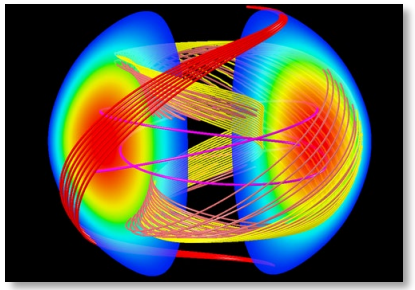
https://kaman.uark.edu/awm/

Most models of physical systems are formulated in terms of the *rate of change* of some variable

- **Ordinary Differential Equations** (ODEs)

$$\dot{y} = f(t, y), \quad y(t_0) = y_0$$

  - PDEs: Method of lines discretization *f* contains discrete spatial operations
  - Chemical reactions: *f* includes terms for each reaction $A + B \to P \quad \Rightarrow \quad \frac{\mathrm{d}[P]}{\mathrm{d}t} = k(T)[A][B]$

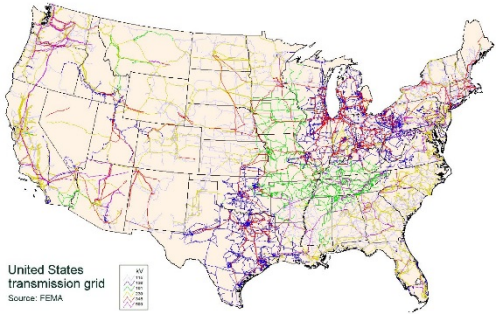


Magnetic reconnection

- **Differential Algebraic Equations** (DAEs)

$$F(t, y, \dot{y}) = 0, \quad y(t_0) = y_0, \quad \dot{y}(t_0) = \dot{y}_0$$

  - PDEs: Method of lines discretization with algebraic constraints
  - Power transmission models: *F* includes differential equations for power generators and network-based algebraic system constraining power flow
  - Electronic circuit models
  - If $\partial F/\partial y$ is invertible, we can solve for $\dot{y}$ to obtain an ODE, but this is not always the best approach, else the system is a DAE.

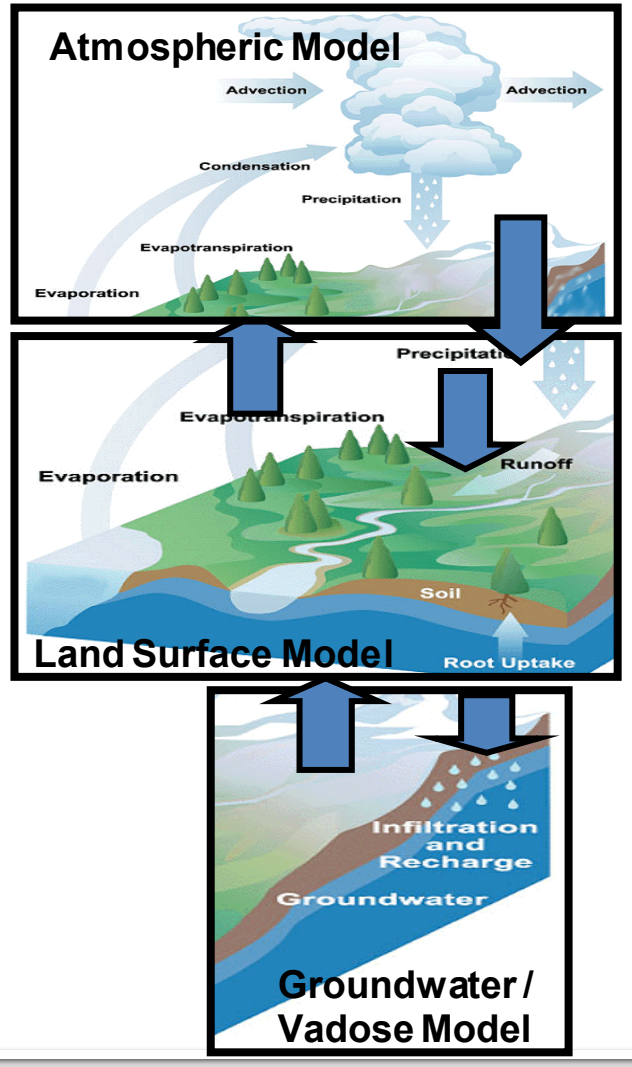


US Transmission grid
(Wikimedia Commons)

The compute power of exascale is bringing significant increases in what can be simulated

- More capacity can be used to run more refined simulations or to add more physics to a simulation
- Additional physics generally results in
  - Multicomponent
  - Multiphysics
  - Multiscale
  - Multirate

These new simulation goals translate into increased demand for high order and stable time integration methods and software for multiscale systems

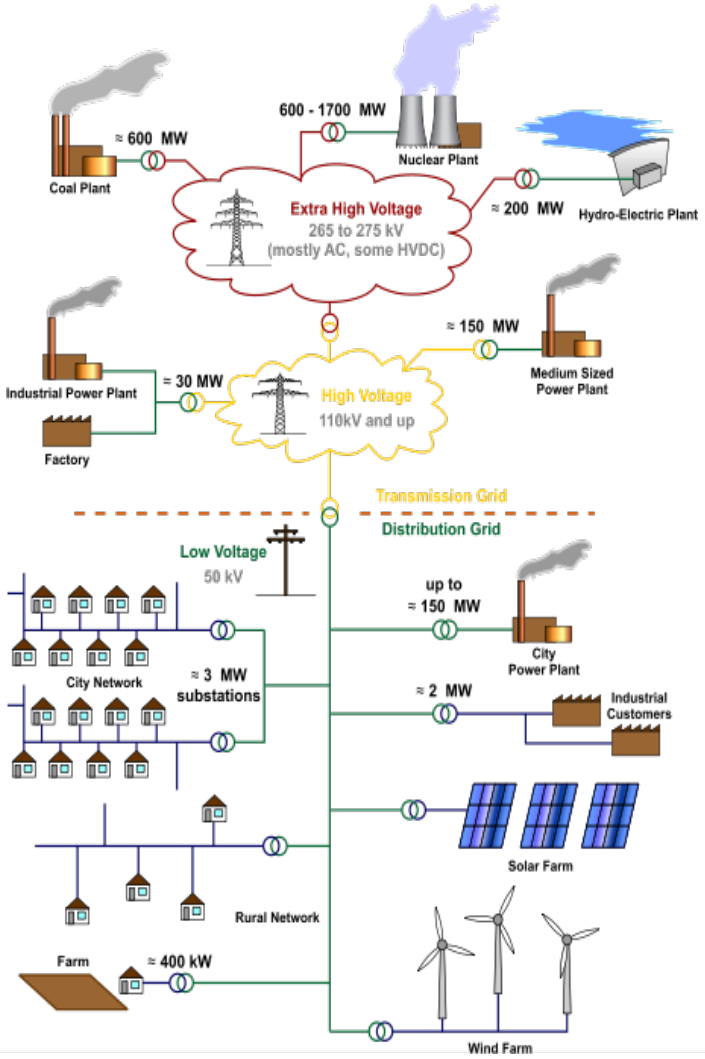


More effects are now considered when studying effects of climate change on **the water cycle**

Each model is typically run as its own component and loosely coupled to others

Atmospheric model time scale is very fast, subsurface model time scale is very slow

Power grid simulations are becoming more multiphysics as renewables are being introduced into the simulations



- Model flow
  - Transmisison - generators to substations
  - Distribution - substations to houses
  - Previously assumed distribution was instantaneous relative to transmission
- Solar and wind energy: generation is now in the distribution network
- Wind energy adds new time scales and weather models
- Solar energy is dependent on daylight and cloud cover
- These simulations are often wrapped in an optimization loop for contingency or market planning

A careful approach to time integration is important

- Scientists often split such simulations based on the scientific models, rather than timescales. Consider,

$$y'(t) = f_1(t, y) + \cdots + f_n(t_1 y), \qquad y\,(t_0) = y_0$$

- The most used splitting is "Lie-Trotter" splitting:

$$y_1{}'(t) = f_1(t, y_1), \qquad y_1(t_0) = y_0$$

$$y_2{}'(t) = f_2(t, y_2), \qquad y_2(t_0) = y_1(t_0 + h)$$

$$\vdots$$

$$y_m{}'(t) = f_m(t, y_m), \qquad y_m(t_0) = y_{m-1}(t_0 + h)$$

with solution, $y(t_0 + h) = y_m(t_0 + h)$

- Each partition of the operator may be integrated separately (or even subcycled), but the highest order of accuracy is 1, *even if each partition is integrated with a higher order method*

Splitting methods can suffer from low accuracy and stability

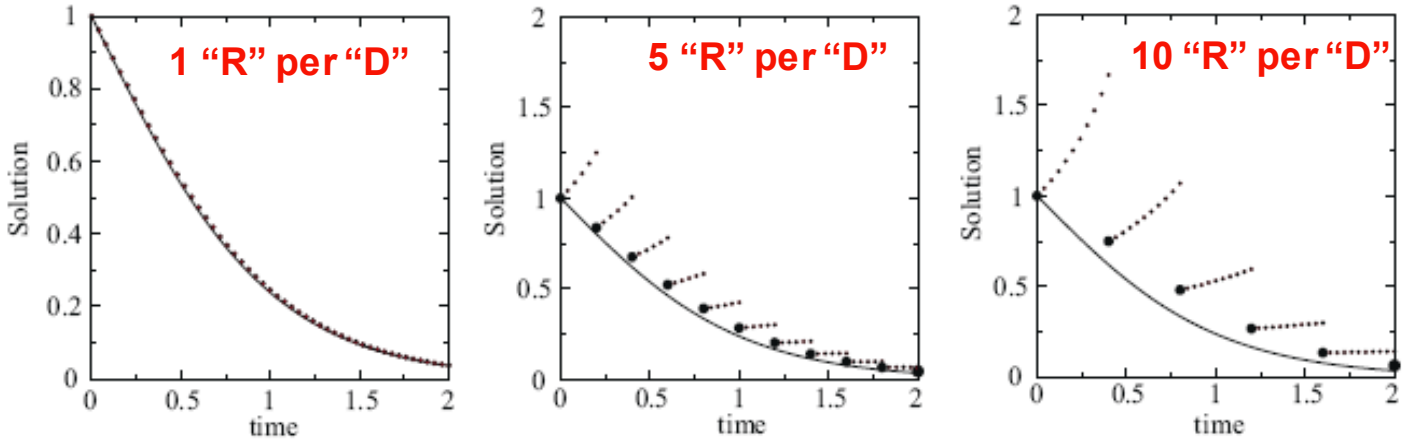
Example from *Estep et al.* (2008) $\dot{u} = -2u + u^2, u(0) = 1\ ,\ \ t > 0$

$$u(t) = \frac{u_0 e^{-\lambda t}}{1 + \frac{u_0}{\lambda}(e^{-\lambda t} - 1)}$$

For large time values, this goes to 0

Phase R, $\dot{u}_R = u_R^2$ subcycled inside phase D $\dot{u}_D = -\lambda u_D$

50 time steps, phase R subcycled inside phase D



In practice, when the "R" step generates an unphysical partial solution, this is often "limited" which lowers efficiency.

Traditional time integration methods have different approaches to achieving high order accuracy

Linear multistep (Adams and BDF) methods construct approximations based on prior states

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

- Retain a history of previous solutions
  - Adams-Bashforth explicit
  - Adams-Moulton implicit for nonstiff systems
  - BDF implicit for stiff systems
- Solve up to one nonlinear system per time step
- Amenable to problems with strong reaction and diffusion effects
- Stiff integrators often use a predictor-corrector scheme

If the system has widely varying time scales, and the phenomena that change on fast scales are stable, then the problem is stiff [Ascher & Petzold 1998]

Multistage (Runge-Kutta) methods construct approximations based on estimates of derivatives at multiple points within a single step

$$z_i = y_{n-1} + \Delta t_n \sum_{j=1}^{s} \left(a_{ij} f(t_{nj}, z_j) \right), \qquad i = 1, \ldots s$$

$$y_n = y_{n-1} + \Delta t_n \sum_{i=1}^{s} \left(b_i f(t_{ni} + c_i \Delta t_n), z_i \right)$$

$$t_{ni} = t_{n-1} + c_i \Delta t_n$$

- Use multiple internal stages per step
- More work per-step
- Can be explicit or implicit; Diagonally Implicit RK (DIRK) has $a_{ij} = 0$ for $j > i$
- Amenable to spatial adaptivity and hyperbolic effects
- The a's, b's, c's, and s define the method, its order of accuracy, and its stability

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

- Time step selection
  - Based on the method, estimate the time step error (embedded method of one lower order or direct error calculation)
  - Accept step if $||E(\Delta t)||_{WRMS} < 1$; Reject it otherwise

$$\|y\|_{\text{wrms}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(w_i\ y_i)^2} \qquad w_i = \frac{1}{RTOL|y_i| + ATOL_i}$$

  - Choose next step, $\Delta t'$, so that $||E(\Delta t')||_{WRMS}$ is expected to be small
- Some algorithms also allow order adaption: give the largest step expected to meet the error condition
- Advanced "error controllers" adapt these step sizes to meet a variety of objectives:
  - minimize failed steps
  - maximize step sizes
  - maintain smooth transitions in the step sizes as integration proceeds
- Temporal adaptivity can lead to much more efficient (and accurate) results

Extensions and variations on these traditional methods target high order and stable methods for multirate and multiscale systems

Goals for such methods include:

- Stability/accuracy for each component, as well as inter-physics couplings
- Custom/flexible time step sizes for distinct components
- Robust temporal error estimation and adaptivity of step size(s)
- Ability to apply optimally efficient and scalable solver algorithms on problem components rather than a one-size-fits-all solver for a monolithic system
- Support for experimentation and testing between methods

Common approaches include:

- Implicit/explicit (IMEX)
- Multirate
- Parallel-in-time

Implicit/explicit (IMEX) integrators try to match stiff integrators with stiff operators

- Consider the split problem,

$$y'(t) = f_{\mathrm{E}}(t,y) + f_I(t,y), \qquad t \in [t_0, t_f], \qquad y(t_0) = y_0$$

where $f_{\mathrm{E}}(t,y)$ contains the nonstiff terms and $f_{\mathrm{I}}(t,y)$ contains the stiff terms

- Variable step size additive Runge-Kutta (RK) methods combine explicit (ERK) and diagonally implicit (DIRK) methods to enable an IMEX solver [Ascher et al. 1997; Araujo et al. 1997; Kennedy & Carpenter 2003; . . .]
- Solve for each stage solution, z_i, sequentially then compute the time-evolved solution, y_n

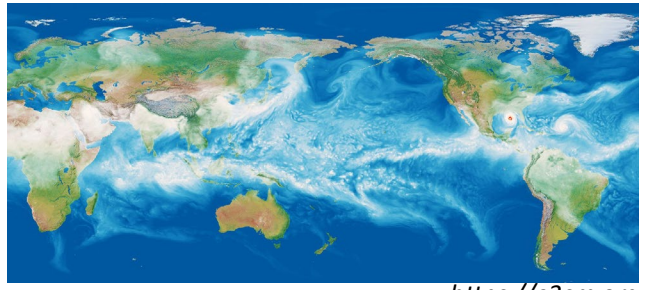
$$z_i = y_{n-1} + \Delta t_n \sum_{j=1}^{i-1} a_{i,j}^E f_E(t_{n,j}^E, z_j) + \Delta t_n \sum_{j=1}^{i} a_{i,j}^I f_I(t_{n,j}^I, z_j) \quad i = 1, \ldots, s,$$

$$y_n = y_{n-1} + \Delta t_n \sum_{i=1}^{s} b_i^E f_E(t_{n,i}^E, z_i) + b_i^I f_I(t_{n,i}^I, z_i),$$

- Coefficients must be chosen to satisfy "order conditions," constraints that, when satisfied, ensure the resulting method is of a specific order of accuracy
- Some methods allow an "embedding" of a method of 1 order lower than the target method computed with the same stage values and that can be used to estimate the error for use in adaptivity

IMEX methods are highly effective when a fast operator is resulting in a very restrictive CFL stability step size requirement

- These methods are popular in several fluids communities



https://e3sm.org

- One area of strong use is in the nonhydrostatic atmospheric dynamics community
  - Acoustic waves have a negligible effect on climate but travel much faster than convection (343 m/s vs 100 m/s horizontal and 1 m/s vertical), leading to overly restrictive explicit stability restrictions
  - Common to treat the gravity waves in the vertical implicitly and the rest of the system explicitly (Horizontally Explicit, Vertically Implicit (HEVI)) methods
  - For these models, parallel decomposition is only over the horizontal, so the implicit solve can be a sweep over independent columns contained on a processor with no communication required
  - Significant speedups have been observed for these methods over explicit [Vogl et al. 2020; Gardner et al. 2018; Giraldo 2013; Weller, Lock, and Wood 2013; …]

- Current work is in developing methods targeting larger stability regions in areas of need for particular applications
  - Low storage methods reduce the numbers of stages needed in memory at any time [Kennedy, Carpenter, and Lewis 2000]
  - Methods with large coverage of the imaginary axis for stability [Steyer et al. 2019]

Sequential time stepping is only one way of solving a large time-dependent system

The parareal method applies a two-level iteration [Lions, Maday, and Turninici, 2001].

Let: $\mathrm{G}(t_n, t_{n-1}, u_{n-1})$ and $\mathrm{F}(t_n, t_{n-1}, u_{n-1})$ be coarse and fine approximations, respectively, to $u(t_n)$ with initial condition $u(t_{n-1}) = u_{n-1}$

Parareal:

Starting with a coarse approximation U_n^0 at the time points $t_1, t_2, \ldots, t_N$, the parareal algorithm performs the correction at iteration k

$$U_{n+1}^{k+1} = F\left(t_{n+1}, t_n, U_n^k\right) + G\left(t_{n+1}, t_n, U_n^{k+1}\right) - G\left(t_{n+1}, t_n, U_n^k\right), \quad \text{for } k = 0, 1, 2, \ldots$$

The propagator, F, can be applied to all time intervals in parallel since it is initiated by the coarse time approximation done with G.

Let $\mathrm{F}(t_{n+1}, t_n, U_n^k)$ denote the exact solution at t_{n+1} and $\mathrm{G}(t_{n+1}, t_n, U_n^k)$ be a one step method with local truncation error bounded by $\mathrm{C}_1 \Delta T^{p+1}$. If

$$|G(t + \Delta T, t, x) - G(t + \Delta T, t, y)| \leq (1 + C_2 \Delta T)|x - y|,$$

then the solution from parareal is of order kp, where k is the iteration number, and p is the order of the coarse integrator, [Gander and Hairer, 2008].

As the method converges, the distance between $\mathrm{G}(t_{n+1}, t_n, U_n^k)$ and $\mathrm{G}(t_{n+1}, t_n, U_n^{k+1})$ gets smaller until just the result of the fine propagator is left.

From: Ong and Schroder, 2020

Parareal ideas can be combined with spectral deferred corrections to give a flexible multiphysics integrator as seen in PFASST

The Parallel Full Approximation Scheme in Space and Time (PFASST) algorithm [Emmett and Minion, 2012]:

- Uses a spectral deferred correction (SDC) algorithm as the coarse and fine integrators
- Extends Parareal to a nonlinear multilevel algorithm through the nonlinear multigrid Full Approximation Scheme
- In SDC, one uses a quadrature scheme to approximate a residual to drive a correction propagation
  - Requires a quadrature scheme
  - Accuracy increases with each iteration of the multilevel hierarchy, subject to accuracy of the quadrature
  - For multiphysics applications, inside integrator can use a first order splitting
  - Has a multirate extension

Has been used in reactive flow modeling to get to higher order without changing the underlying operator split methods [Minion et al., 2003; Emmett et al., 2014]

See Gander 2015 for review of PinT methods.

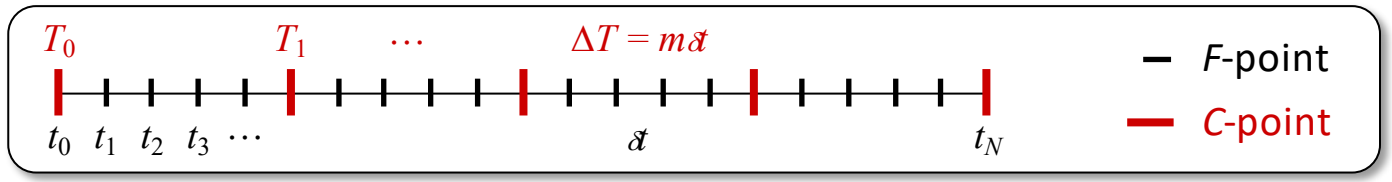
Multigrid reduction in time (MGRIT) is a parallel multigrid method applied in the time domain

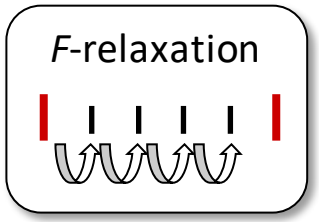
- For the ordinary differential equation $y'(t) = f\big(t, y(t)\big), \qquad y(0) = y_0$
- Consider the general one-step method $y_i = \Phi_i(y_{i-1}) + g_i, \qquad i = 1, \dots, N$
- For the linear case (for simplicity), time stepping is a block forward solve
- An $\mathcal{O}(N)$ direct method, but sequential

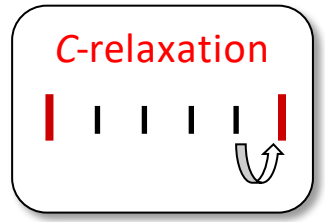
$$\begin{pmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_N \end{pmatrix}$$

- Replace the sequential solve with a multigrid reduction in time (MGRIT)
- Extend to nonlinear systems with the Full Approximation Scheme multigrid method
- An $\mathcal{O}(N)$ iterative method, but highly parallel

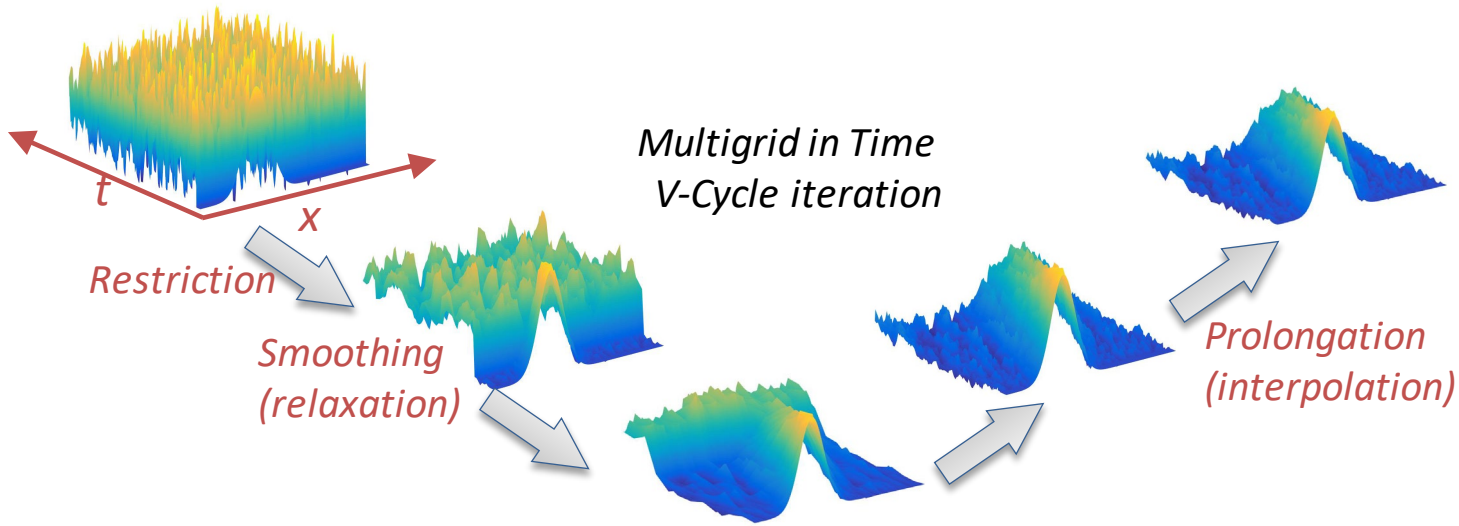
The two-grid MGRIT algorithm applies two types of relaxation methods







- Relaxation alternates between *F* / *C*-points
  - F-relaxation: integration in coarse intervals (done in parallel)
  - C-relaxation: one integration step
- Restrict the fine grid approximation and residual to the coarse grid
  - Coarse system is a time re-discretization
- Solve the coarse system and compute the coarse grid error approximation
- Correct the fine grid solution at *C*-points, then apply *F*-relaxation



XBraid implements the MGRIT algorithm, as a nonintrusive black box

- XBraid is designed to be as nonintrusive as possible
  - Requires only the "step" function (identical to the sequential one) and some utilities
  - Only stores C-points to minimize storage
  - Allows "spatial" parallelism (MPI or openMP)
  - Implements V- and F- cycles
  - Possibility of temporal adaptivity if the step function supports it
- Parallel in time with XBraid has delivered speedups as great as 50x in applications (fluid flow in particular)
- Speedups increase with increases in the number of time steps
- *Great example of taking linear algebra ideas into a new context*

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

Multirate methods advance multiple timescales within a problem with differing step sizes

- Consider

$$\frac{dy}{dt} = f^S(t,y) + f^F(t,y), \qquad y(t_0) = y_0$$

- Integrate the slow partition with a method using step size H
- Integrate the fast partition using step size h < H
- Coupling information needs to be shared between slow and fast integrations, and different methods handle this differently
- Result is higher order accuracy while advancing operator partitions at different rates

The time integration community is developing multirate methods to try to address stability and accuracy issues with operator splitting

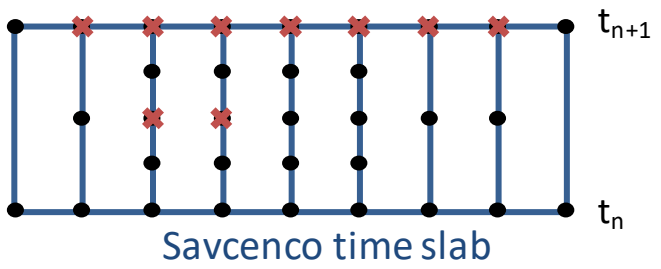
Savcenco, et al. 2007: developed a method for solving partitioned systems when the fast and slow parts are dictated by variable:

$$y' = f(y) \text{ where } y = [y_1 y_2 \ldots | \ldots y_N]$$

slow fast

$$\frac{dy}{dt} = \begin{pmatrix} f^s(t,y) \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ f^f(t,y) \end{pmatrix}$$

In the Savcenco approach, all equations are solved with a large step, an error test is performed, and the variables whose accuracy exceeds tolerance are evolved again with a smaller step size, and so on



Savcenco time slab

This method has been used in many applications, but overall accuracy and stability are not fully understood

Multirate infinitesimal step methods, based on the Runge-Kutta framework, have significant efficiency advantages

Given a RK method for the slow time scale, advance as follows:

1. Set $z_1 = y_{n-1}$
2. For each slow Runge-Kutta stage z_i, $i = 2, \ldots, s+1$:
   a) Let $v\left(t^S_{n,i-1}\right) = z_{i-1}$
   b) Compute slow forcing
   $$r_{i-1} = \frac{1}{c^S_i - c^S_{i-1}} \sum_{j=1}^{i-1} \left(A^S_{i,j} - A^S_{i-1,j}\right) f^S\left(t^S_{n,j}, z_j\right)$$
   c) For $\tau \in [t^S_{n,i-1}, t^S_{n,i}]$, solve the fast ODE
   $$\dot{v}(\tau) = f^F(\tau, v) + r_{i-1}$$
   d) Set $z_i = v(t^S_{n,i})$
3. Set $y_n = z_{s+1}$

The slow stage times are $t^S_{n,j} = t_{n-1} + c^S_j h^S$ and $A^S_{s+1,j} = b^S$

Knoth & Wolke 1998; Schlegel et al. 2009

- Require ***only one traversal*** of any point in time for each RHS partition
- Solves a modified ODE at the fast time scale
- Provides flexibility for fast time scale integration
- Inner method may be subcycled -> a telescopic method (allows for n-rate)
- Because partitions are integrated separately, can leverage single rate infrastructure
- Inner and outer methods can be problem-specific
- Numerous specific methods developed:
  - 2nd and 3rd order [Knoth & Wolke 1998; Schlegel et al. 2009]
  - 4th order [Bauer & Knoth 2019]
  - 3rd and 4th order multirate with IMEX splitting at the slow time scale [Chinomona et al. 2021]

Generalized Additive Runge-Kutta (GARK) methods (Sandu and Günther, 2015) provide a more general framework

Slow base scheme

$$Y_i^{\{s\}} = y_n + H\sum_{j=1}^{s^{\{s\}}} a_{i,j}^{\{s,s\}} f^{\{s\}}\left(Y_j^{\{s\}}\right) + h\sum_{\lambda=1}^{M}\sum_{j=1}^{s^{\{f\}}} a_{i,j}^{\{s,f,\lambda\}} f^{\{f\}}\left(Y_j^{\{f,\lambda\}}\right), \qquad i = 1,\dots s^{\{s\}}$$

Coupling of fast to slow

Subcycling

$$Y_i^{\{f,\lambda\}} = y_n + h\sum_{l=1}^{\lambda-1}\sum_{j=1}^{s^{\{f\}}} b_j^{\{f\}} f^{\{f\}}\left(Y_j^{\{f,l\}}\right) + \sum_{j=1}^{s^{\{s\}}} a_{i,j}^{\{f,s,\lambda\}} f^{\{s\}}\left(Y_j^{\{s\}}\right)$$

Coupling of slow to fast

$$+h\sum_{j=1}^{s^{\{f\}}} a_{i,j}^{\{f,f\}} f^{\{f\}}\left(Y_j^{\{f,\lambda\}}\right) \qquad \lambda = 1,\dots,M, \qquad i = 1,\dots s^{\{f\}}$$

Fast base scheme

$$y_{n+1} = y_n + h\sum_{\lambda=1}^{M}\sum_{i=1}^{s^{\{f\}}} b_j^{\{f\}} f^{\{f\}}\left(Y_i^{\{f,\lambda\}}\right) + H\sum_{j=1}^{s^{\{s\}}} b_i^{\{s\}} f^{\{s\}}\left(Y_i^{\{s\}}\right)$$

Due to the high numbers of method parameters, developing order conditions is challenging.

- $(a^{\{s,s\}}, a^{\{s\}})$ for the slow component; $(a^{\{f,f\}}, a^{\{f\}})$ for the fast component
- The coefficients $a^{\{s,f,\lambda\}}, a^{\{f,s,\lambda\}}$ realize coupling between the components

Many high-order GARK methods have been formulated

- Many degrees of freedom for method design; order conditions have been developed up to 4^{th} order for two-rate methods (Sarshar et al., 2019):
  - 2 for order 1
  - 4 for order 2
  - 10 for order 3
  - 36 for order 4
  - Makes method formulation very challenging
- Unlike MIS methods, GARK can traverse time intervals many times
- Methods up to 4^{th} order have been derived
  - Sarshar, et al., 2019
  - Sandu, 2019, developed MIS methods within the GARK framework using a time dependent forcing at the fast scale rather than the constant forcing in MIS

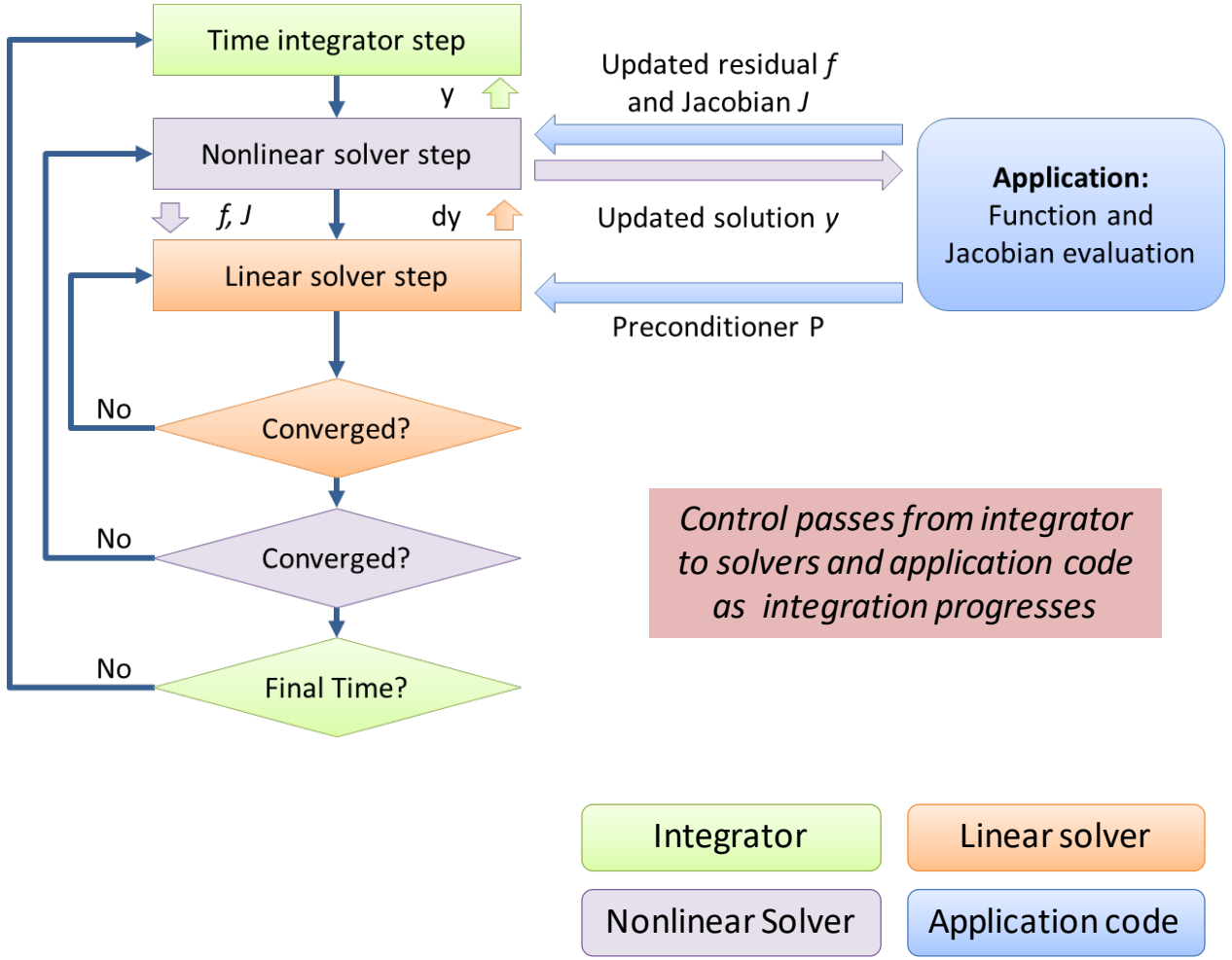
Stability properties for implicit multirate methods are being investigated

- For implicit methods (implicit on both slow and fast), complexities arise:
  - Solve decoupled methods solve each partition separately
  - Solve-coupled methods include coupling between the fast and slow partitions in implicit solves: Significantly more complex solver infrastructure is needed for these
- Stability is surprisingly difficult to analyze – unclear even what test problem to use
- Recent results show limitations of stability for these methods
  - Theorem: A decoupled GARK method is *conditionally* stable for a real 2D test problem; stability depends on M, the ratio between the fast and slow scales
  - Theorem: An internally consistent multirate GARK method of order exactly one has *conditional* scalar stability for all but a finite number of multirate ratios, M
  - Higher orders harder to prove properties, but we expect stability very limited by size of M
- Relax requirement for internal consistency (stage "times" between fast and slow no longer are required to match) gives better stability
  - S. Roberts has new methods that do this
  - Implementation is less general

Roberts, Loffeld, Sarshar, W., and Sandu, 2021

So, why am I talking about time integrators in a course on scalable linear algebra?

- Generally, any implicitness in a time integrator will require solution of nonlinear systems at each time step or stage (in a multistage method)
- These nonlinear solves often require linear solves at each iteration
- The data layout and memory needs of the solvers can impact how the rest of the problem is set up
- Time integrators are in the middle of the algorithm stack – called by application and call linear algebra
- One goal is to make the integrators and nonlinear solvers agnostic of data layout in memory



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
- Implicit time integrators can be made more efficient through control of properties of the nonlinear and linear solver, but these properties can be parameterized, e.g.,
  - Stopping criteria
  - How often to call setup
  - Max iterations before cutting a time step and retrying
  - …
- Nonlinear solvers can be made more efficient through control of properties of the subsidiary linear solver, but these properties can be parameterized
- 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

With an implicit integrator, a nonlinear system must be solved at every stage or every step

Writing the systems as root-finding problems gives:

ODE w/ multistep method: $y^n - h_n \beta_{n,0} f(t_n, y^n) - a_n = 0$

ODE w/ multistage method: $M z_i - h_n A^I_{i,i} f_I(t^I_{n,i}, z_i) - a_i = 0$

DAE w/ multistep method: $F(t_n, y^n, h_n^{-1} \sum_{i=0}^{q} \alpha_{n,i} y^{n-i}) = 0$

$\Big\} \; F(y) = 0$

Applying Newton's method gives a linear system for the update:

$y^{(m+1)} = y^{(m)} + \delta^{(m+1)} \quad J(y^{(m)}) \delta^{(m+1)} = -F(y^{(m)}) \quad J \equiv \partial F / \partial y \; \Big\} \; Ax = b$

For ODEs, this amounts to solving $I - \alpha \Delta t J$ as the linear system

Writing the systems as fixed point problems gives*:

ODE w/ multistep method: $h_n \beta_{n,0} f(t_n, y^n) + a_n = y^n$

ODE w/ multistage method: $h_n A^I_{i,i} f_I(t^I_{n,i}, z_i) + a_i = z_i$

$\Big\} \; G(y) = y$

*DAEs are often too stiff for fixed point methods to be useful

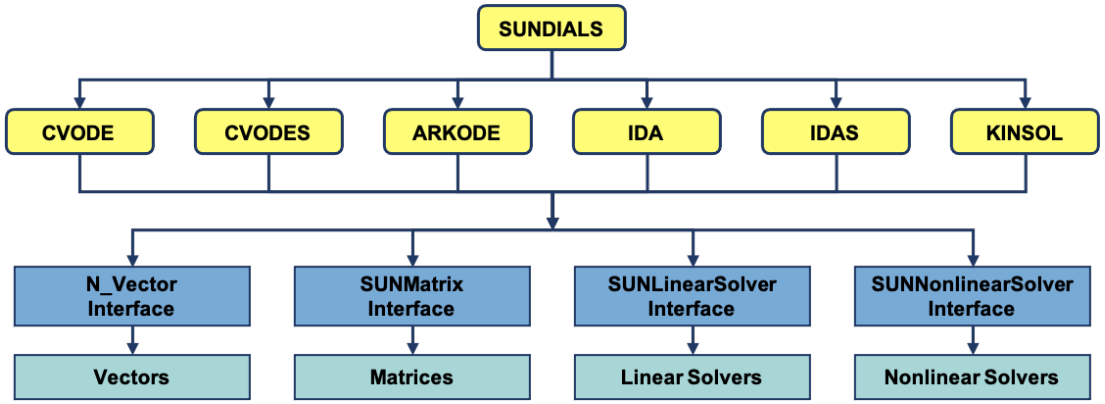
Modern stiff integrator codes will use ***inexact Newton methods*** with iterative solvers for large systems and ***Modified Newton methods*** for smaller ones where the factorization can be reused over many Newton iterations and time steps.

Historical note:

Newton-Krylov methods were developed in tandem with the first fully adaptive efficient implicit multistep integrators in the 1980s.

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 (>90,000 downloads in 2020)
- Detailed user manuals are included with each package



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

SUNDIALS offers packages with linear multistep and multistage methods

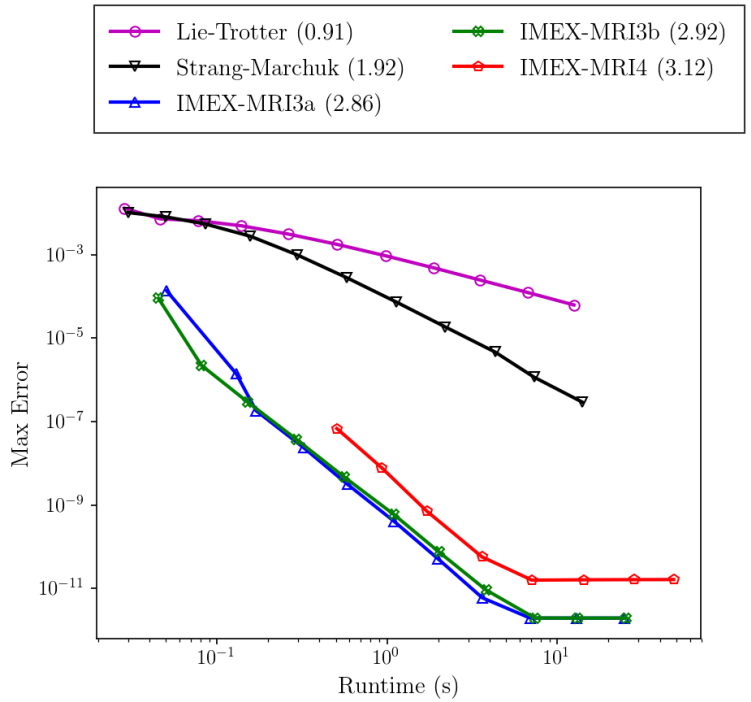
- CVODE, IDA, and their sensitivity analysis variants (forward and adjoint), CVODES and IDAS are based on linear multistep methods
  - CVODE solves ODEs, $\dot{y} = f(t, y)$
  - IDA solves DAEs, $F(t, y, \dot{y}) = 0$
  - Adaptive in both order and step sizes
  - Both packages include stiff BDF methods
  - 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)
  - Multistage embedded methods 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 SUNDIALS vectors and the explicit, implicit, and IMEX methods from ARKStep

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 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 calls ARKStep and thus allows for explicit, implicit, or IMEX integration (user-supplied fast integrator support coming soon)
- The fast time scale can use adaptive or fixed time step sizes
- Supports user-defined method tables for both time scales
- Supports MRI-GARK methods (Sandu, SIAM J. Numer. Anal., 57, 2019), including solve-decoupled, diagonally-implicit treatment of slow scale
- Currently available
  - 2nd and 3rd order multirate MIS methods
  - 4th order multirate MRI-GARK methods, explicit and solve decoupled implicit
  - 3rd and 4th order multirate with IMEX splitting at the slow time scale (soon)



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).

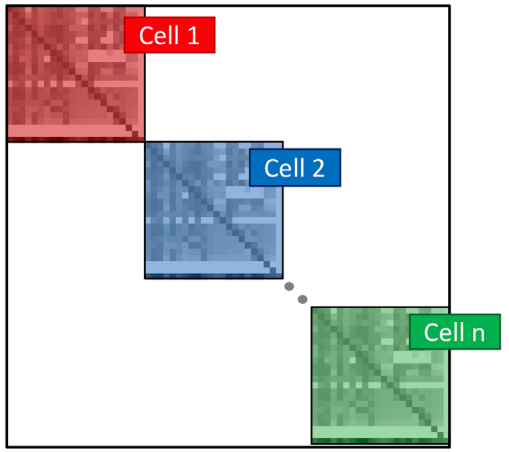
The SUNDIALS strategy for parallelism relies heavily on encapsulation of integrators from data and solvers

- For distributed parallelism:
  - Integrator logic is executed by each distributed task
  - Underlying data structures, solvers, and problem-defining functions implement distributed code
- For GPU parallelism:
  - Keep integrator logic on CPU
  - Put data on the GPU and leave it there
  - Applications perform function evaluations on the GPU: only scalars transfer to the CPU unless the user needs to output their data
  - Supply native vector data structures with optimized methods for each programming environment
  - Supply interfaces to multiple linear solver packages with GPU-enabled direct solvers
- Flexibility for users to supply their own data structures, solvers, and memory managers

The Exascale Computing Project gave us a specific on-node use case

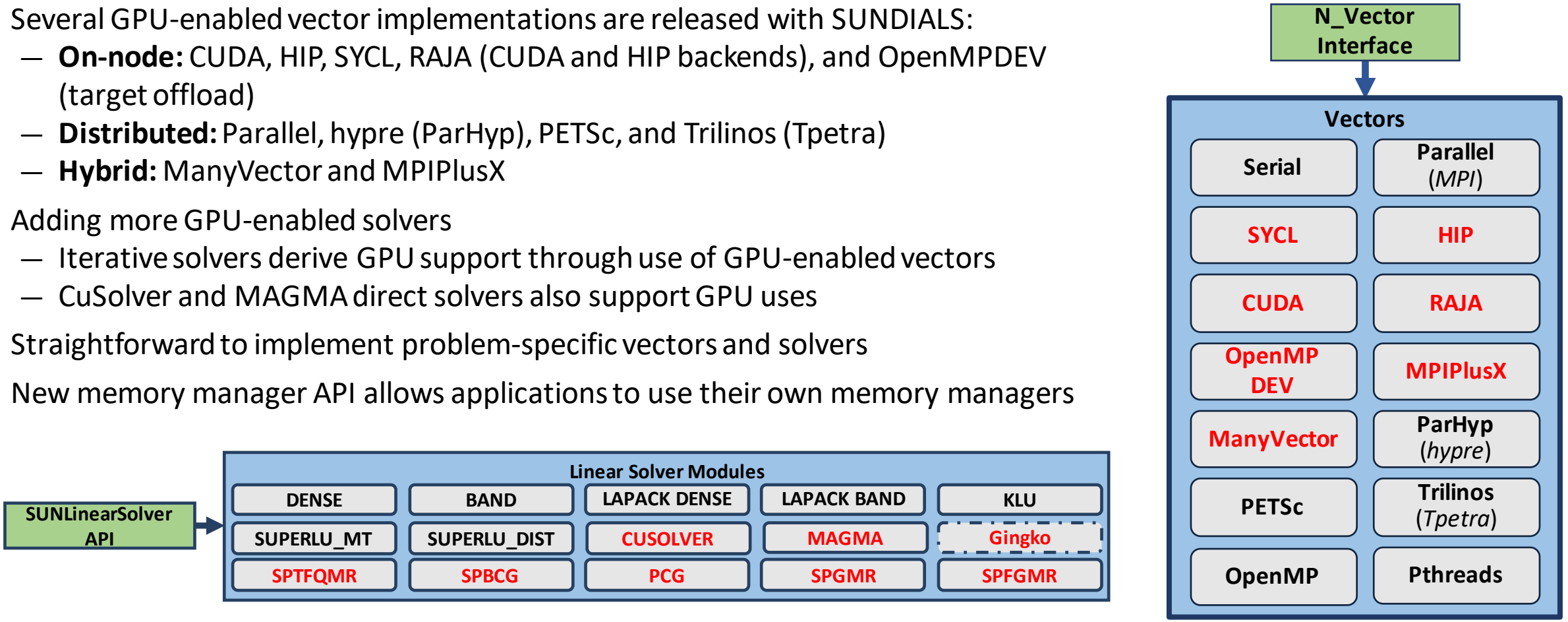
SUNDIALS is used as a local integrator for many small independent subsystems, e.g., reactive flow problems where chemical systems are split from the flow

- Group the systems and integrate the group as a larger system
  - No communication needed between systems
  - Allows for longer vectors and better performance
  - Each system has the same structure
  - Suffers from requiring easy systems to use small time steps dictated by hard systems
- Solve multiple groups simultaneously in different CPU threads/GPU streams
- Linear solver encapsulation allows for batching solves
  - Need fast, GPU-enabled linear solvers designed for block-diagonal linear systems



To support pre-exascale and exascale uses we have added more vectors and solver interfaces to SUNDIALS

- Several GPU-enabled vector implementations are released with SUNDIALS:
  - **On-node:** CUDA, HIP, SYCL, RAJA (CUDA and HIP backends), and OpenMPDEV (target offload)
  - **Distributed:** Parallel, hypre (ParHyp), PETSc, and Trilinos (Tpetra)
  - **Hybrid:** ManyVector and MPIPlusX
- Adding more GPU-enabled solvers
  - Iterative solvers derive GPU support through use of GPU-enabled vectors
  - CuSolver and MAGMA direct solvers also support GPU uses
- Straightforward to implement problem-specific vectors and solvers
- New memory manager API allows applications to use their own memory managers



GPU-enabled direct linear solvers via interfaces to cuSOLVER[1] and MAGMA[2] libraries (Gingko is planned)

[1] https://developer.nvidia.com/cusolver [2] https://icl.cs.utk.edu/magma/

A note on scaling and adaptive integrators

- To ensure the relative weights of various components are accounted for, adaptive integrators measure any error-like quantity with a weighted root mean square norm

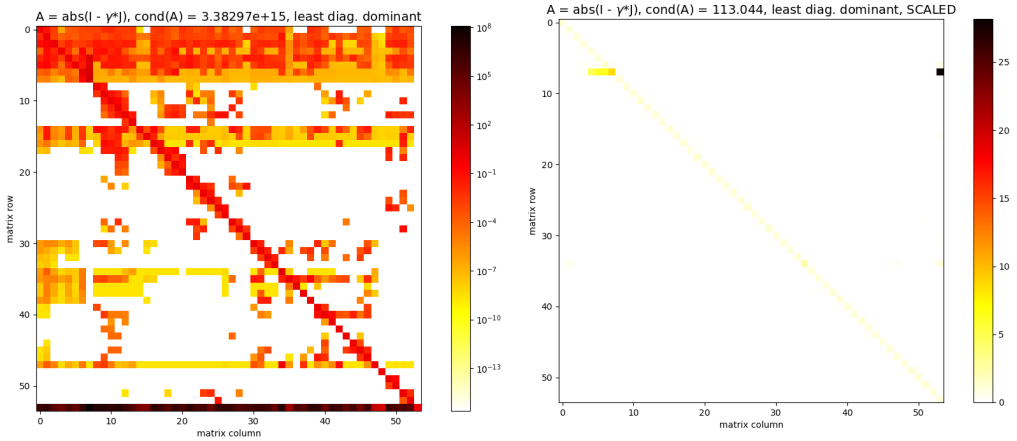
$$w_i = \frac{1}{RTOL|y_i| + ATOL_i} \qquad \|y\|_{\text{wrms}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(w_i\ y_i)^2}$$

- Iterative solvers
  - To help better condition the linear algebra, SUNDIALS applies a scaling vector, w above is set to S_1 and S_2 below, for both equations and unknowns in its preconditioned Krylov methods so that it solves $\tilde{A}\tilde{x} = \tilde{b}$

$$\tilde{A} = S_1 P_1^{-1} A P_2^{-1} S_2^{-1}, \qquad \tilde{b} = S_1 P_1^{-1} b, \qquad \tilde{x} = S_2 P_2 x,$$

  - These factors are critical to good performance

- Direct solver – SUNDIALS did not scale matrices in the past as most linear algebra packages provided pivoting. With direct solvers now required on the GPU, we are rethinking this practice



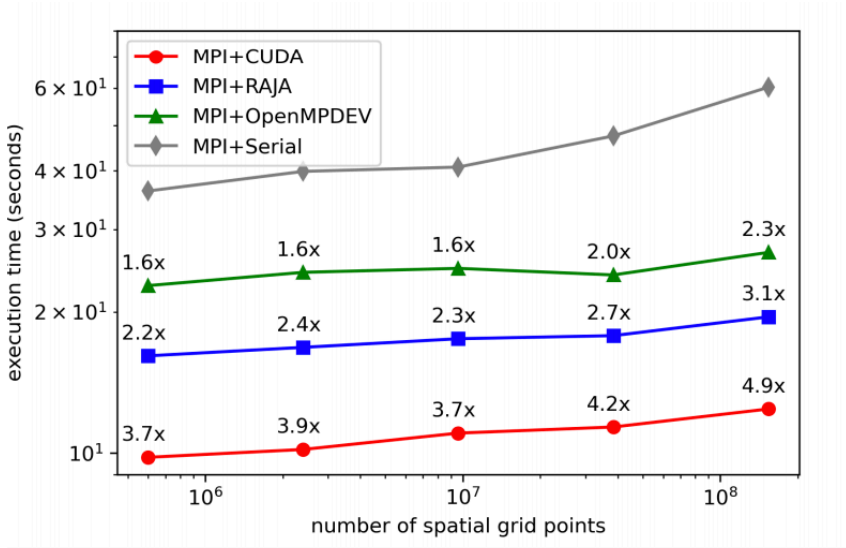
Unscaled and scaled Jacobians from a dodecane reaction system modeled with PelePhysics. The condition number dropped from ~3e15 to ~100.

MPI+X performance on a demonstration problem on Summit shows benefit from GPU use

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

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

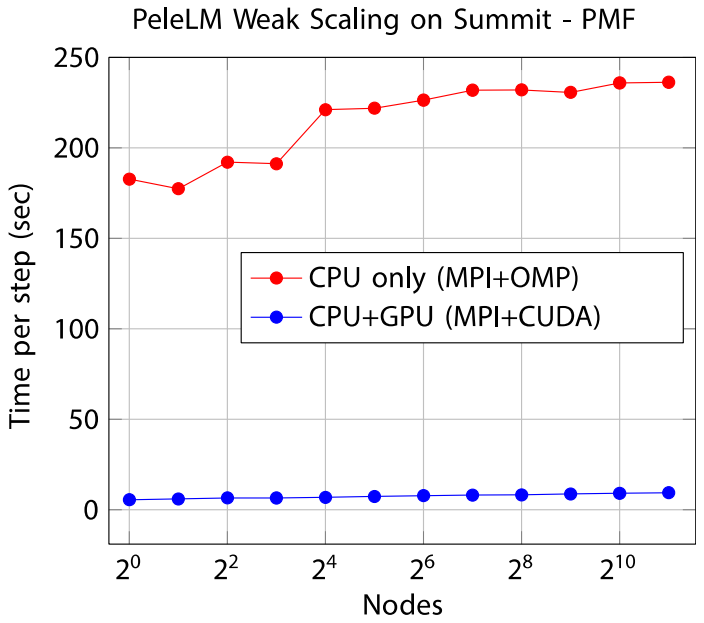
- Reactions treated implicitly; advection treated explicitly
- Nonlinear solver is a "Task-local" Newton solve (solve per spatial node) + direct inversion
- Greatest speedup achieved when using the CUDA vector and CUDA for the RHS



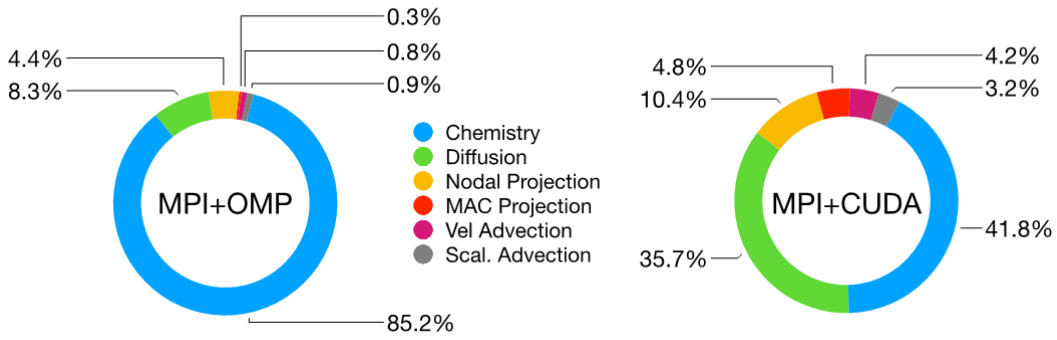
Weak scaling using the MPI+X vector. Configurations include 1, 4, 16, 64, and 256 Summit nodes, each with 6 MPI tasks per node(1 MPI task per GPU) except the MPI-only which uses 40 MPI tasks per node. Data points are annotated with the speedup over the configuration with the MPI+Serial vector.

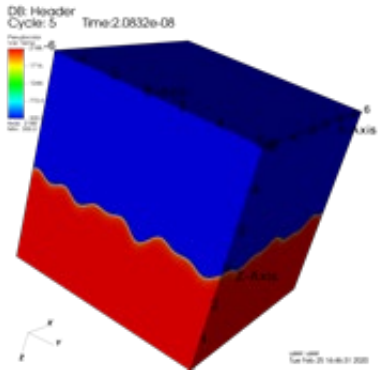
We have been working with the Pele combustion project to transition to use of GPUs

- The Pele code suite uses AMReX for structured grid adaptive mesh refinement and uses an integrator to evolve the combustion mechanism within each grid cell
- **PelePhysics:** Interfaces to CVODE and ARKODE, batched cuSolver interface allows for setting matrix entries directly on GPU, HIP and CUDA vectors interfaced. This infrastructure is available to both PeleC and PeleLM
- **PeleLM:** CVODE is default chemistry integrator; testing solver options, iterative and direct
- **PeleC:** Explicit integrator in ARKODE is now the default integrator for chemistry; ERKStep integrator in SUNDIALS with CUDA provides 6x speedup over fastest CPU configuration



PeleLM Profiles





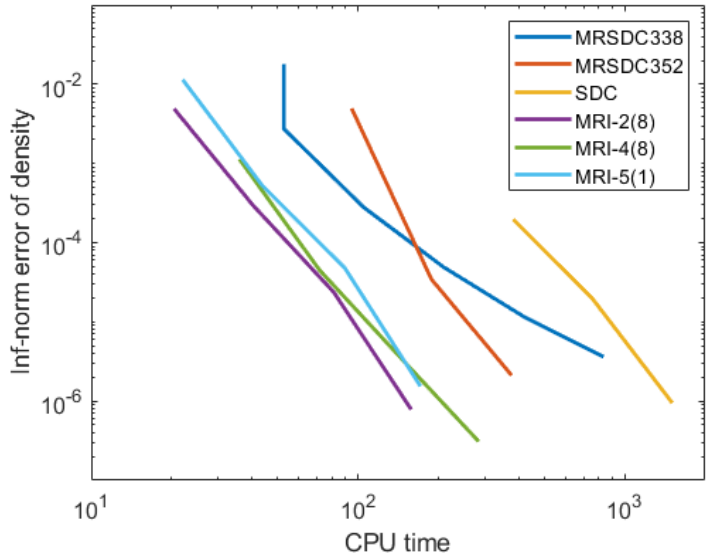
Single-level premixed flame

- Wrinkled flame sheet
- 22 species, 84 reactions
- 4.3M cells / node
- ~ 25x speedup on 2048 nodes
- 42 P9 cores vs. 6 V100's per node on Summit

Figures and results courtesy of Marc Day (NREL) and Jon Rood (NREL)

Multirate methods can be competitive with SDC

- Interfaced ARKODE package with AMReX adaptive mesh refinement library allowing explicit, implicit, and implicit-explicit single and multirate time integrators to evolve AMReX multiphysics systems
- Compared ARKODE MRIStep multirate infinitesimal step methods with AMReX native SDC implementation which allows operator split with subcycling as the inner integrator
- Comparisons done on an advection-diffusion-reaction system with reactions integrated with 1 to 8 fast steps per 1 slow step for the advection and diffusion
- MRI methods show a significant performance advantage over the single-rate SDC, and MRI methods have a sizable efficiency advantage over the SDC multirate schemes (MRSDC methods)



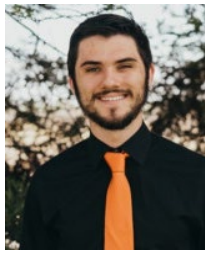
Comparison of the error vs CPU time for SDC schemes using the standard four iterations per multiphysics time step with SUNDIALS IMEX-MRI multirate methods on an advection-diffusion-reaction system. MRI methods are denoted MRI-P(k), where is P the order of the fast method and k is the number of fast steps per slow step. (Tests run on LLNL Quartz system using 8 MPI ranks distributed over a single node with 2 Xeon E5-2695 v4 chips.

Concluding remarks

- As HPC systems continue to gain capacity and speed, scientific simulations continue to grow in complexity and temporal scales
- Effective integration methods
  - Rely on operator splittings but achieve high order accuracy and stability
  - Introduce implicitness only where necessary in order to achieve faster run times
- Implicit approaches require efficient nonlinear and linear algebraic solvers
- Well-designed integrator packages can easily take advantage of new and efficient solver software underneath the integrators
- Parallelism in time will be essential for continued progress in many fields

Collaborators

The SUNDIALS team


Cody Balos (LLNL)


David Gardner (LLNL)


Alan Hindmarsh (LLNL)


Dan Reynolds (SMU)

- SUNDIALS alumni
  - Slaven Peles (PNNL)
  - Radu Serban (Univ. WI)
- Rob Falgout (LLNL)
- John Loffeld (LLNL)
- Andy Nonaka (LBNL)
- Jacob Schroder (Univ. NM)
- Rujeko Chinomona (SMU)
- Marc Day (NREL)
- Adrian Sandu (VA Tech)
- Steven Roberts (VA Tech)
- Chris Vogl (LLNL)
- Andrew Steyer (SNL)
- Peter Brown (LLNL)

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