ARKode: A library of high order implicit/explicit methods for multi-rate problems

Daniel R. Reynolds[†], Carol S. Woodward[*],
David J. Gardner[†] and Alan C. Hindmarsh[*]

reynolds@smu.edu

[†]Department of Mathematics, Southern Methodist University
[*]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

SIAM Conference on Parallel Processing for Scientific Computing
February 21, 2014

SMU FASTMATH Lawrence Livermore National Laboratory

Outline

Outline

Multiphysics Problems

“Multiphysics problems” typically involve a variety of interacting processes:

- System of components coupled in the bulk [cosmology, combustion]
- System of components coupled across interfaces [climate, tokamak fusion]

Typical difficulties in simulating multiphysics problems include:

- Multi-rate proceses, but too close to analytically reformulate.
- Optimal solvers may exist for some pieces, but not for the whole.
- Mixing of stiff/nonstiff processes, challenging standard standard solvers.

Many codes utilize lowest-order time step splittings, but may suffer from:

- Low accuracy – typically $\mathcal{O}(\Delta t)$-accurate; symmetrization/extrapolation may improve this but at significant cost [Ropp, Shadid,& Ober 2005].
- Poor/unknown stability – even when each part utilizes a 'stable' step size, the combined problem may admit unstable modes [Estep et al., 2007].

Increased Implicit Accuracy & Stability

Current production IVP libraries focus on linear multistep methods:

- Single implicit solve per step; high order comes through reusing old steps.
- Linearly stable up to $\mathcal{O}(\Delta t^2)$, but stability region shrinks rapidly for higher order, with little utility over $\mathcal{O}(\Delta t^5)$.
- Adaptivity is based on similarity between predictor (explicit) & corrector (implicit); primarily valid in regimes where both methods useful, i.e. questionable for stiff problems.

Runge-Kutta methods:

- No Dahlquist barrier – A-stability possible even at high order. B-stability provable for many methods.
- Adaptivity based on embedded methods, allow implicit/stable solves for both solution & embedding $\Rightarrow$ applicable across a wider problem set.
- Benefits come at the price of multiple implicit solves per step, or a single but larger implicit solve per step.

Single-Step Evolution for Space-Time Adaptivity

While temporally adaptive, traditional IVP libraries limit spatial adaptivity:

- Assume the solution $y \in \mathbb{R}^N$, with N fixed throughout solve.
- Essential for linear multistep methods since step history generates order.
- Spatial adaptivity possible, but requires costly projection of step history and internal data structures onto new spatial domain.

Runge-Kutta methods:

- Since high order is obtained via stages *within* step, no history is required.
- Only need $y \in \mathbb{R}^{N_k}$, with N_k fixed within (but variable between) steps.
- Spatial adaptivity between steps easily incorporated, assuming solver data structures support vector resizing.

Outline

2-Additive Runge-Kutta Methods [Ascher et al. 1997; Araújo et al. 1997]

ARKode employs an additive Runge-Kutta formulation, supporting up to two split components: *explicit* and *implicit*,

$$M\dot{y} = f_E(t,y) + f_I(t,y), \quad t \in [t_0, T], \quad y(0) = y_0,$$

- $M = M(t)$ is any nonsingular linear operator (mass matrix, typically $M = I$),
- $f_E(t,y)$ contains the explicit physics,
- $f_I(t,y)$ contains the implicit physics.

We combine two s-stage methods (ERK + DIRK). Denoting $t_{n,j} = t_n + c_j \Delta t$,

$$Mz_i = My_n + \Delta t \sum_{j=1}^{i-1} A^E_{i,j} f_E(t_{n,j}, z_j) + \Delta t \sum_{j=1}^{i} A^I_{i,j} f_I(t_{n,j}, z_j), \quad i = 1, \ldots, s,$$

$$My_{n+1} = My_n + \Delta t \sum_{j=1}^{s} b_j \left(f_E(t_{n,j}, z_j) + f_I(t_{n,j}, z_j)\right) \quad \text{[solution]}$$

$$M\tilde{y}_{n+1} = My_n + \Delta t \sum_{j=1}^{s} \tilde{b}_j \left(f_E(t_{n,j}, z_j) + f_I(t_{n,j}, z_j)\right) \quad \text{[embedding]}$$

ARK Coefficients

Allows two Butcher tables that define the method:

- $\{c_i\}_{i=1\,\ldots,s}$ are the shared *stage times* for the two tables
- $\{b_i\}_{i=1,\ldots,s}$ are the shared *solution coefficients* for the two tables
- $\left\{\tilde{b}_i\right\}_{i=1,\ldots,s}$ are the shared *embedding coefficients* for the two tables
- $\{a^E_{i,j}\}_{i=1,\ldots,s,j=1,\ldots,i-1}$ are the *explicit method coefficients*
- $\{a^I_{i,j}\}_{i=1,\ldots,s,j=1,\ldots,i}$ are the *diagonally-implicit method coefficients*

Notes:

- Explicit method: $a^I_{i,j} = 0$ and all physics in $f_E(t,y)$.
- Implicit method: $a^E_{i,j} = 0$ and all physics in $f_I(t,y)$.
- ImEx method: coefficients are derived in unison to satisfy coupling *between* components (unlike arbitrary splittings).

Solution Algorithm – Stage Solutions, z_i, $i = 1, \ldots, s$

- **ERK stages**: The stage is computed explicitly as a linear combination of previous stage right-hand sides, followed by a simple linear solve (if $M \neq I$)

$$M z_i \;=\; M y_n + h \sum_{j=1}^{i-1} A^E_{i,j} f_E(t_{n,j}, z_j)$$

- **DIRK stages**: The stage is computed as the solution of a nonlinearly implicit equation, with right-hand side like an ERK method,

$$M z_i - h A^I_{i,i} f_I(t_{n,i}, z_i) \;=\; M y_n + h \sum_{j=1}^{i-1} A^I_{i,j} f_I(t_{n,j}, z_j)$$

- **ARK stages**: The stage is computed like a DIRK, but the right-hand side contains explicit components,

$$M z_i - h A^I_{i,i} f_I(t_{n,i}, z_i) \;=\; M y_n + h \sum_{j=1}^{i-1} \left(A^E_{i,j} f_E(t_{n,j}, z_j) + A^I_{i,j} f_I(t_{n,j}, z_j) \right)$$

Solution Algorithm – Solution & Error Estimate

Once all stages, z_i, $i = 1, \ldots, s$, have been computed, we finish the step:

- **Solution**: just a linear combination of the stage right-hand sides, followed by a simple linear solve (if $M \neq I$):

$$My_{n+1} = My_n + h \sum_{j=1}^{s} b_j \left(f_{E,j} + f_{I,j} \right)$$

- **Local Error Estimate**: the embedding is like y_{n+1} but with coefficients $\tilde{b}_j$, so we compute the local temporal error estimate T by computing/solving:

$$MT = h \sum_{j=1}^{s} (b_j - \tilde{b}_j) \left(f_{E,j} + f_{I,j} \right)$$

- **Scalar Error Estimate**: to determine step success/failure, we ensure that the scalar error estimate satisfies (where $y_n \in \mathbb{R}^{N_k}$):

$$\|T\|_{WRMS} := \left(\frac{1}{N_k} \sum_{i=1}^{N_k} \left(\frac{T_i}{r_{\text{tol}} \, |y_{n,i}| + a_{\text{tol}}} \right)^2 \right)^{1/2} \leq 1.$$

Implicit Solver – with multiple solves per step, efficiency is paramount

Nonlinear:

- *Modified Newton* (serial, dense linear algebra) – Newton method that reuses Jacobian between multiple stages/steps for increased efficiency.
- *Inexact Newton* – linear solver tolerances are modified at each step to achieve superlinear convergence with minimal linear solver work.
- *Andersen-accelerated fixed-point* (see Carol Woodward's talk, MS 6) – fixed point solver with GMRES-like subspace acceleration.
- Nonlinear tolerances adjusted by solver to attain requested solution accuracy without oversolves.

Linear:

- Direct – full and band solvers from SUNDIALS or LAPACK; KLU & SuperLU coming soon.
- Krylov – GMRES, FGMRES, BiCGStab, TFQMR or PCG.
- User-supplied right/left preconditioning supported.
- Newton and mass-matrix solvers can be mix-and-matched.

Efficiency Enhancements

Additional options that may increase efficiency:

- *Implicit predictors* – supports simple/safe predictors, through quadratic Hermite predictors. Also allows user-supplied predictors.
- *Advanced temporal adaptivity controllers* – supports modern control-theoretic algorithms for maximizing step sizes while reducing error/convergence failures. Also supports user-supplied controllers.
- *Explicit stability control* – supports user-supplied routines that provide maximally stable explicit step, to minimize error failures.
- *Data structure resize capability* – for problems with changing N_k, data structures may be *resized* without requiring destruction/reinitialization.
- All internal solver parameters are fully documented and modifiable by the user to tune for a particular problem.

ARKode, the newest member of SUNDIALS

As a part of the FASTMath SciDAC Institute, ARKode is being integrated as a new component solver within SUNDIALS.

- Similar user interface as CVODE, albeit with separate user-specified $f_E(t,y)$ and $f_I(t,y)$, and potentially user-supplied M or My routines.
- Data structure agnostic – as long as the basic vector kernels are supplied, problem-specific data structures are allowed. Will even call a user-supplied vector "resize" function to expand/contract the data structure.
- High-order accurate dense output, allowing efficient interpolation of results between integration steps.
- Parameters optimized for iterative solvers and large-scale parallelism.
- Exhaustive suite of example and regression test problems.
- Main site: http://faculty.smu.edu/reynolds/arkode
  Repository: http://bitbucket.org/drreynolds/arkode_pub

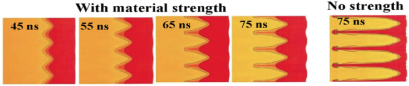
Outline

ParaDiS – Parallel Dislocation Dynamics Simulator

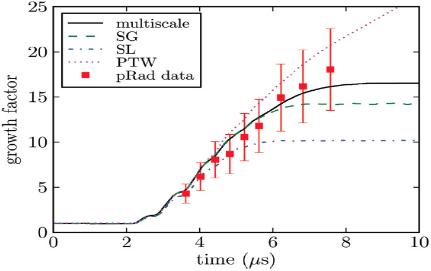
Modeling material strain hardening:

- A *dislocation* is a line defect in the regular crystal lattice structure.
- *Plasticity* is caused by multiple dislocation lines forming in response to an applied stress/strain.
- ParaDiS simulates the motion, multiplication, and interactions of discrete dislocation lines.
- Attempts to connect dislocation physics with material strength, to understand how material strength changes under applied load.

Growth factor calculations in an explosively driven Rayleigh-Taylor instability:



[Park et al., *PRL*, 104, 135504 (2010)]

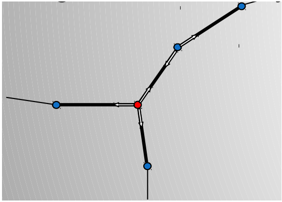


[Barton et al., *J. App. Phys.*, 109, 073501 (2011)]

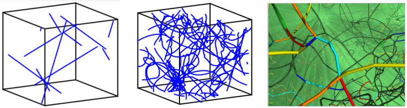
SMU FASTMATH Lawrence Livermore National Laboratory

The ParaDiS Model

- Discretize dislocation lines as segments terminated by nodes



- Force calculations utilize local and FMM methods
- MPI + OpenMP parallelization
- Fully adaptive data structure, with topology changes at every step



Algorithm flow:

- Nodal force calculation:

$$f_i^{\text{tot}}(t,r) = f_i^{\text{self}}(r) + f_i^{\text{seg}}(r) + f_i^{\text{ext}}(t,r)$$

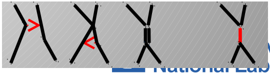
- Nodal velocity calculation (material-dependent M_{ij}):

$$v_i(t,r) = \frac{dr_i}{dt} = M_{ij}\, f_j(t,r)$$

- Time integration:

$$r_i(t+\Delta t) = r_i(t) + \int_t^{t+\Delta t} v_i(t,r)\,\mathrm{d}t$$

- Topology changes (insert/merge nodes):



SMU

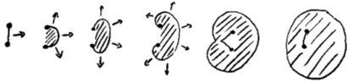
FASTMATH

Lawrence Livermore National Laboratory

ParaDiS Results – Frank-Read Source

Simple test problem:

- Single initial dislocation
- Constant strain bends/reconnects, creating two concentric dislocations



- Strain rate 1 s^{-1}; Final time 50 μs
- Comparison between:
  - Native Trapezoid solver: basic fixed-point (2,3 iters)
  - KINSOL Trapezoid solver: AA (2-4 iters)
  - DIRK, $\mathcal{O}(\Delta t^3) \to \mathcal{O}(\Delta t^5)$: NK and AA (4 iters each)

Method	**Steps**	**% Speedup**
Trap FP I2	6284	0.0
Trap FP I3	4990	20.0
Trap AA I2 V1	6447	-4.9
Trap AA I3 V2	2316	61.7
Trap AA I4 V3	2017	66.3
DIRK3 NK I4	242	93.0
DIRK4 NK I4	213	95.3
DIRK5 NK I4	212	92.1
DIRK3 AA I4 V3	127	97.5
DIRK4 AA I4 V3	194	95.6
DIRK5 AA I4 V3	128	96.9

[Graphic: http://classes.geology.illinois.edu/07fallclass/geo411/ductile/ductile.html]

SMU FASTMATH Lawrence Livermore National Laboratory

ParaDiS Results – Target Test Problem

"Real" problem, mid-simulation:

- Body-centered-cubic crystal structure, $\Omega = 4.25\ \mu\text{m}^3$
- Strain rate 10^2 s^{-1}
- 3.3 μs $\leq t \leq$ 6.25 μs
- ~ 2850 initial nodes, ~ 5000 final
- MPI test runs with 16 cores
- Comparison between:
    - Native Trapezoid solver: basic fixed-point (2 iters)
    - KINSOL Trapezoid solver: AA (2-6 iters)
    - DIRK $\mathcal{O}(\Delta t^3)$ solver: AA (2-6 iters), $\varepsilon_n = 1$
- Larger tests (~ 250k cores) ongoing

Method	Steps	% Speedup
Trap FP I2	9137	0.0
Trap AA I4 V3	3262	42.9
Trap AA I5 V4	2987	45.0
Trap AA I6 V5	2032	55.1
Trap AA I7 V6	1981	53.5
DIRK3 AA I4 V3	323	65.1
DIRK3 AA I5 V4	297	66.9
DIRK3 AA I6 V5	303	64.9
DIRK3 AA I7 V6	311	63.0
DIRK5 AA I4 V3	280	51.2
DIRK5 AA I5 V4	241	53.9
DIRK5 AA I6 V5	246	50.5
DIRK5 AA I7 V6	274	45.0

Outline

Conclusions

ARK methods allow accurate/stable methods for a variety of problems:

- No Dahlquist barrier – high accuracy & stability simultaneously possible
- Allows adaptive ERK, DIRK or fully-coupled ImEx methods
- Embeddings allow robust error estimation and timestep adaptivity
- Single-step methods play well with spatial adaptivity
- ImEx allows "convenient" preconditioners that treat only stiff components

The ARKode library:

- Flexible solver infrastructure, with a variety of nonlinear/linear solvers
- Support for non-identity mass matrices (FEM)
- Allows on-the-fly vector resizing
- Freely-available, included in the upcoming SUNDIALS release

Thanks & Acknowledgements

Collaborators/Students:

- Carol S. Woodward [LLNL]
- Alan C. Hindmarsh [LLNL]
- David J. Gardner [SMU, PhD]

Current Grant/Computing Support:

- DOE SciDAC & INCITE Programs
- LLNL Computation
- SMU Center for Scientific Computation

Software:

- ARKode – http://faculty.smu.edu/reynolds/arkode
- SUNDIALS – https://computation.llnl.gov/casc/sundials

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

First-Order Splittings

Denote $S_i(h, u(t_n))$ as a solver for the component $\partial_t u = f_i(t, u)$ over a time step $t_n \to t_n + h \equiv t_{n+1}$, with initial condition $u(t_n)$.

To evolve $u(t_n) \to u(t_{n+1})$, we can use different solvers at the same h,

$$\begin{aligned} \hat{u} &= S_1\left(h, u(t_n)\right), \\ u(t_{n+1}) &= S_2\left(h, \hat{u}\right), \end{aligned}$$

or we may subcycle time steps for individual components,

$$\begin{aligned} \hat{u}_{j+1} &= S_1\left(\tfrac{h}{m}, \hat{u}_j\right), \ j = 0, \ldots, m, \quad \hat{u}_0 = u(t_n), \\ u(t_{n+1}) &= S_2\left(h, \hat{u}_m\right), \end{aligned}$$

Unless the S_i commute [i.e. $S_1(h, S_2(h, u)) = S_2(h, S_1(h, u))$] or the splitting is symmetric, these methods are at best $O(h)$ accurate
(no matter the accuracy of the individual solvers).

Fractional Step (Strang) Splitting [Strang 1968]

"Strang splitting" attempts to achieve a higher-order method using these separate component solvers, through manually symmetrizing the operator:

$$\hat{u}_1 = S_1\left(\tfrac{h}{2}, u(t_n)\right),$$

$$\hat{u}_2 = S_2\left(h, \hat{u}_1\right),$$

$$u(t_{n+1}) = S_1\left(\tfrac{h}{2}, \hat{u}_2\right).$$

This approach is $O(h^2)$ as long as each S_i is $O(h^2)$.

However:

- This asymptotic accuracy may not be achieved until h is very small, since error terms are dominated by inter-process interactions [Ropp, Shadid,& Ober 2005].
- Numerical stability isn't guaranteed *even if h is stable for each component* [Estep et al., 2007].

Operator-Splitting Issues – Accuracy [Ropp, Shadid, & Ober 2005]

Coupled systems can admit destabilizing modes not present in either component, due to *numerical resonance instabilities* [Grubmüller 1991].

Brusselator Example (Reaction-Diffusion):

$$\partial_t T = \frac{1}{40}\nabla^2 T + 0.6 - 3T + T^2 C,$$
$$\partial_t C = \frac{1}{40}\nabla^2 C + 2T - T^2 C,$$

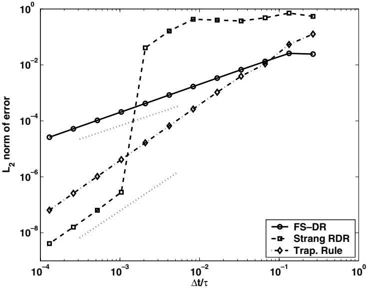
Three solvers:

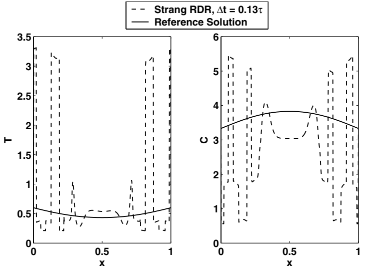
(a) Basic split: D (trap.) then R (subcycled BDF).

(b) Strang: $\frac{h}{2}$R, hD, $\frac{h}{2}$R,

(c) Fully implicit trapezoidal rule,

Results:
(a) is stable but inaccurate for all tests;
(b) unusable until h is "small enough".





Operator Splitting Issues – Accuracy [Estep 2007]

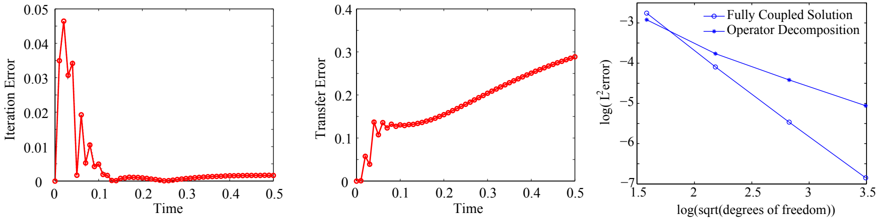
Consider $\Omega = \Omega_1 \cup \Omega_2$ where the subdomains share a boundary $\Gamma = \partial\Omega_1 \cap \partial\Omega_2$:

$$\partial_t u_1 = \nabla^2 u_1, \; x \in \Omega_1, \qquad \partial_t u_2 = \tfrac{1}{2}\nabla^2 u_2, \; x \in \Omega_2,$$

$$u_1 = u_2, \qquad \nabla u_1 \cdot n = \nabla u_2 \cdot n, \qquad \text{for } x \in \Gamma.$$

Solved using one Gauss-Seidel iteration: S_1 on Ω_1, then S_2 on Ω_2 (both trapezoidal). Errors from not iterating to convergence, and from error transfer between subdomains.

Using adjoints, they measured these errors separately:



- Error from incomplete iteration decreased with time.
- Transfer error accumulated and became dominant with time.
- While each S_i was $O(h^2)$, the coupled method was only $O(h)$.

Operator-Splitting Issues – Stability [Estep et al., 2007]

Second Reaction-Diffusion Example (split subcycling; exact solvers):

$$\partial_t u = -\lambda u + u^2, \quad u(0) = u_0, \quad t > 0.$$

Phase 1 (R): $\partial_t u_r = u_r^2, \quad u_r(t_n) = u_n, \quad t \in [t_n, t_{n+1}],$

Phase 2 (D): $\partial_t u_d = -\lambda u_d, \quad u_d(t_n) = u_r(t_{n+1}), \quad t \in [t_n, t_{n+1}].$

True solution, $u(t) = \dfrac{u_0 e^{-\lambda t}}{1 + \frac{u_0}{\lambda}\left(e^{-\lambda t} - 1\right)},$ is well-defined $\forall t$ if $\lambda > u_0$.

Split solution, $u(t_{n+1}) = \dfrac{u(t_n) e^{-\lambda h}}{1 - u(t_n) h},$ can blow up in finite time.

Results using 50 time steps, with varying amounts of subcycling.