The ARKode Library – Flexible and Accurate Multiphysics Time Integrators

Daniel Reynolds[1], David Gardner[2], Carol Woodward[2], Jean M. Sexton[3]
& the SUNDIALS team

reynolds@smu.edu, gardner48@llnl.gov, woodward6@llnl.gov, jmsexton@lbl.gov

[1] Department of Mathematics, Southern Methodist University
[2] Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
[3] Center for Computational Sciences & Engineering, Lawrence Berkeley National Laboratory

7 August 2018

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]

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/nonstiff processes, challenging legacy algorithms.

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

- Low accuracy – typically $\mathcal{O}(h)$-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].

Need for Flexible & Accurate Multirate Integrators

"Multirate" methods evolve distinct problem components with their own rate-specific time steps. Historical approaches:

- Simple $\mathcal{O}(h)$-accurate subcycling approaches
- Interpolation to handle fast/slow coupling (typically $\mathcal{O}(h^2)$, sometimes $\mathcal{O}(h^3)$) [Kværnø & Rentrop, 1999; ...].
- Extrapolation methods to 'bootstrap' accuracy for low order methods [Engstler & Lubich, 1997; Constantinescu & Sandu, 2013; ...].

Next-generation methods will require a variety of criteria:

- High-order accuracy & stability, both within and between components
- Flexible rate structure within integration, or even to dynamically identify 'fast' vs 'slow' partitioning of components
- Robust temporal error estimation & adaptivity of step size(s)
- Ability to apply solver optimal algorithms for individual components
- Built-in support for spatial adaptivity
- Enable problem-specific options, e.g. SSP or symplectic for specific components
- Support for testing a variety of methods and solution algorithms

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*,

$$M\dot{y} = f^E(t,y) + f^I(t,y), \quad t \in [t_0, t_f], \quad y(0) = y_0,$$

- $M = M(t)$ is any nonsingular linear operator (mass matrix, typically $M = I$),
- $f^E(t,y)$ contains the explicit terms,
- $f^I(t,y)$ contains the implicit terms.

We combine two s-stage methods; denoting e.g. $t^E_{n,j} = t_n + c^E_j h_n$, $h_n = t_{n+1} - t_n$:

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

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

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

ARK Coefficients

Two Butcher tables define the method:

- $\left\{c^E, A^E, b^E, \tilde{b}^E\right\}$ define the *explicit Butcher table*
- $\left\{c^I, A^I, b^I, \tilde{b}^I\right\}$ define the *diagonally-implicit Butcher table*

Formulation supports adaptive or fixed-step ERK, DIRK and ARK methods:

- Explicit methods: $A^I = 0$ and all IVP terms are in $f^E(t, y)$.
- Implicit methods: $A^E = 0$ and all IVP terms are in $f^I(t, y)$.
- ARK methods: both tables derived in unison to satisfy inter-component coupling conditions.
- Fixed-step methods ($h_n = h$), or user-defined h_n: $\tilde{b}^E$ and $\tilde{b}^I$ need not be defined.

Solving each stage z_i, $i = 1, \ldots, s$

Each stage is implicitly defined by solving a root-finding problem:

$$0 = G_i(z) \equiv Mz - My_n - h_n \left[A^I_{i,i} f^I(t^I_{n,i}, z) + \sum_{j=1}^{i-1} \left(A^E_{i,j} f^E(t^E_{n,j}, z_j) + A^I_{i,j} f^I(t^I_{n,j}, z_j) \right) \right]$$

- if $f^I(t, y)$ is *linear* in y then G_i is linear, and we need only solve a single linear system for each z_i,
- otherwise G_i is nonlinear, and must be must utilize an iterative nonlinear solver.

Nonlinear solver options:

- *Modified Newton* (direct linear solvers) reuses Jacobian between multiple stages/steps.
- *Inexact Newton* (iterative linear solvers) sets linear tolerances to minimize linear solver work; preconditioner reused between multiple stages/steps (if supplied).
- *Anderson-accelerated fixed point* solver requires no linear solves; utilizes GMRES-like acceleration over subspace of preceding iterates.
- Soon (Fall 2018): User-supplied solvers may be "plugged in" via a clear, object-oriented API.

Linear Solvers and Vector Data Structures

Linear solver options:

- Direct – dense/band/sparse solvers (incl. LAPACK, KLU & SuperLU)
- Krylov – GMRES, FGMRES, BiCGStab, TFQMR or PCG
  - support user-supplied preconditioning (left/right/both)
  - support residual/solution scaling for "unit-aware" stopping criteria
  - support "matrix-free" methods through approximation of product Jv, where $J \equiv \frac{\partial}{\partial y} f^I(t, y)$
- User-supplied solvers may be "plugged in" via a clear, object-oriented API.

All solvers (except for direct linear) formulated via vector operations:

- Serial, MPI, pThreads, OpenMP, PETSc, CUDA, RAJA and *hypre* vectors are supplied
- User-supplied data structures may be "plugged in" via a clear, object-oriented API.

ARKode Flexibility Enhancements

Additionally, ARKode includes enhancements for multi-physics codes, including:

- Variety of built-in RK tables; supports user-supplied
- Variety of built-in adaptivity functions; supports user-supplied
- Variety of built-in implicit predictor algorithms
- Ability to specify that problem is linearly implicit
- Ability to resize data structures based on changing IVP size

Outline

ARKode usage skeleton

1. Create vector of initial conditions values
2. Create ARKode integrator: supply initial conditions and RHS routine(s)
3. Set optional inputs: tolerances, method order, initial step size, ...
4. Optionally create matrix, linear & nonlinear solver objects; attach to ARKode
5. Advance solution in time (typically in a loop):
   ```
   ier = ARKode(arkode_mem, t_out, y_out, &t_ret, itask);
   ```
6. Retrieve optional outputs: integrator statistics, interpolated solution values, ...
7. Deallocate memory for solver, solution vector and optional objects

Vector object: y

Users can supply their own application-specific "`N_Vector`" module underneath any SUNDIALS package:

- *Content* structure specifies data and information needed to create new vectors.
- Implementations of vector *operations* on the supplied structure.
- Routines to clone vectors for use within SUNDIALS.
- All parallelism resides in vector operations: dot products, norms, etc.
- SUNDIALS-provided vector implementations may be used directly or as templates for problem-specific modules.

Right-hand side function routine(s): $f^E(t,y)$ & $f^I(t,y)$

Users must provide routine(s) to define the ODE based on their vector structure:

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

Where

- `t` – the current value of the independent variable.
- `y` – the current value of the dependent variable vector, $y(t)$.
- `ydot` – the output vector that forms a portion of the ODE right-hand side, $f^E(t,y) + f^I(t,y)$.
- `user_data` – "black box" pointer allowing users to pass data through ARKode without global variables.

For a purely explicit or implicit problem, only one `ARKRhsFn` need be supplied (the other should be `NULL`)

Nonlinear solver object (optional): $G_i(z) = 0$

As of our next release, users can supply their own application-specific "`SUNNonlinearSolver`" module underneath most SUNDIALS packages (including ARKode):

- *Content* structure stores all solver-specific data for performing the solve.
- Implementations of nonlinear solver *operations* on the supplied structure.
- These may leverage vector and linear solver APIs.
- SUNDIALS-provided nonlinear solver implementations may be used directly or as templates for application-specific solvers.

Matrix & Linear solver objects (optional)

If applicable in the nonlinear solve, users can supply an application-specific "`SUNLinearSolver`" module (and associated "`SUNMatrix`" module) underneath any SUNDIALS package:

- *Content* structure stores all solver-specific data for storing the matrix or performing the solve.
- Implementations of matrix or linear solver *operations* on the supplied structure.
- These may leverage vector API.
- SUNDIALS-provided parallel linear solver implementations may be used directly or as templates.
- No matrix objects are required when using the supplied parallel linear solvers, but may be used with application-specific linear solver modules.

Preconditioner routines (optional; necessary for scalability)

When using SUNDIALS-provided parallel linear solvers, users should supply their own application-specific preconditioner for scalability:

- Linear system $Ax = b$ is re-cast to an equivalent problem:

$$\begin{aligned} \left(P^{-1}A\right) x &= \left(P^{-1}b\right) && \text{[left]}, \\ \left(AP^{-1}\right) (Px) &= b && \text{[right]}, \\ \left(P^{-1}AP^{-1}\right) (Px) &= \left(P^{-1}b\right) && \text{[both]}. \end{aligned}$$

- If the preconditioned matrix is $\sim I$, then iterations converge rapidly and independently of problem size.
- Preconditioner 'setup' is performed infrequently, to amortize costs of preconditioner construction.
- Preconditioenr 'solve' is performed repeatedly, and should be efficient.

Outline

Multirate Infinitesimal Step Methods [Knoth & Wolke 1998; Schlegel et al. 2009; . . .]

MIS/RFSMR is a highly efficient, up to $\mathcal{O}(h^3)$ method used in numerical weather prediction. We consider 2-rate problems in additive form,

$$y'(t) = f^{\{f\}}(t,y) + f^{\{s\}}(t,y), \quad t \in [t_0, t_f], \quad y(t_0) = y_0 \in \mathbb{R}^n,$$

- $f^{\{f\}}(t,y)$ contains the "fast" terms,
- $f^{\{s\}}(t,y)$ contains the "slow" terms,
- the "slow" and "fast" time scales are separated by a factor m,
- y may optionally be partitioned as well, e.g. $y = \left[y^{\{f\}}\ y^{\{s\}}\right]^{\mathsf{T}}$

The MIS derivation assumes:

- the slow component is integrated using an explicit "outer" RK method, $T_o = \{A^o, b^o, c^o\}$, where $c^o_j \le c^o_{j+1}$, $j = 1, \ldots, s^o - 1$.
- 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" RK method (any type) with table T_i.

MIS Algorithm

We denote the slow stages as $\left\{ z_j^{\{s\}} \right\}_{j=1}^{s^o}$. Then a single MIS step of size h is:

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

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

Solve $v_j' = f^{\{f\}}(t, v_j) + \sum_{k=1}^{j-1} \frac{a^o_{j,k} - a^o_{j-1,k}}{c^o_j - c^o_{j-1}} f^{\{s\}}\left(t_n + c^o_k h, z_k^{\{s\}}\right)$,

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

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

Solve $v' = f^{\{f\}}(t, v) + \sum_{k=1}^{s^o} \frac{b^o_k - a^o_{s^o,k}}{1 - c^o_{s^o}} f^{\{s\}}\left(t_n + c^o_k h, z_k^{\{s\}}\right)$,

where $t \in [t_n + c^o_{s^o} h, t_n + h]$, with $v(t_n + c^o_{s^o} h) = z_{s^o}^{\{s\}}$.

Set $y_{n+1} = v(t_n + h)$.

MIS Method Properties

MIS methods satisfy a number of desirable multirate method properties:

- If both T_o and T_i are at least $\mathcal{O}\left(h^2\right)$ then the MIS method is $\mathcal{O}\left(h^2\right)$.
- If both T_o and T_i are at least $\mathcal{O}\left(h^3\right)$, and T_o satisfies

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

  then the MIS method is $\mathcal{O}\left(h^3\right)$.
- 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 Butcher tableau (SSP, symplectic, ...).
- m can be varied between steps to adapt with problem rate structure.
- Highly efficient – only a single traversal of $[t_n, t_n + h]$ is required to obtain y_{n+1}. In previous tests, we could not find a more efficient method.

Relaxed Multirate Infinitesimal Step Methods (RMIS) [Sexton & Reynolds 2018]

The RMIS algorithm is nearly identical to MIS, only changing how the fast stages contribute to the time-evolved solution:

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

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

Solve $v_j' = f^{\{f\}}(t, v_j) + \sum_{k=1}^{j-1} \frac{a^o_{j,k} - a^o_{j-1,k}}{c^o_j - c^o_{j-1}} f^{\{s\}}\left(t_n + c^o_k h, z_k^{\{s\}}\right)$,

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

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

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

RMIS Method Properties

RMIS methods inherit properties of MIS, with minor changes:

- The first stage of T_i must be explicit (but the rest can be anything).
- If both T_o and T_i are at least $\mathcal{O}\left(h^3\right)$ then the RMIS method is $\mathcal{O}\left(h^3\right)$.
- If T_i is at least $\mathcal{O}\left(h^3\right)$, and if T_o is $\mathcal{O}\left(h^4\right)$ and satisfies

$$v^{o\intercal} A^o c^o = \frac{1}{12}, \tag{2}$$

where

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

then the RMIS method is $\mathcal{O}\left(h^4\right)$.

- MIS can be used as an $\mathcal{O}\left(h^3\right)$ embedding within the $\mathcal{O}\left(h^4\right)$ RMIS method.

Outline

Conclusions

ARKode's ImEx infrastructure strives for application flexibility and algorithmic experimentation:

- Numerous built-in RK methods, support for user-supplied.
- Numerous built-in solver algorithms, support for user-supplied.
- Support for problem-specific simplifications (linearly implicit, etc.)
- Fully supports spatial and temporal adaptivity

Limitations:

- Many multiphysics applications involve more than 2 components
- All ARK components utilize the same step size h_n
- Splittings must be user-defined; cannot be chosen automatically

Conclusions (continued)

ARKode's upcoming algorithms will enable high-order & stable multirate integration without sacrificing efficiency:

- $\mathcal{O}(h^3)$ and $\mathcal{O}(h^4)$ methods with a single traversal of time interval $[t_n, t_n + h]$.
- Initial versions will assume only 2 rates, both evolved explicitly with user-defined step size h.
- Following versions will support implicit methods for fast time scale, and automated step size (h) and multirate (m) adaptivity.
- Exploring extensions to n-rates, where implicitness is confined to only the fastest time scale, through exploiting *telescopic* property.

Limitations:

- No current support for implicitness at the slow time scale.
- Fast/slow splittings must be user-defined (not automatic).

Thanks & Acknowledgements

Collaborators/Students:

- Carol S. Woodward [LLNL]
- David J. Gardner [LLNL]
- John Loffeld [LLNL]
- Rujeko Chinomona [SMU, PhD]
- Vu Thai Luan [SMU, postdoc]

Current Grant/Computing Support:

- DOE SciDAC & ECP Programs
- 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.

SMU FASTMATH Lawrence Livermore National Laboratory

References

- Ropp, Shadid & Ober, *J. Comput. Phys.*, 203, 2005.
- Estep et al., *Comput. Meth. Appl. Mech. Eng.*, 196, 2007.
- Kværnø & Rentrop, *Preprint 99/1. Univ. Karlsruhe*, 1999.
- Engstler & Lubich, *Appl. Numer. Math.*, 1997.
- Constantinescu & Sandu, *J. Sci. Comput.*, 2013.
- Ascher et al., *Applied Numerical Mathematics*, 25, 1997.
- Araújo et al., *SIAM J. Numer. Anal.*, 34, 1997.
- Knoth & Wolke, *Appl. Numer. Math.*, 1998.
- Schlegel et al., *J. Comput. Appl. Math.*, 2009.
- Sexton & Reynolds, *in preparation*, 2018.

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):

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

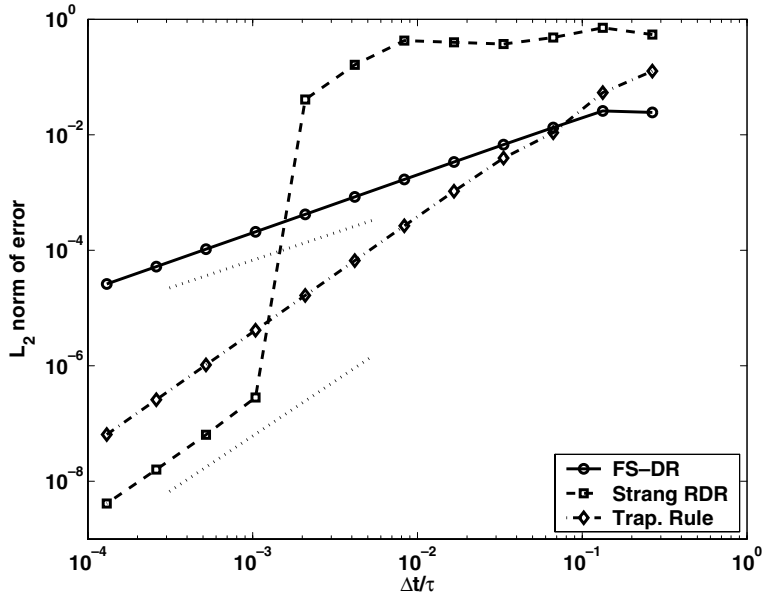
Three solvers:

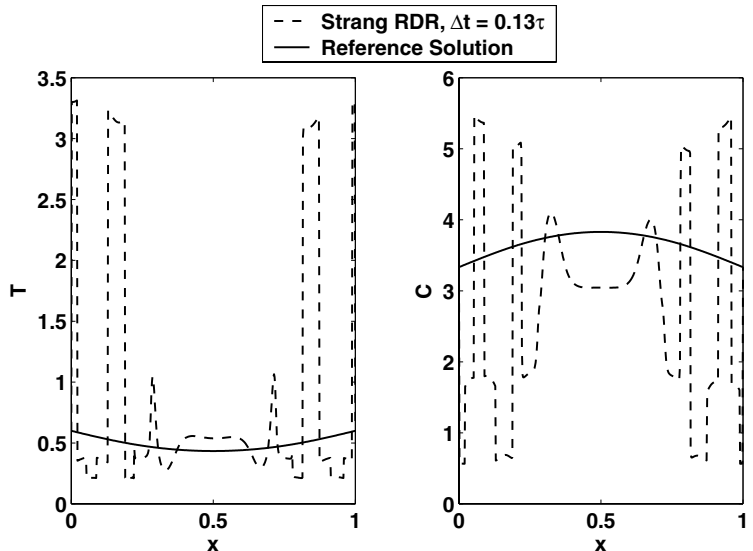
(a) Basic split: D (trap.) then R (subcycled BDF).

(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".





SMU

FASTMATH

Operator Splitting Issues – Accuracy [Estep 2007]

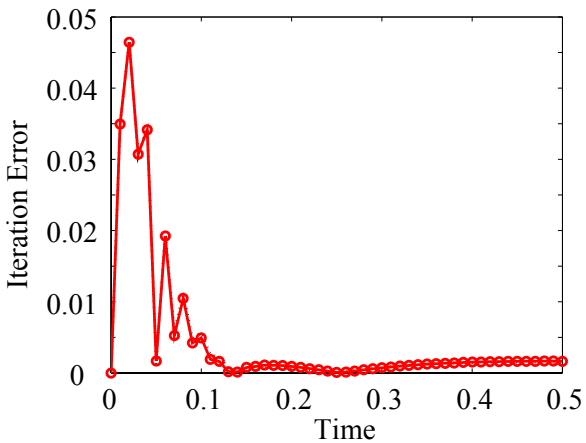
Consider $\Omega = \Omega_1 \cup \Omega_2$ where the subdomains share a boundary $\Gamma = \partial\Omega_1 \cap \partial\Omega_2$:

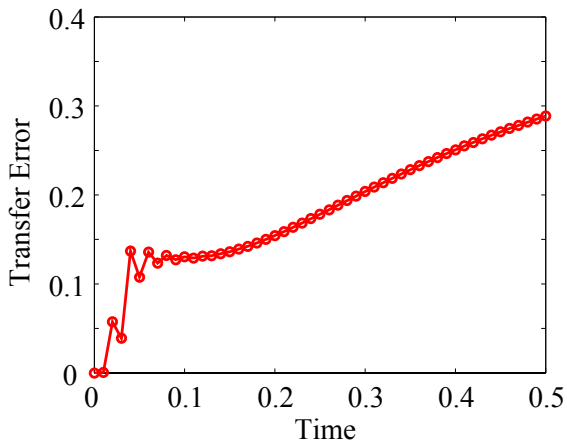
$$\partial_t u_1 = \nabla^2 u_1,\ x \in \Omega_1, \qquad \partial_t u_2 = \tfrac{1}{2}\nabla^2 u_2,\ x \in \Omega_2,$$

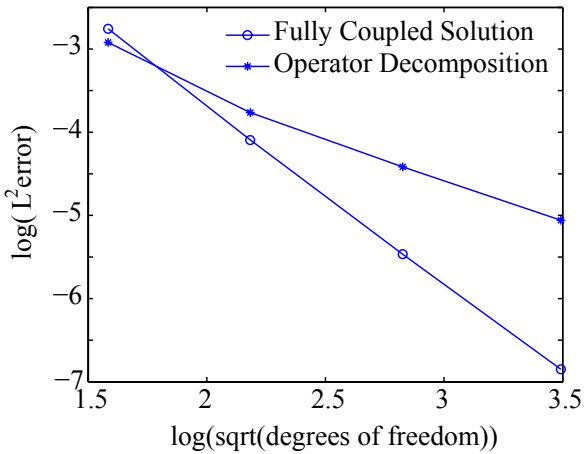
$$u_1 = u_2, \qquad \nabla u_1 \cdot n = \nabla u_2 \cdot n, \qquad \text{for } x \in \Gamma.$$

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):

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

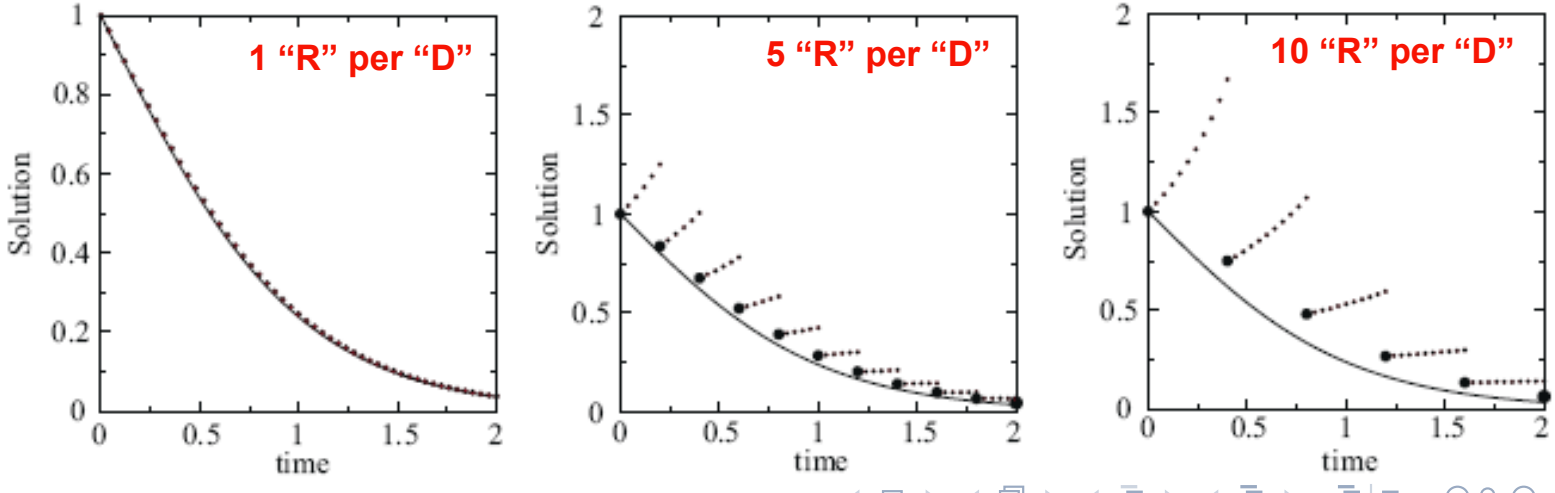
Phase 1 (R): $\partial_t u_r = u_r^2, \qquad u_r(t_n) = u_n, \qquad t \in [t_n, t_{n+1}]$,

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.



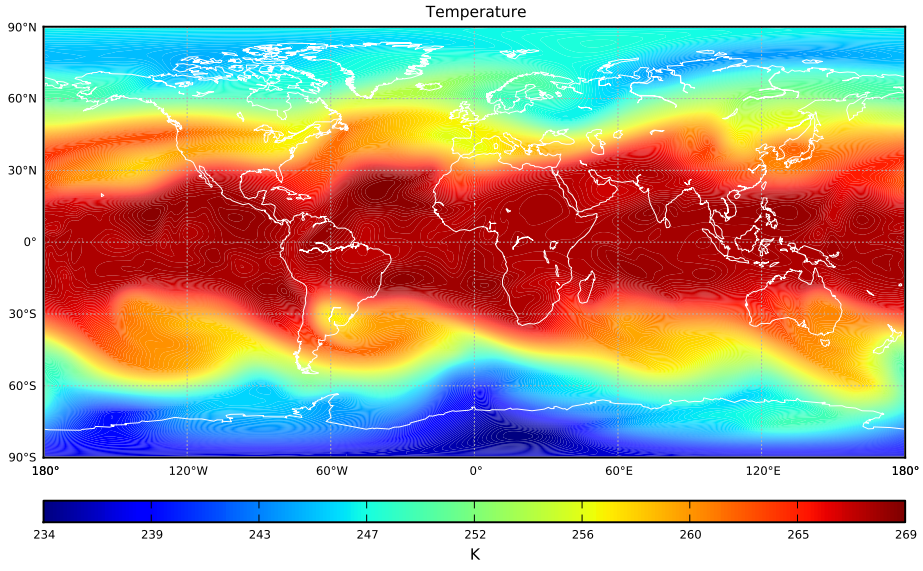
SMU

Outline

Tempest – non-hydrostatic 3D dynamical core for atmosphere

D.J. Gardner, J.E. Guerra, F.P. Hamon, D.R. Reynolds, P.A. Ullrich, C.S. Woodward

- Adaptive $\mathcal{O}(\Delta x^4)$ SEM horizontal grid
- Regular Staggered FEM vertical grid
- Parallelization of horizontal grid (each task owns multiple vertical columns)
- Project goals:
  - Examine ImEx splittings & ARK methods for accuracy/stability
  - Examine solution methods for implicit components



Baroclynic test @ 31 days (Temperature)

- Provide guidance for upcoming non-hydrostatic dycore for HOMME component of *Community Atmospheric Model* (CAM) and *Community Earth System Model* (CESM)

SMU FASTMATH Lawrence Livermore National Laboratory

The Tempest Model [Ullrich 2014]

Formulated in terms of horizontal velocities u_α and u_β, vertical velocity w, potential temperature θ, and density ρ in an arbitrary coordinate system (α, β, ξ):

$$\frac{\partial u_\alpha}{\partial t} = -\frac{\partial}{\partial \alpha}(K + \Phi) - \theta \frac{\partial \Pi}{\partial \alpha} + (\vec{\eta} \times \vec{u})_\alpha \tag{3}$$

$$\frac{\partial u_\beta}{\partial t} = -\frac{\partial}{\partial \beta}(K + \Phi) - \theta \frac{\partial \Pi}{\partial \beta} + (\vec{\eta} \times \vec{u})_\beta \tag{4}$$

$$\frac{\partial w}{\partial t} = \left(\frac{\partial \xi}{\partial r}\right)\left[-\frac{\partial}{\partial \xi}(K + \Phi) - \theta \frac{\partial \Pi}{\partial \xi} + (\vec{\eta} \times \vec{u})_\xi\right] \tag{5}$$

$$\frac{\partial \theta}{\partial t} = -u^\alpha \frac{\partial \theta}{\partial \alpha} - u^\beta \frac{\partial \theta}{\partial \beta} - u^\xi \frac{\partial \theta}{\partial \xi} \tag{6}$$

$$\frac{\partial \rho}{\partial t} = -\frac{1}{J}\frac{\partial}{\partial \alpha}(J \rho u^\alpha) - \frac{1}{J}\frac{\partial}{\partial \beta}\left(J \rho u^\beta\right) - \frac{1}{J}\frac{\partial}{\partial \xi}\left(J \rho u^\xi\right) \tag{7}$$

ImEx Splittings

Tested a variety of ImEx splittings:

- Horizontally explicit vertically implicit (implicitness relegated to decoupled columns):
  - HEVI–A: implicit treatment of all vertical dynamics except vertical advection of horizontal velocity,
  - HEVI–B: HEVI–A with explicit treatment of vertical velocity advection,
  - HEVI–C: HEVI–A with explicit treatment of thermodynamic advection
  - HEVI–D: HEVI–A with explicit treatment of vertical velocity and thermodynamic advection.
- More general ImEx: HEVI–A with additional implicit horizontal terms:
  - IMEX–A; implicit treatment of density equation,
  - IMEX–B: implicit treatment of density, thermodynamics, and Exner pressure.

Tempest + ARKode

Utilized a wide variety of ARKode features:

- Constructed a Tempest-specific vector data structure
- Examined standard "linearly implicit" approximation
- Swapped in a multitude of fixed step ARK methods:
  - $\mathcal{O}(h^2)$: ARK232 [Giraldo et al. 2013]; ARS222, ARS232 [Ascher et al. 1997]; SSP2(222), SSP2(332)a, SSP3(332), [Pareschi & Russo 2005]; SSP2(332)b [Higueras 2006]; SSP2(332)lpm1, SSP2(332)lpm2, SSP2(332)lpum, SSP2(332)lspum [Higueras et al. 2014]
  - $\mathcal{O}(h^3)$: ARK324 [Kennedy & Carpenter 2003]; ARS233, ARS343, ARS443 [Ascher et al. 1997]; SSP3(333) [Higueras 2009]; SSP3(433) [Pareschi & Russo 2005]
  - $\mathcal{O}(h^4)$ and $\mathcal{O}(h^5)$: ARK436, ARK548 [Kennedy & Carpenter 2003]
- Supplied communication-free column-wise banded linear solver for HEVI
- Utilized Jacobian-free GMRES linear (w/ column-wise preconditioner) for splittings with implicit horizontal components

Tempest Results – Gravity Wave Test

Test setup:

- Initially balanced atmosphere on a reduced radius Earth (1/125 in size)
- Small potential temperature perturbation induces gravity waves.
- One hour simulation duration.

Takeaway conclusions:

- Linearly implicit approx. valid to within modeling/discretization error
- Splitting stability $\propto$ implicitness: IMEX–B $>$ IMEX–A $>$ HEVI
- Inclusion of horizontally explicit terms increases runtime by 25% – 60%
- Most accurate: $\mathcal{O}(h^2)$ "SSP" methods by Higueras et al. 2014; $\mathcal{O}(h^3)$ SSP3(433) method
- Most stable: $\mathcal{O}(h^2)$ ARK232, ARS232 and SSP3(332); $\mathcal{O}(h^3)$ ARS343

Tempest Results – Baroclynic Wave Test

Test setup [Ullrich et al. 2014]:

- Simulates the development/propagation of a baroclinic wave
- 30 day simulation duration (wave develops in first 10)
- Compare solution quality using the largest stable h for each method

Takeaway conclusions:

- Nonlinearity becomes significant after 10 days (~ 2 Newton iters. required); linearized tests show nonphysical results at these h
- Splitting stability again improves with implicitness
- Most SSP methods show nonphysical vertical velocities for HEVI splittings; others unstable except at small h
- Best stability/accuracy from ARS343 and ARK324 methods
- Significant stability improvement from horizontally implicit with penalty of costlier solve; overall IMEX–A cost between HEVI–A/B and HEVI–C/D

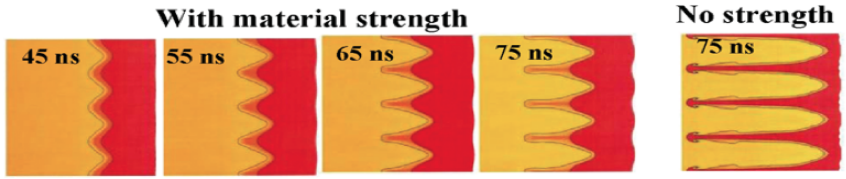
ParaDiS – Parallel Dislocation Dynamics Simulator

D.J. Gardner, C. S. Woodward, D.R. Reynolds, G. Hommes, S. Aubry, A.T. Arsenlis (2015)

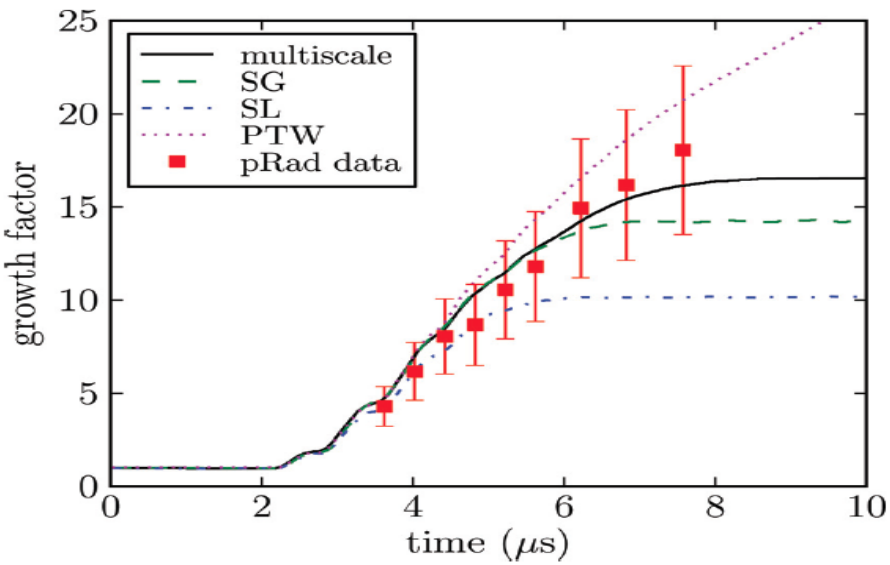
Modeling material strain hardening:

- 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)]

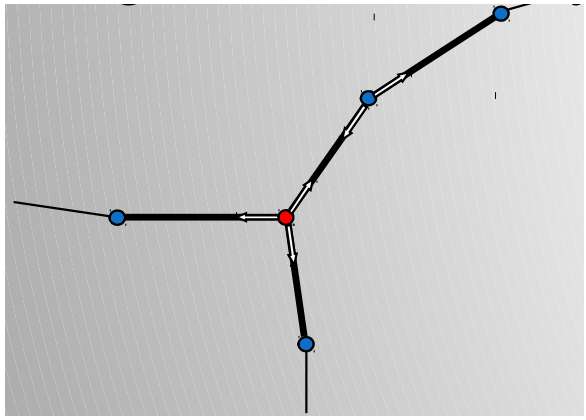
SMU

FASTMATH

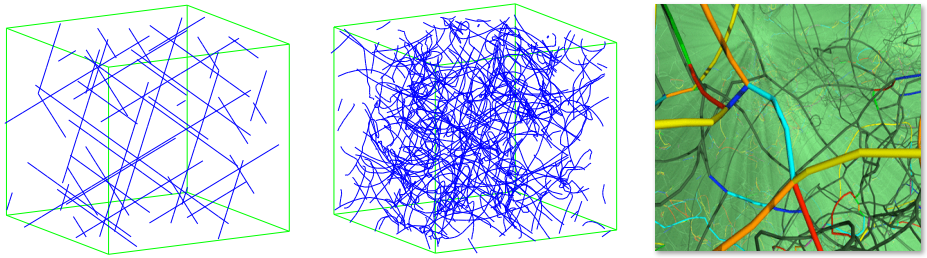
Lawrence Livermore National Laboratory

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:

$$F_i(t, \mathbf{x}) = F_i^{\text{self}}(\mathbf{x}) + F_i^{\text{seg}}(\mathbf{x}) + F_i^{\text{ext}}(t, \mathbf{x})$$

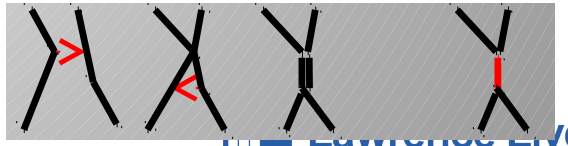
- Nodal velocity calculation (the mobility law M is material-dependent, nonlinear):

$$v_i(t, \mathbf{x}) = M(F_i(t, \mathbf{x}))$$

- Time integration (nodal positions):

$$x_i' = v_i(t, \mathbf{x})$$

- Topology changes (insert/merge nodes):



SMU

FASTMATH

Lawrence Livermore National Laboratory

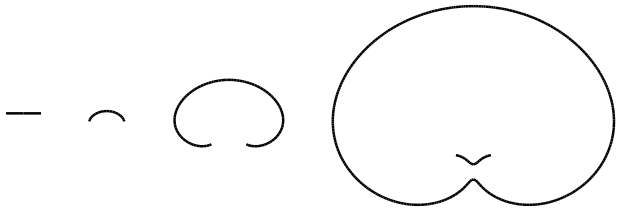
ParaDiS + ARKode

- Utilized built-in embedded DIRK integrators:
  - 3-stage, $\mathcal{O}(h^3)$ SDIRK [Billington 1983]
  - 5-stage, $\mathcal{O}(h^4)$ SDIRK [Hairer & Wanner 2010]
  - 7-stage (6 implicit), $\mathcal{O}(h^5)$ ESDIRK [Kværno 2004]
- Constructed a ParaDiS-specific vector data structure
- "Resized" the solver and vector data structure between each time step
- Utilized both matrix-free inexact Newton (w/ GMRES), and accelerated fixed-point nonlinear solvers
- Examined wide variety of implicit predictor methods

ParaDiS Results – Frank-Read Source

Simple test problem:

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



- Strain rate 1 s^{-1}; Final time 50 μs
- Comparisons ($\epsilon_n = 1$, $\epsilon_l = \frac{1}{2}$):
  - ParaDiS Trapezoid solver: basic fixed-point (2,3 iters)
  - KINSOL Trapezoid solver: AFP (2-4 iters)
  - DIRK, $\mathcal{O}(\Delta t^3) \rightarrow \mathcal{O}(\Delta t^5)$: NK and AFP (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	270	92.1
DIRK4 NK I4	140	95.3
DIRK5 NK I4	169	93.8
DIRK3 AA I4 V3	127	97.5
DIRK4 AA I4 V3	194	95.6
DIRK5 AA I4 V3	128	96.9

ParaDiS Results – Target Problem ("Warm-start Test")

"Real" problem, mid-simulation:

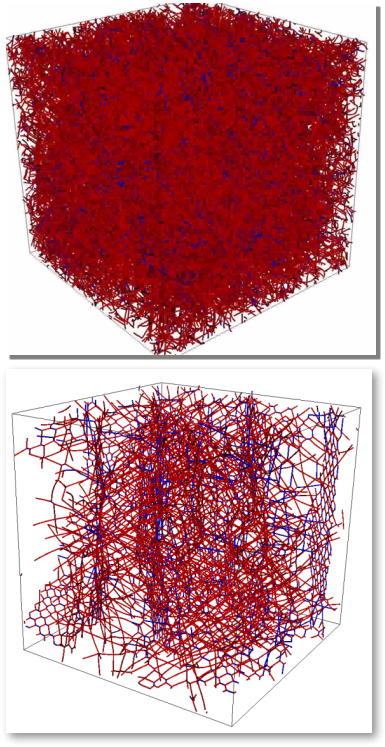
- Body-centered-cubic crystal structure, $\Omega = 4.25\ \mu\text{m}^3$
- Strain rate $10^2\ \text{s}^{-1}$
- $3.3\ \mu\text{s} \leq t \leq 4.4\ \mu\text{s}$
- ~ 2850 initial nodes, ~ 5000 final
- Comparison between:
  - Native Trapezoid solver: basic fixed-point (2 iters)
  - KINSOL Trapezoid solver: AA (2-6 iters)
  - DIRK $\mathcal{O}(\Delta t^3)$ solver: AA (2-6 iters), $\varepsilon_n = 1$

Method	Steps	% Speedup
Trap FP I2	2267	0.0
Trap AA I4 V3	809	44.6
Trap AA I5 V4	715	45.7
Trap AA I6 V5	624	50.0
Trap AA I7 V6	568	51.9
DIRK3 AA I4 V3	116	56.2
DIRK3 AA I5 V4	124	50.5
DIRK3 AA I6 V5	123	51.6
DIRK3 AA I7 V6	129	46.5
DIRK5 AA I4 V3	118	25.0
DIRK5 AA I5 V4	105	28.0
DIRK5 AA I6 V5	104	32.3
DIRK5 AA I7 V6	105	25.0

ParaDiS – Conclusions

- Limited differentiability of $F_i(t, \mathbf{x})$ calculations seemingly capped utility of high-order methods
- Adaptive methods ultimately limited by $h_{\max}$ – bound due to parallelism constraints (neighborhoods in FMM data structure)
- Lack of analytical Jacobian (or preconditioner) hindered inexact Newton performance (Jv product required FMM calculation at each iteration)

Pretty pictures (annealing):



References

- Ullrich et al., *Quarterly J. Royal Meteor. Soc.*, 140, 2014.
- Giraldo et al., *SIAM J. Sci. Comput.*, 35, 2013.
- Pareschi & Russo, *J. Sci. Comput.*, 25, 2005.
- Higueras, *SIAM J. Numer. Anal.*, 44, 2006.
- Higueras et al., *J. Comput. Appl. Math.*, 272, 2014.
- Kennedy & Carpenter, *Appl. Numer. Math.*, 44, 2003.
- Higueras, *J. Sci. Comput.*, 39, 2009.
- Park et al., *PRL*, 104, 2010.
- Barton et al., *J. App. Phys.*, 109, 2011.
- Billington, *PhD Thesis, University of Manchester*, 1983.
- Hairer & Wanner, Solving Ordinary Differential Equations II, Springer, 2010.
- Kværno, *BIT Numer. Math.*, 44, 2004.
- Gardner et al., *Model. Simul. Mater. Sci. Eng.*, 23, 2015.