Parallel Multigrid Reduction in Time – Theory, Practice, and Applications

The forty-third Woudschoten Conference
Zeist, The Netherlands

CASC
Center for Applied Scientific Computing

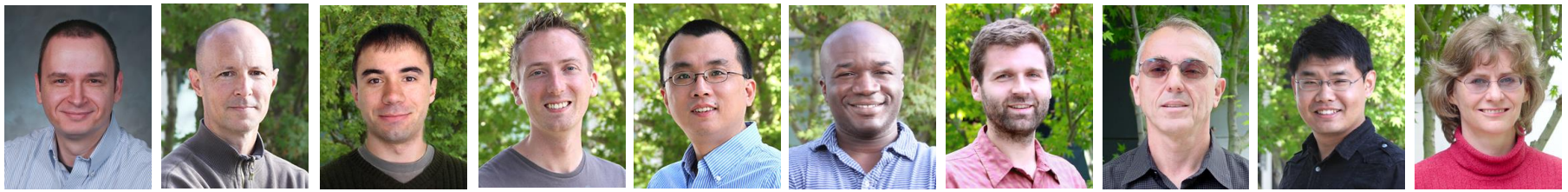
Robert D. Falgout
Center for Applied Scientific Computing

October 3 - 5, 2018

LLNL-PRES-759163
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Our Multigrid and Parallel Time Integration Research Team



Veselin Dobrev | Rob Falgout | Tzanio Kolev | Matthieu Lecouvez | Ruipeng Li | Daniel Osei-Kuffuor | Jacob Schroder | Panayot Vassilevski | Lu Wang | Ulrike Yang

- University collaborators and summer interns
  - CU Boulder (Manteuffel, McCormick, Ruge, O'Neill, Mitchell, Southworth), Penn State (Brannick, Xu, Zikatanov), UCSD (Bank), Ball State (Livshits), U Wuppertal (Friedhoff, Kahl), Memorial University (MacLachlan), U Illinois (Gropp, Olson, Bienz), U Stuttgart (Röhrle, Hessenthaler), Monash U (De Sterck), TU Kaiserslautern (Günther)

- Software, publications, and other information

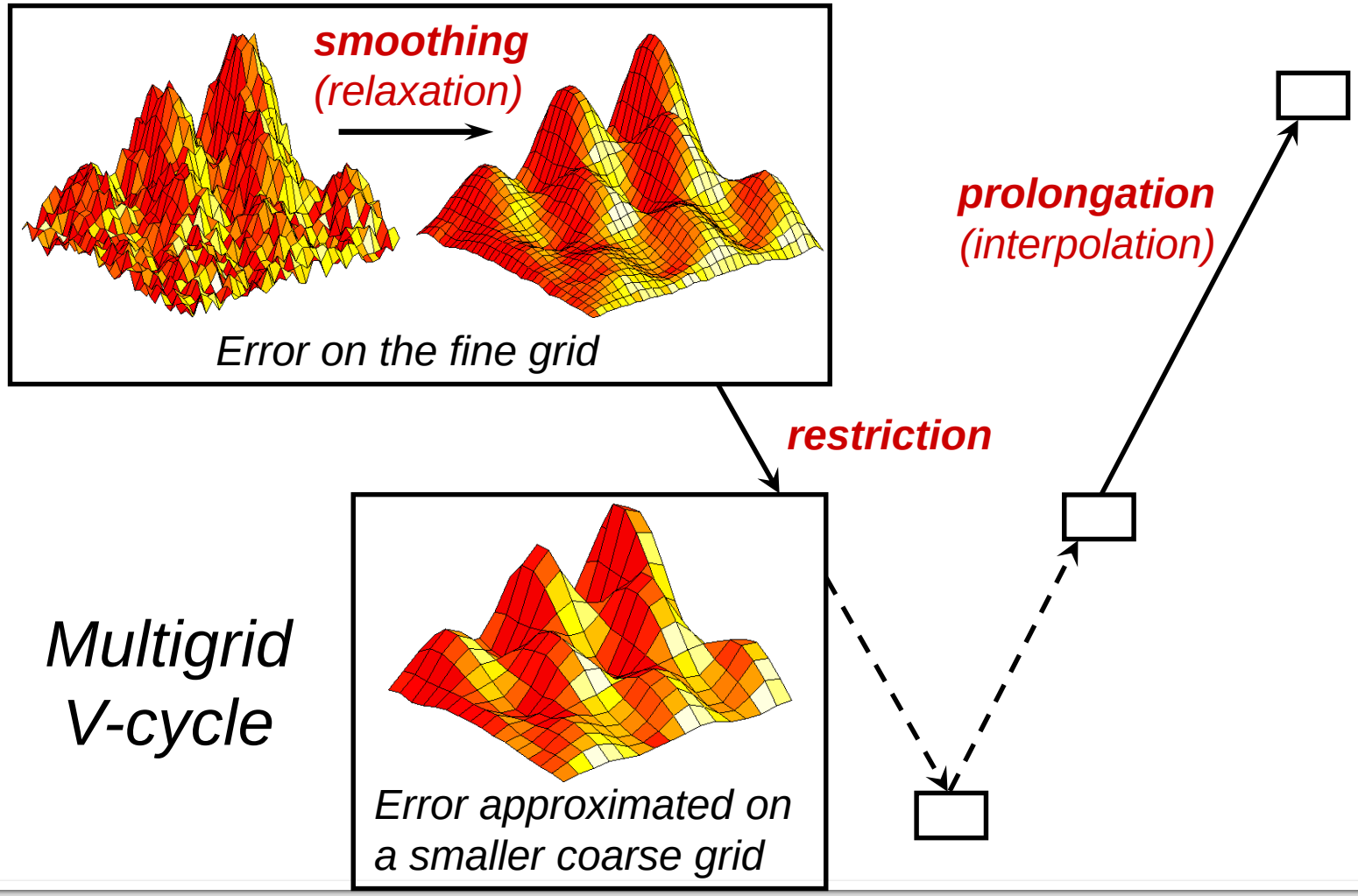
hypre http://llnl.gov/casc/hypre

XBRAID http://llnl.gov/casc/xbraid

Outline

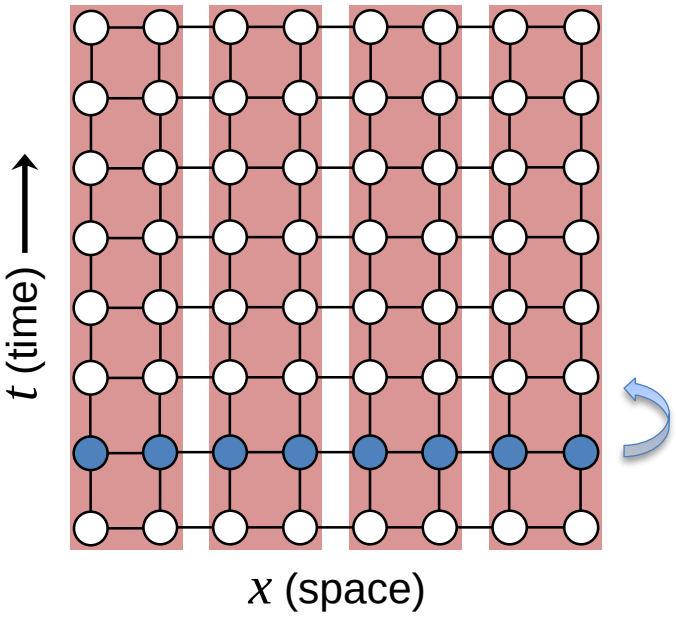
- Basic MGRIT algorithm, philosophy, and properties
- Software
- Space-time adaptivity
- Multistage and multistep methods
  - Power grid simulations
- Hyperbolic problems
- Theory
- Richardson extrapolation
- New optimization feature in XBraid
- Summary and conclusions

Our approach for parallel-in-time: leverage spatial multigrid research and experience

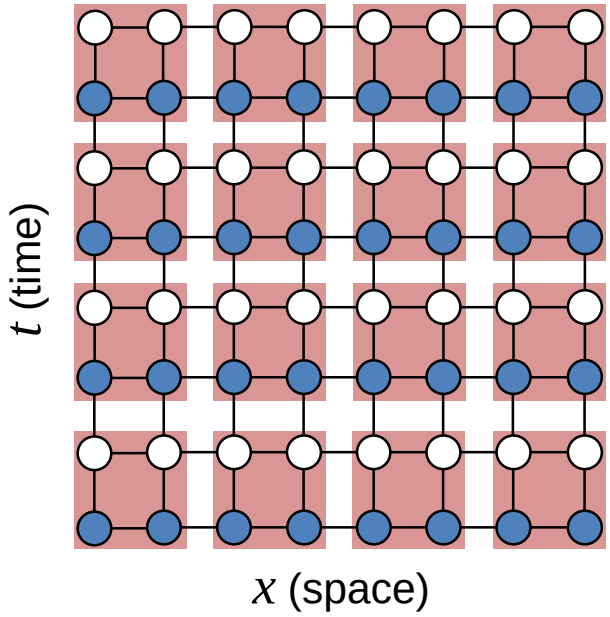


Significantly more parallel resources can be exploited with multigrid in time

Serial time stepping



Multigrid in time



- Parallelize in space only
- Store only one time step
- Parallelize in space and time
- Store several time steps

It's useful to view the time integration problem as a large block matrix system

- General one-step method

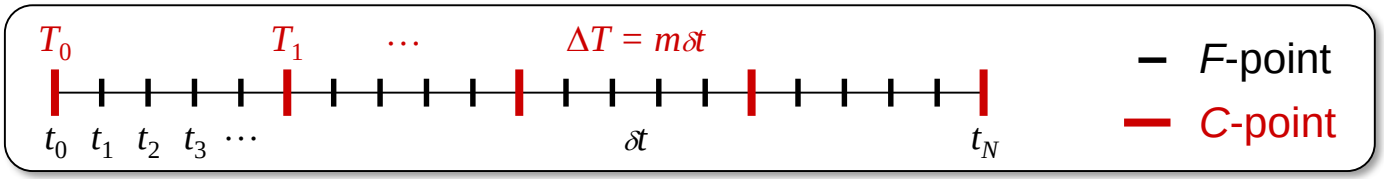
$$\boldsymbol{u}_i = \Phi_i(\boldsymbol{u}_{i-1}) + \boldsymbol{g}_i, \quad i = 1, 2, ..., N$$

- Linear setting: time marching = block forward solve
  - $O(N)$ direct method, but sequential

$$A\mathbf{u} \equiv \begin{pmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_N \end{pmatrix} = \begin{pmatrix} \boldsymbol{g}_0 \\ \boldsymbol{g}_1 \\ \vdots \\ \boldsymbol{g}_N \end{pmatrix} \equiv \mathbf{g}$$

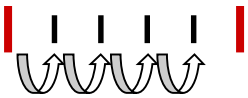
- Our approach is based on multigrid reduction (MGR) methods (approximate cyclic reduction)
  - $O(N)$ iterative method, but highly parallel

MGR dates to 1979 (Ries & Trottenberg) and we are extending it "in time" (MGRIT)



- Relaxation alternates between *F* / *C*-points
  - F-relaxation = integration over coarse intervals
- Coarse system is a time re-discretization
  - Replaces the exact Petrov-Galerkin system
- Non-intrusive approach
  - Time discretization is unchanged
  - User only provides time integrator Φ

F-relaxation



Coarse Petrov-Galerkin system is not practical → approximate it

$$A_\Delta \mathbf{u}_\Delta = \mathbf{g}_\Delta \equiv R_\Phi \mathbf{g}$$

$$A_\Delta = \begin{pmatrix} I & & & \\ -\Phi^m & I & & \\ & \ddots & \ddots & \\ & & -\Phi^m & I \end{pmatrix}$$

Our MGRIT approach builds as much as possible on existing codes and technologies

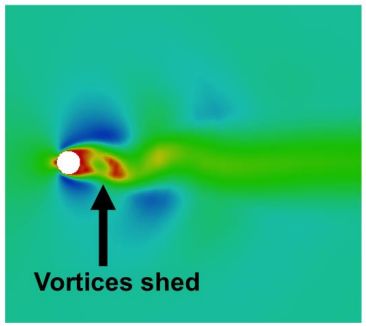
- Combines algorithm development, theory, and software proof-of-principle

- Goal: Create concurrency in the time dimension
- Non-intrusive, with unchanged time discretization
  - Implicit, explicit, multistep, multistage, …
- Converges to same solution as sequential time stepping
- Extends to nonlinear problems with FAS formulation

$$\begin{pmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{pmatrix}$$

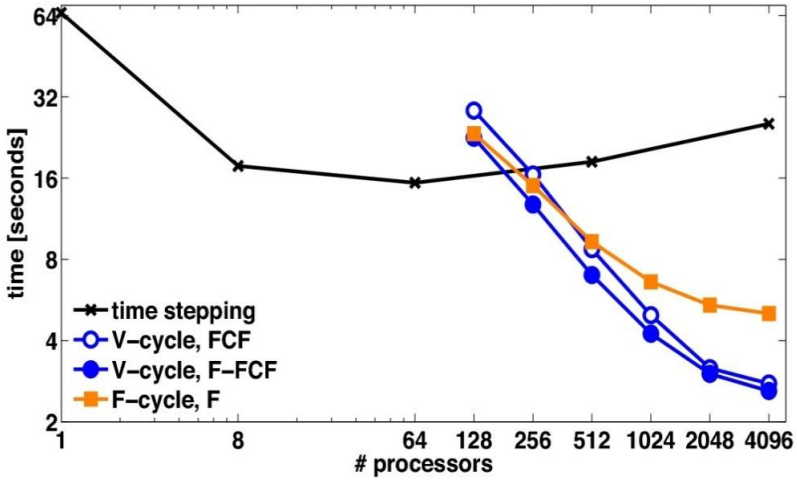
- XBraid is our open source implementation of MGRIT
  - User defines two objects and writes several wrapper routines (Step)
  - Only stores *C*-points to minimize storage

- Many active research topics, applications, and codes
  - Adaptivity in space and time, moving meshes, BDF methods, …
  - Linear/nonlinear diffusion, advection, fluids, power grid, elasticity, …
  - MFEM, hypre, Strand2D, Cart3D, LifeV, CHeart, GridDyn



Parallel speedups can be significant, but in an unconventional way

- Parallel time integration is driven entirely by hardware
  - Time stepping is already $O(N)$
- Useful only beyond some scale
  - There is a crossover point
  - Sometimes need significantly more parallelism just to break even
  - Achievable efficiency is determined by the space-time discretization and degree of intrusiveness



3D Heat Equation: 33^3 x 4097,
8 procs in space, 6x speedup

- The more time steps, the more speedup potential
  - Applications that require lots of time steps benefit first
  - Speedups (so far) up to 49x on 100K cores

Software

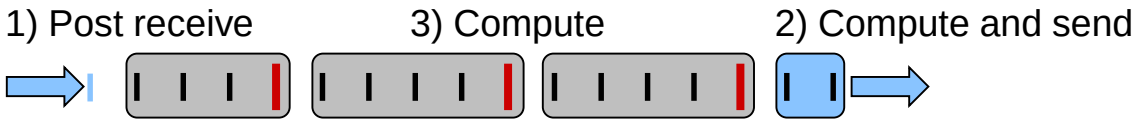
XBraid is open source and designed to be both non-intrusive and flexible

- User defines two objects:
  - *App* and *Vector*

- User also writes several wrapper routines:
  - *Step*, *Init*, *Clone*, *Sum*, *SpatialNorm*, *Access*, *BufPack*, *BufUnpack*
  - *Coarsen*, *Refine* (optional, for spatial coarsening)

- Example: *Step(app, u, status)*
  - Advances vector *u* from time *tstart* to *tstop* and returns a target refinement factor

- Code stores only C-points to minimize storage
  - Ability to coarsen by large factors means fewer parallel resources
  - Memory multiplier per processor: ~O(log N) with time coarsening, O(1) with space-time coarsening
  - Each proc starts with the right-most interval to overlap comm/comp

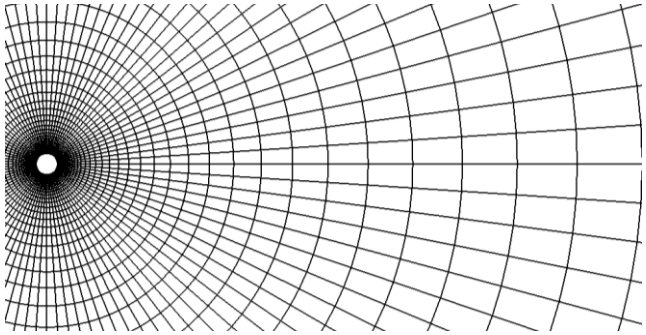


Experiments coupling XBraid with various application research codes

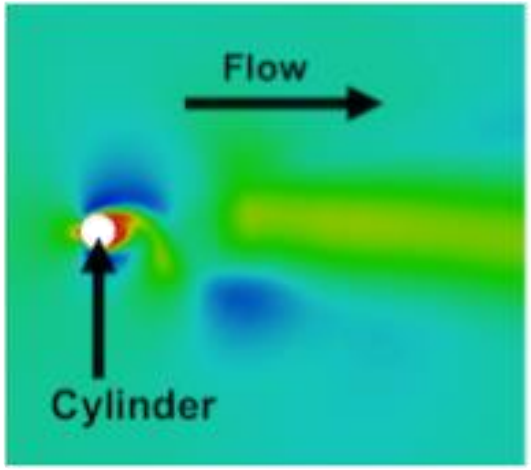
- Navier-Stokes (compressible and incompressible)
  - Strand2D, CarT3D, LifeV (Trilinos-based)
- Heat equation (including moving mesh example)
  - MFEM, hypre
- Nonlinear diffusion, the p-Laplacian
  - MFEM
- Power-grid simulations
  - GridDyn
- Explicit time-stepping coupled with space-time coarsening
  - Heat equation
  - Advection plus artificial dissipation
  - MFEM, hypre

Compressible Navier-Stokes (nonlinear) – up to 7.5x speedup, 4K cores, typical multigrid scaling

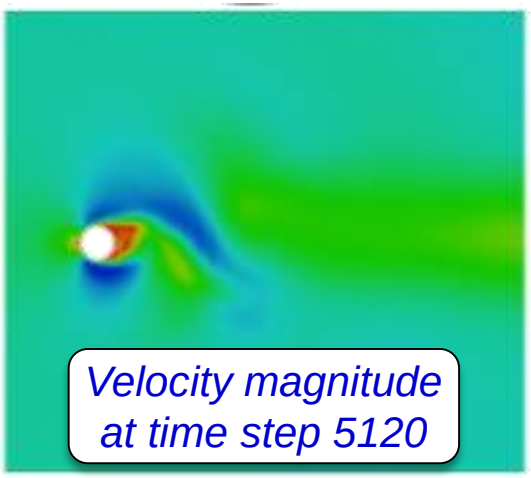
- Coupled XBraid with Strand-2D
  - ~ 500 lines of XBraid code to wrap 13,500 lines of Strand-2D code
  - ~ 3 weeks with minimal outside help



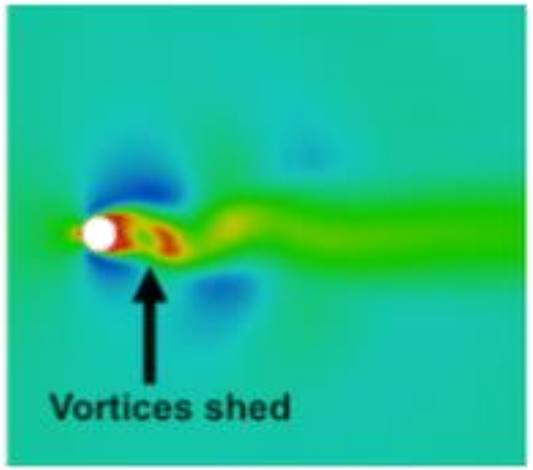
Iteration 1



Iteration 5



Iteration 13



Only a small amount of new code was required to couple Strand2D & XBraid

- Defined two objects and wrote a handful of short routines to wrap the existing 13,500 lines of code

Vector Object

```
18 // Wrapper for BRAID's Vector object
19 class BraidVector
20 {
21 public:
22    // Empty Constructor
23    BraidVector() {  }
24
25    // State vector
26    Array3D<double> vec;
27 };
```

App Object

```
30 class StrandBraidApp : public BraidApp
31 {
32 public:
33    string   in_file;
34    double   dt;
35    int      buff_size;
36    int      state_vec_size;
37    int      nSurfNode, nStrandNode, nq;
38    BraidVector *q_init;
39    Strand2dFCManager *manager;
```

Time Integration Routine (Phi / Step)

```
126   virtual int Phi(braid_Vector    _u,
127                   BraidPhiStatus  &pstatus)
128   {
129      BraidVector *u = (BraidVector*) _u;
130      double tstart, tstop, accuracy, t, dt;
131      int step;
132
133      // Initialize
134      pstatus.GetTstartTstop(&tstart, &tstop);
135      t = tstop;
136      dt = tstop-tstart;
137      step = round(t / dt);
138
139      // Reset Strand with new time step size and state vector
140      manager->w_resetQ(u->vec);
141      manager->w_resetDtUnsteady(dt);
142
143      // Carry out one time step (via many pseudo steps)
144      int  nPseudoSteps  = manager->getNPseudoSteps();
145      bool converged     = false;
146      for (int pseudoStep=0; pseudoStep<nPseudoSteps; pseudoStep++) {
147         manager->takePseudoStep(step, pseudoStep, converged);
148         if (converged)
149            break;
150      }
151
152      // Save state vector from Strand
153      manager->w_getQ(u->vec);
154      return 0;
155   }
```

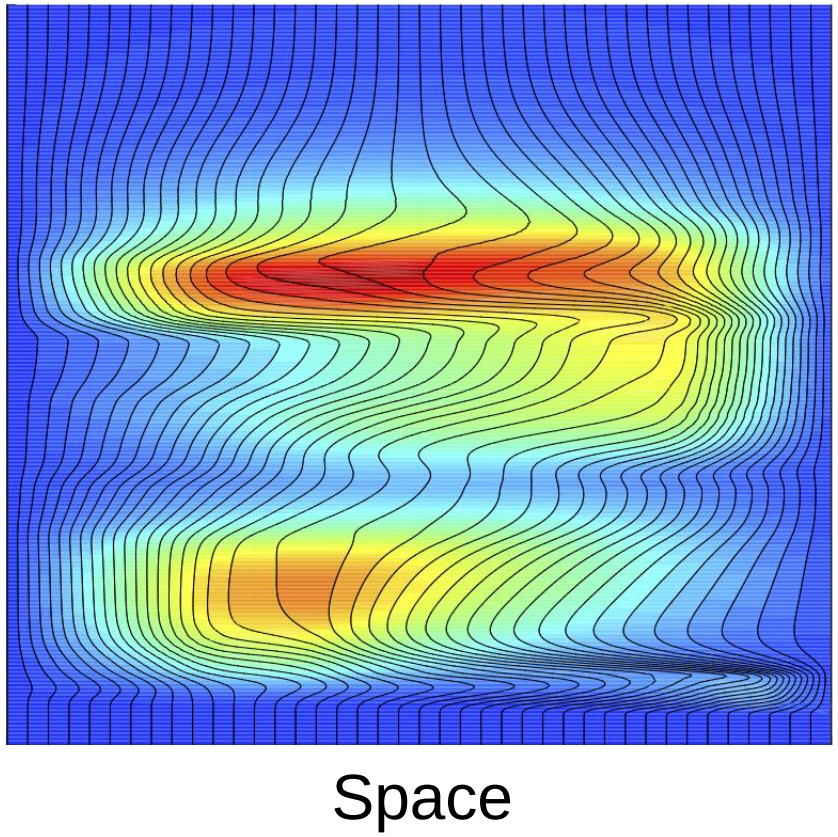
Space-Time Adaptivity

Exploring model problems to demonstrate adaptivity approach and develop capability in Braid

- Moving spatial mesh
  - 1D diffusion with time dependent source
  - Ben Southworth (CU Boulder)
- Temporal refinement
  - ODE simulation of satellite orbit
  - Matthieu Lecouvez (LLNL)
- Temporal and spatial refinement
  - 2D heat equation with FOSLS formulation
  - Ben O'Neill (CU Boulder)
- Current emphasis is on algorithmic development
  - Demonstrating parallel speedup will come later

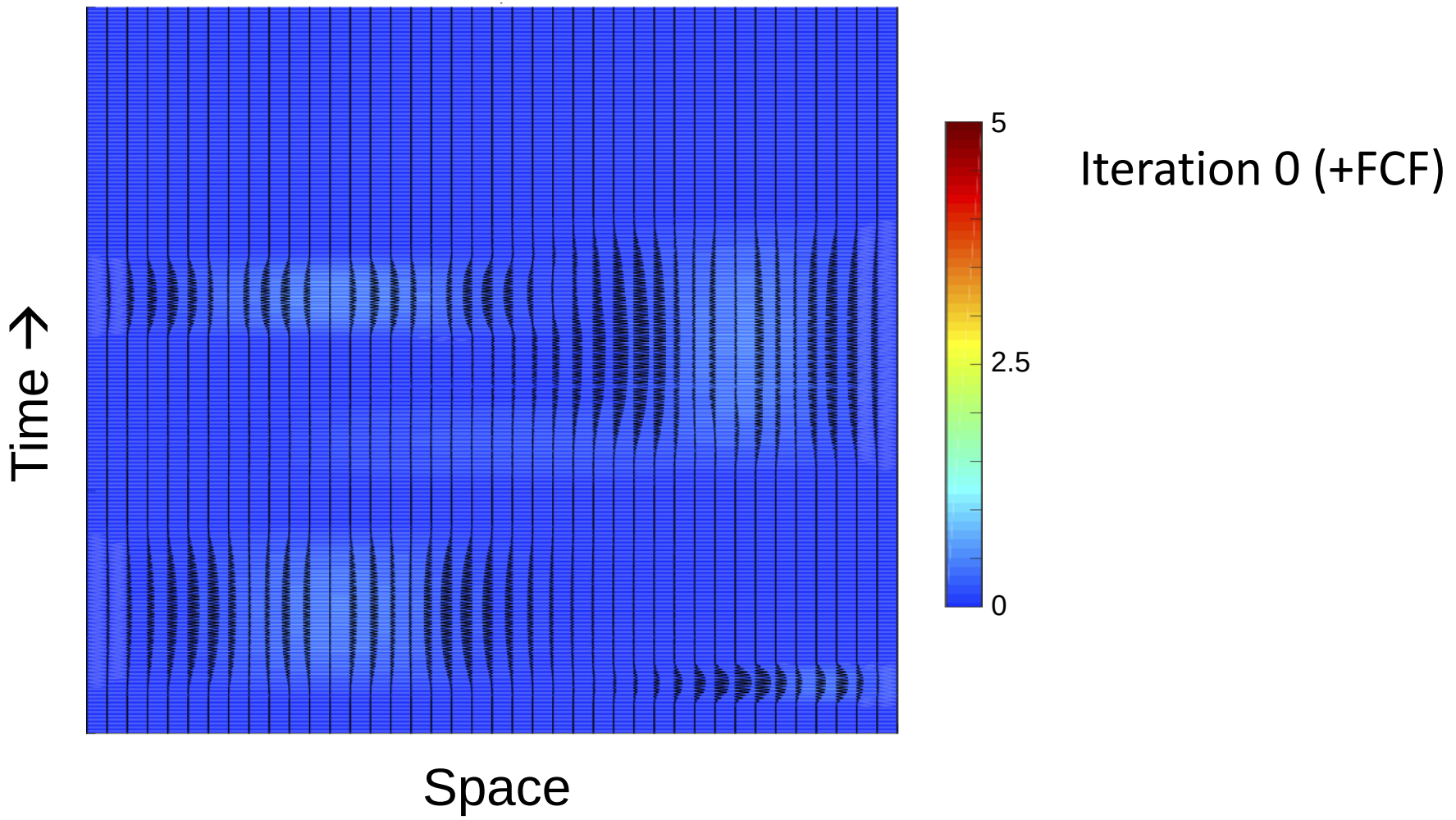
Moving mesh proof-of-concept – 1D space, nonlinear parabolic problem

- Mesh points move towards regions with rapidly changing solution, induced by time dependent heat sources

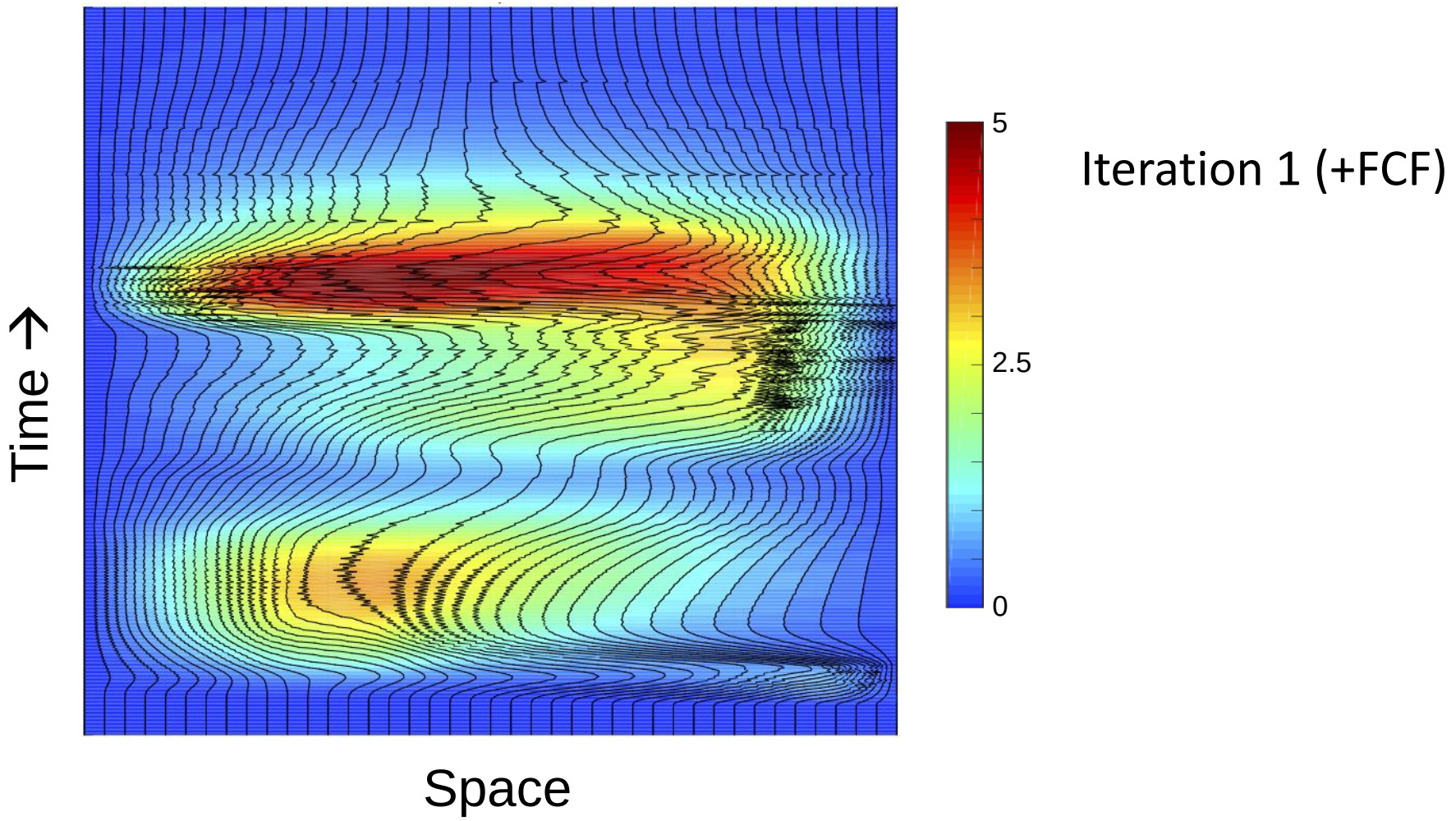


- Initial spatial mesh is uniform
- Time step:
  - Evolve the solution on the existing mesh
  - Move the mesh based on another PDE
  - Remap the solution to the new mesh

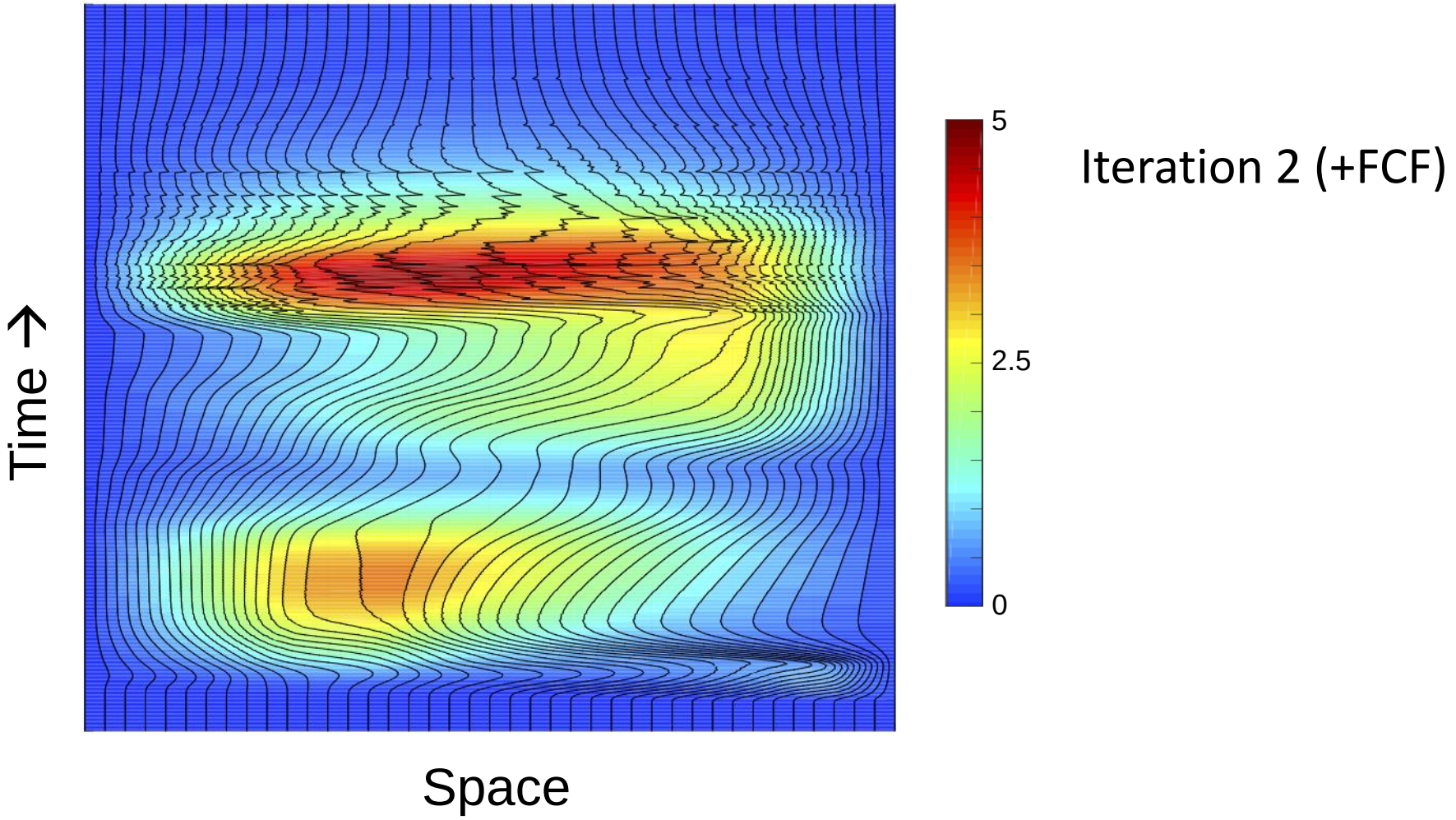
Moving mesh proof-of-concept – 1D space, nonlinear parabolic problem



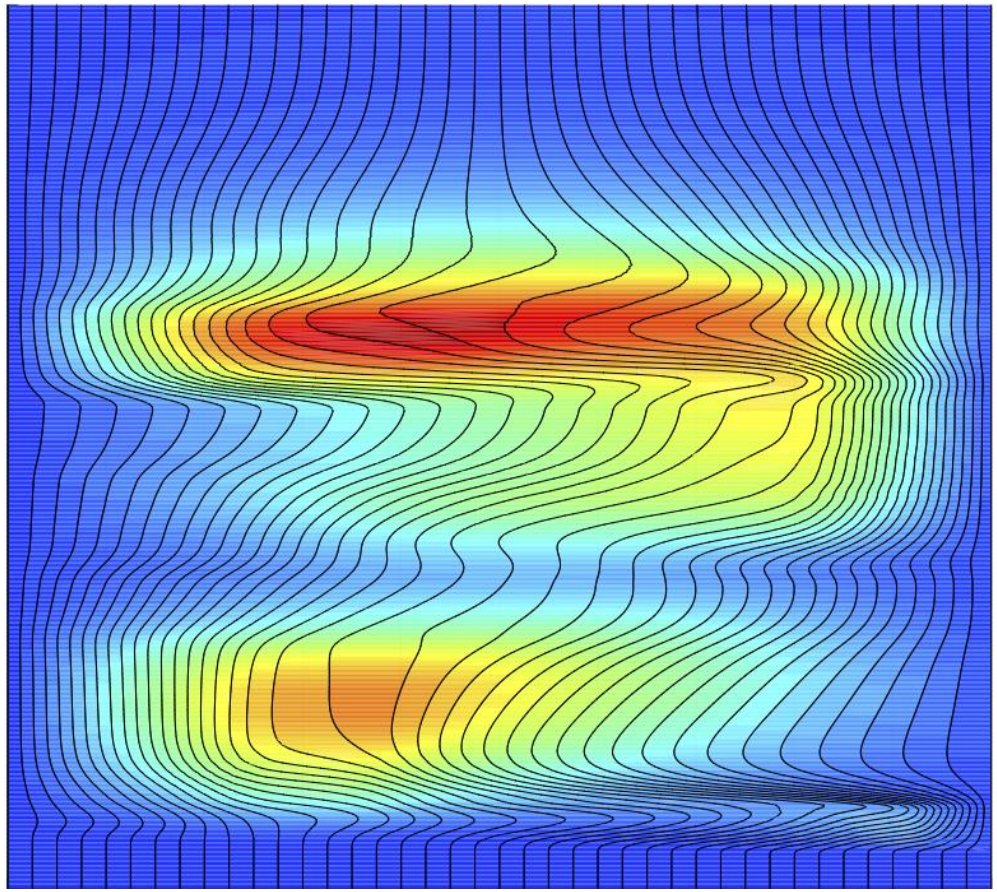
Moving mesh proof-of-concept – 1D space, nonlinear parabolic problem

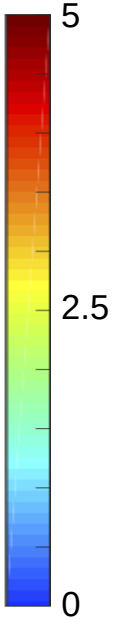


Moving mesh proof-of-concept – 1D space, nonlinear parabolic problem



Moving mesh proof-of-concept – 1D space, nonlinear parabolic problem





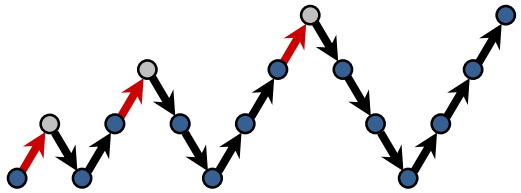
Iteration 4

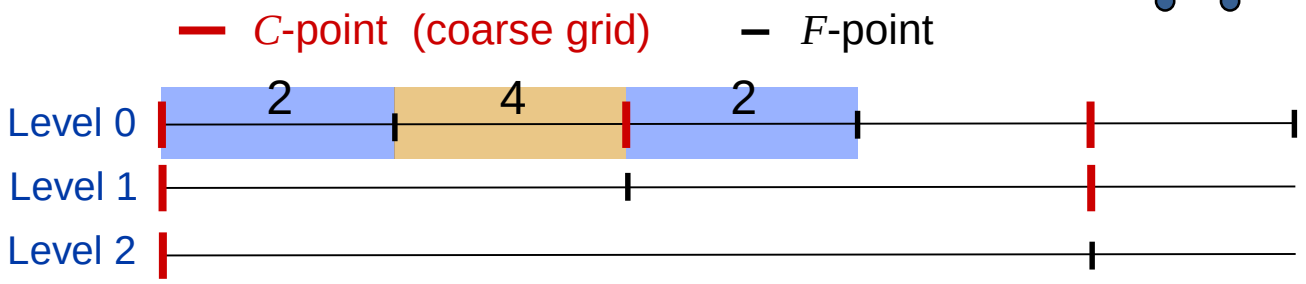
Fast and scalable convergence

Future: more complex moving mesh problems

Adaptivity in time/space via full multigrid (FMG)

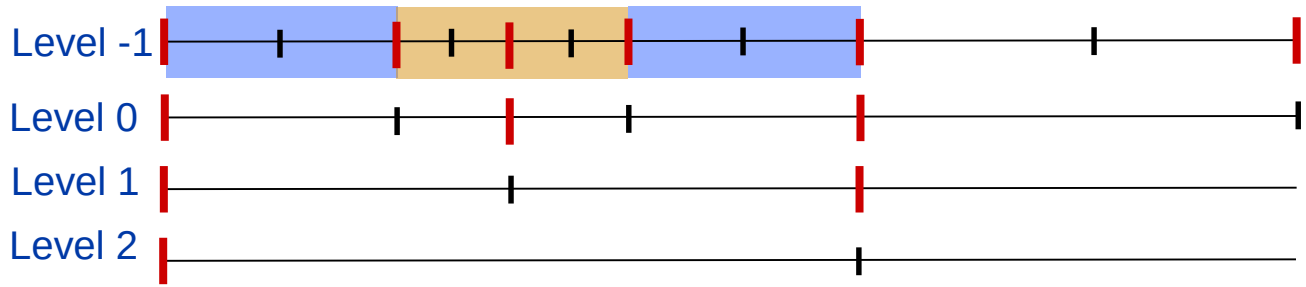
- User returns refinement factor in *Step()*
- Example time grid hierarchy





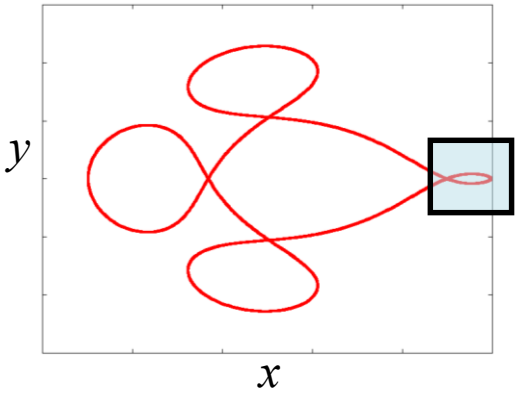
Notice new C-pts

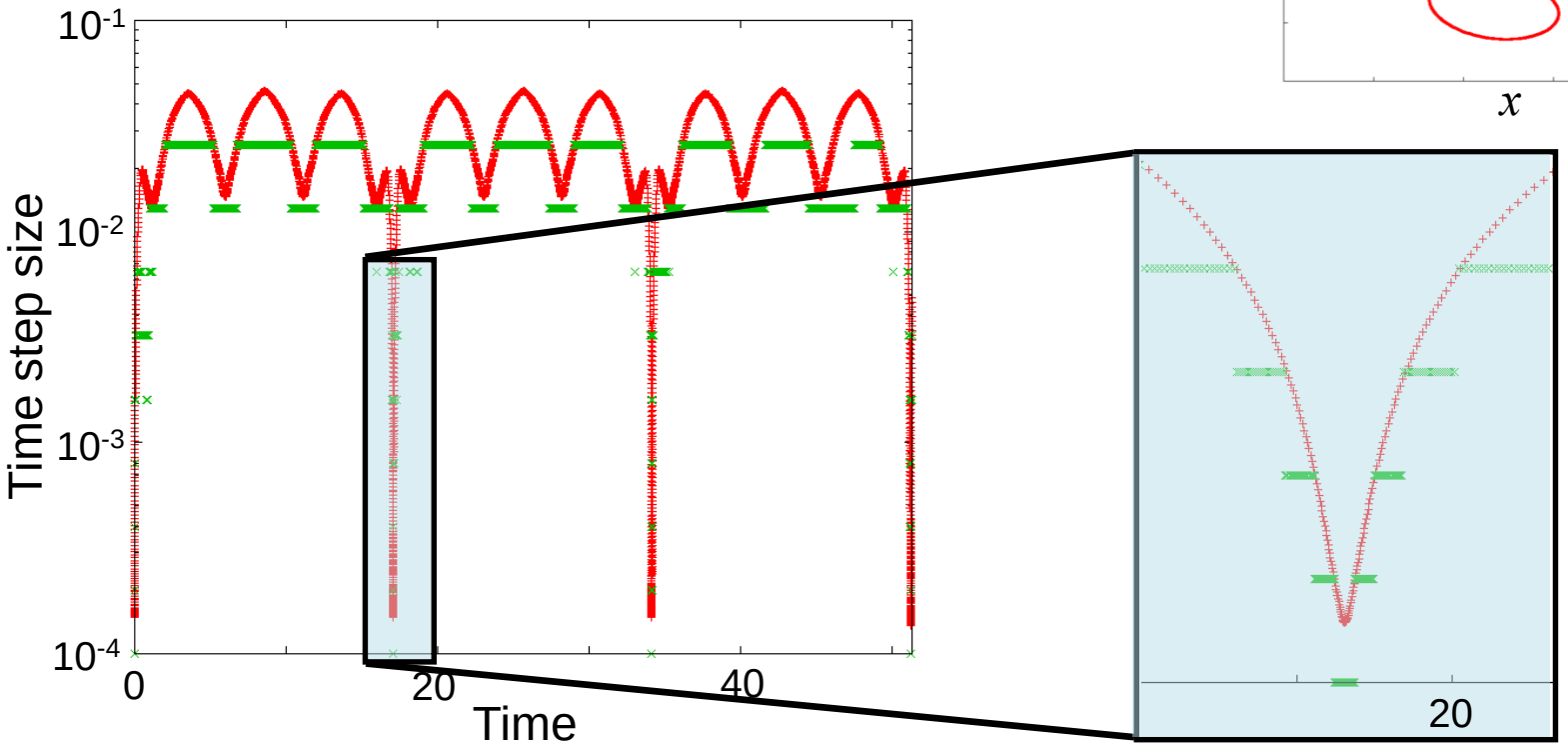
- User requests refinement factors on the finest grid which generates a new grid and hierarchy



Temporal adaptivity proof-of-concept: Classic ODE simulation of satellite orbit around earth and moon

- One region requires very fine time steps
  - 4 orbit periods, refining step size as needed
  - 2 variables in space (x, y)





Adaptivity in time and space: FOSLS formulation for linear heat equation

- 2D Linear Heat Equation – 2nd order

$$\frac{\partial p(\mathbf{x},t)}{\partial t} - \nabla^2 p(\mathbf{x},t) = f(\mathbf{x},t), \quad \mathbf{x} \in \Omega, \ t \in [0,T]$$
$$p(\mathbf{x},0) = g(\mathbf{x}) \ \mathbf{x} \in \Omega$$
$$p(\mathbf{x},t) = h(\mathbf{x},t) \ \mathbf{x} \in \partial\Omega$$

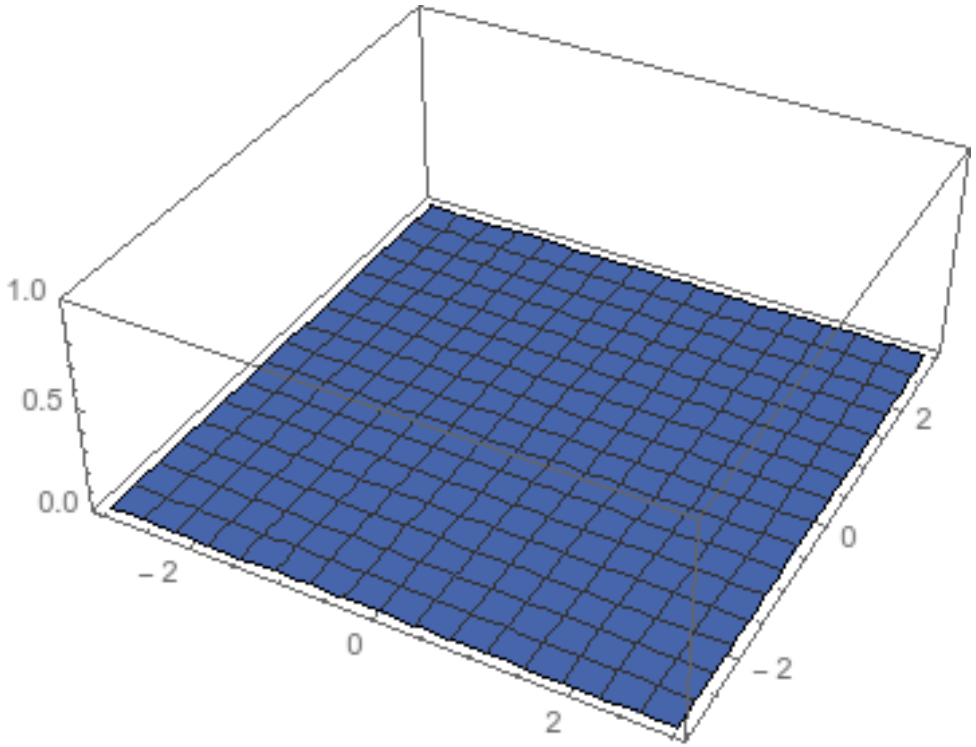
- Backward Euler

- FOSLS functional (LSF): Find $\mathbf{v}_{k+1}$ such that $G(\mathbf{v}_{k+1}) = \|L\mathbf{v}_{k+1} - \mathbf{f}_{k+1}\|$ is minimized at each time step

$$\overbrace{\begin{bmatrix} I & -\nabla \\ -\sqrt{dt}\nabla\cdot & \frac{I}{\sqrt{dt}} \\ \sqrt{dt}\nabla\times & 0 \end{bmatrix}}^{L} \overbrace{\begin{bmatrix} \mathbf{u}_{k+1} \\ p_{k+1} \end{bmatrix}}^{\mathbf{v}_{k+1}} = \overbrace{\begin{bmatrix} 0 \\ \frac{p_k}{\sqrt{dt}} + \sqrt{dt} f(\mathbf{x}, t_{k+1}) \\ 0 \end{bmatrix}}^{\mathbf{f}_{k+1}}$$

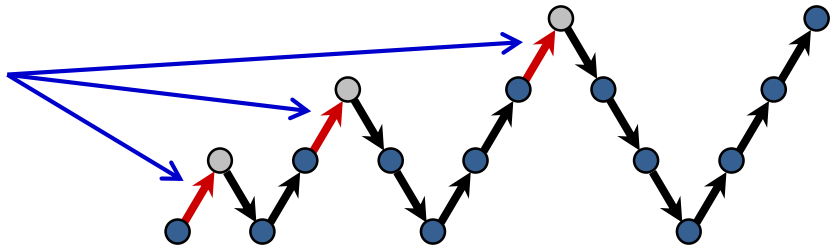
Manufactured solution yields refined space-time grids in the center of the space-time domain

$$u(x, y, t) = \exp[-100(t - 0.5)^2) - 6(x^2 + y^2)]$$
$$\Omega = [-3, 3]^2$$
$$t \in [0, 1]$$



Spatial and temporal adaptivity are done in the same place in the FMG cycle

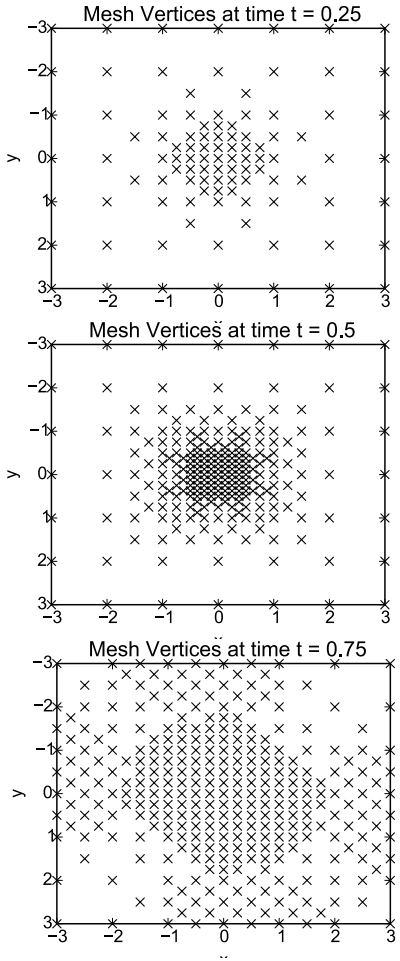
Spatial and temporal refinement done here



- This approach simplifies the code somewhat
  - For example, can assume vector sum occurs on the same spatial grid
- Temporal and spatial adaptivity are currently done separately
  - Temporal – based on Richardson extrapolation
  - Spatial – based on FOSLS error estimator (LSF); FOSLS functional computed locally on each cell; done with Coarsen() and Refine() functions
- Threshold refinement is used to mark the spatial mesh for refinement
  - Refine only a given percentage of the total error (Dörfler marking)

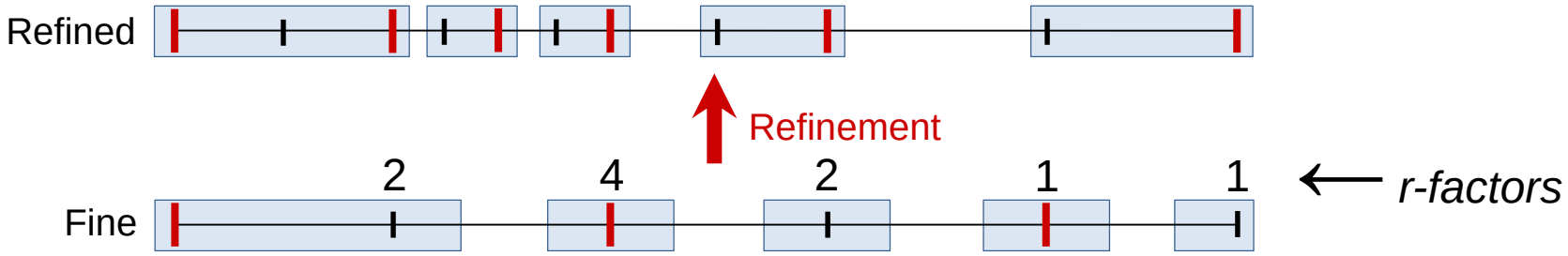
Some comments on results and plans

- Braid space-time solution resembles sequential case
- Without threshold refinement, get over-refinement in both space and time at later time points
  - Sequential uses 96 time steps
  - Braid uses 126 time steps
- Need load balancing in the temporal dimension
- Parameters used:
  - Time error tolerance = 0.0001
  - Space LSF tolerance = 0.001
  - Threshold refinement: 75% of global error
- Future: Add support for de-refinement

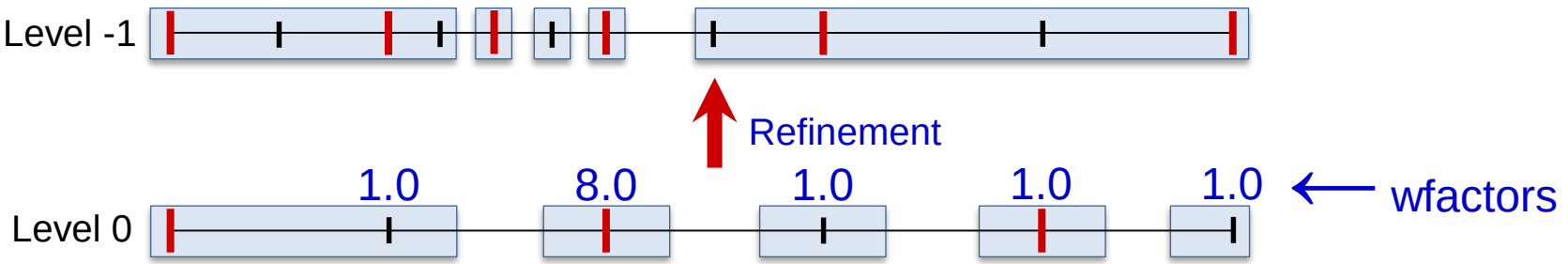


Load balancing is needed for space-time refinement since the cost of each time step varies

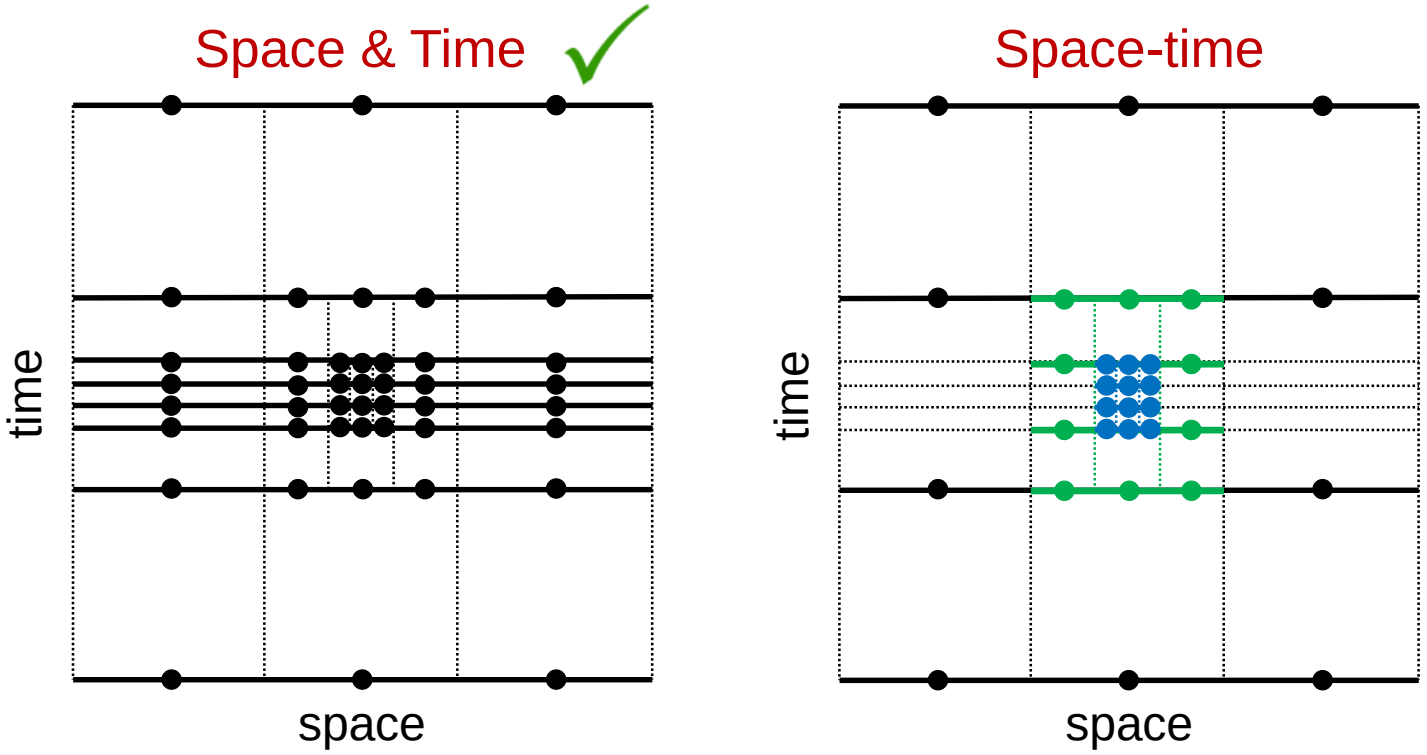
- Current temporal refinement approach, 5 processor example
  - User provides rfactors = requested refinement factors



- Developed a load balancing feature (not yet in the release)
  - User provides wfactors = weights to indicate time step costs
  - Employs assumed partition algorithm from *hypre*

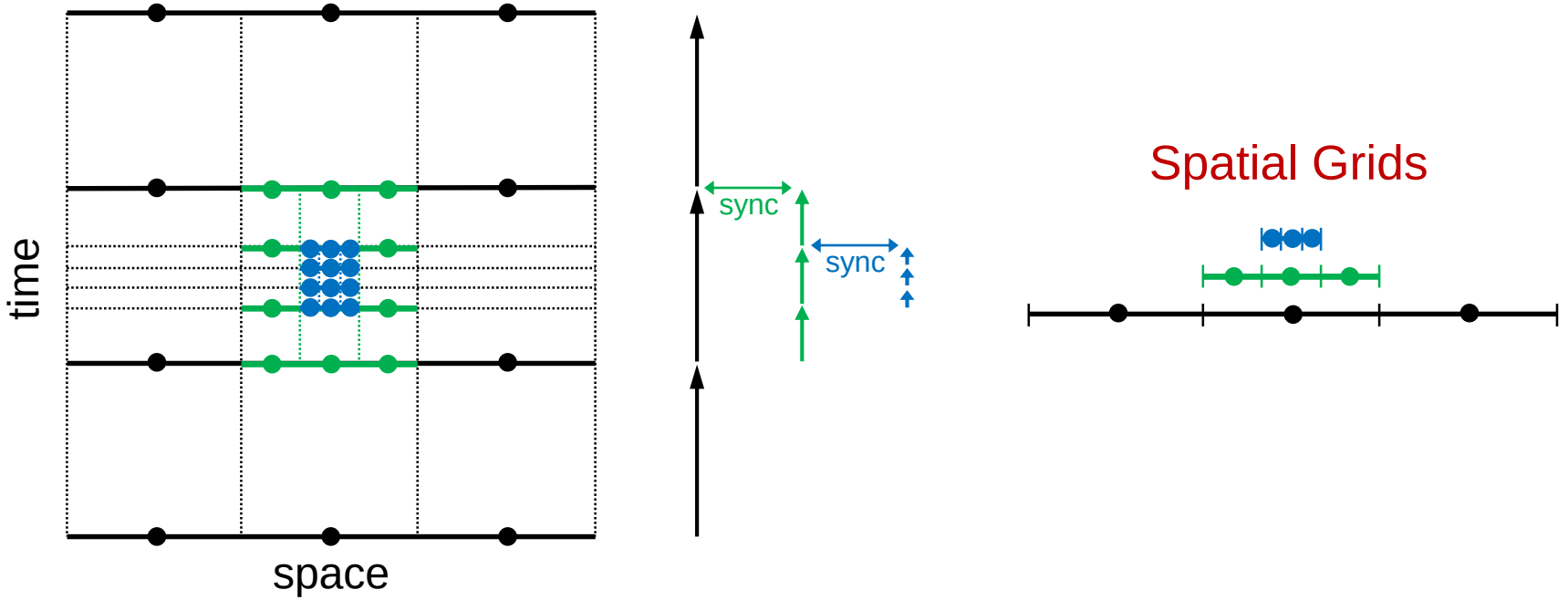


Future work – providing support for space-time refinement in block-structured AMR codes



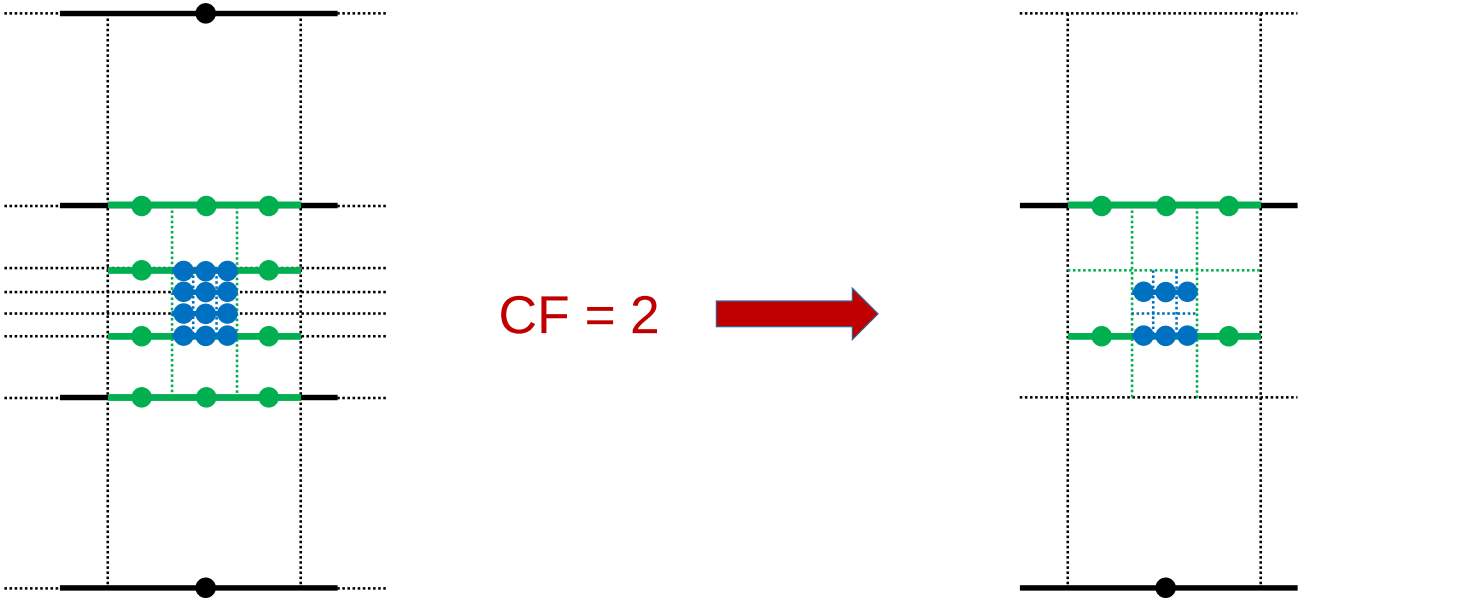
- Space-time refinement has been used for many years in the block-structured AMR community
  - E.g., AMReX (BoxLib), Chombo, SAMRAI

Time-stepping in SAMR is done via a recursive algorithm on a hierarchy of structured grids



- XBraid Vector = finest spatial-grid unknowns + unknowns on coarser grids nearby + other state data like flux arrays
  - Or something similar… details will vary between SAMR frameworks

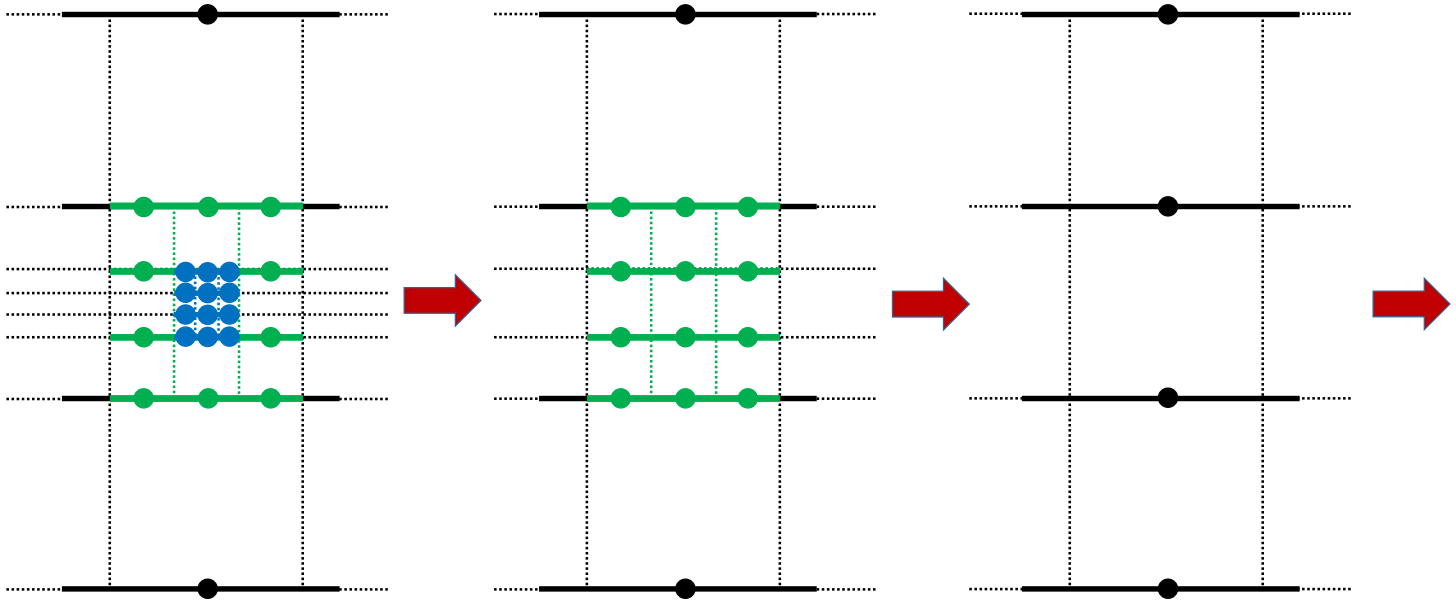
XBraid's fixed coarsening factor approach will not integrate well with SAMR codes



Note: no spatial coarsening

- This could work for new space-time FE approaches being developed, but another coarsening option is needed here

Most natural coarsening strategy for SAMR is to use reuse the XBraid FMG hierarchy



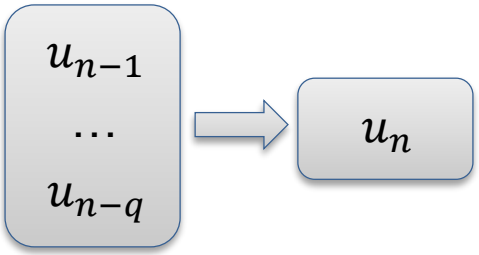
- New features need to be implemented
  - Variable coarsening (not too difficult)
  - Multilevel load balancing (harder)

Multistage / Multistep Methods and Power Grid Simulations

We are exploring parallel in time methods for both multistage and multistep methods

- XBraid framework is designed for one-step time-integrators (such as the multistage Runge-Kutta methods)

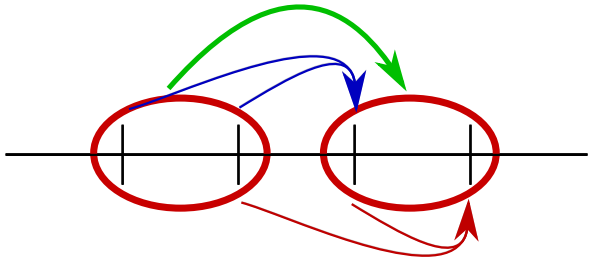
- Backward Difference Formula methods (BDF) are very efficient
  - They are multistep methods
  - They are **much cheaper**: require only **one** nonlinear solve per step
  - Easily provide local **error estimates**, useful for time adaptivity



How can we use multistep methods within the XBraid framework?

To fit in the XBraid one-step framework, we use a "trick"

- The multistep method can be rewritten as one-step method



  - Step 1: $u_{2n} = \varphi_{2n} \quad (u_{2n-2}, u_{2n-1})$
  - Step 2: $u_{2n+1} = \varphi_{2n+1}(u_{2n-1}, u_{2n})$

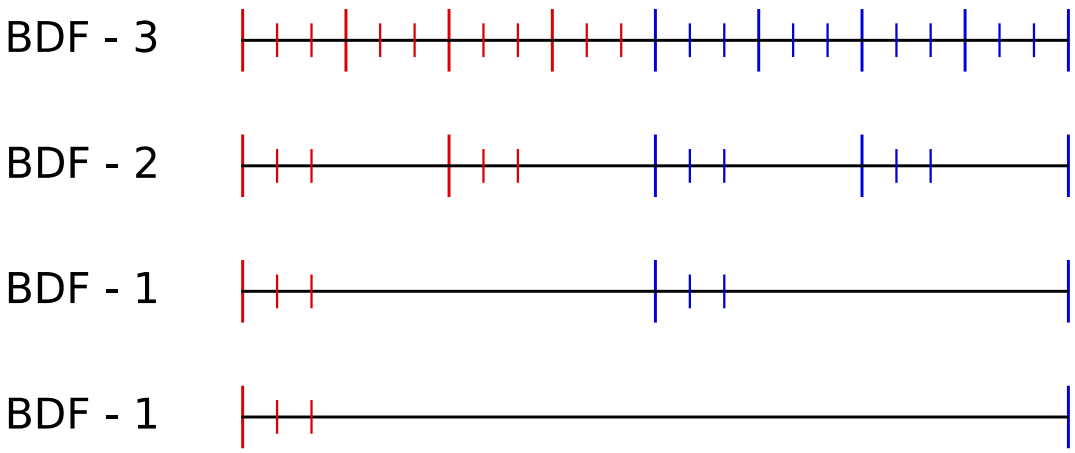
Exactly the same operations

- Redefine the stepping function as a one step method

$$w_n = \begin{bmatrix} u_{2n} \\ u_{2n+1} \end{bmatrix} = \widehat{\varphi_n}(w_{n-1}) \equiv \begin{bmatrix} \varphi_{2n}(u_{2n-2}, u_{2n-1}) \\ \varphi_{2n+1}\left(u_{2n-1}, \varphi_{2n}(u_{2n-2}, u_{2n-1})\right) \end{bmatrix}$$

Grouping unknowns can lead to stability problems with BDF methods

- We reduce the order on coarse time grids to maintain stability
- In almost all cases, this approach results in stability on all grids



Parallel-in-time for power grid systems

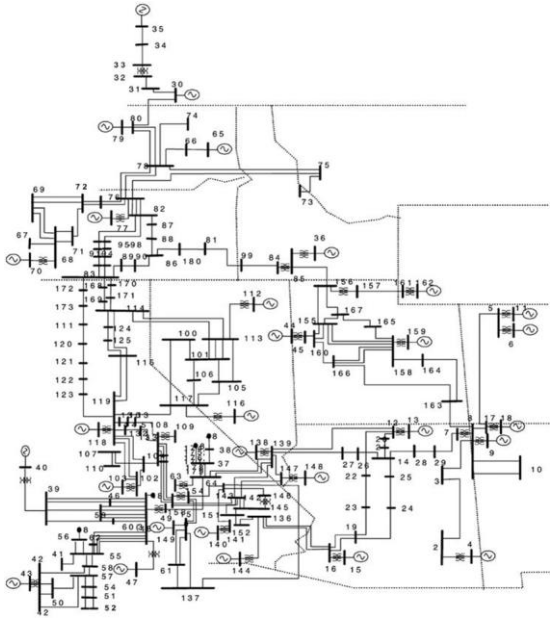
- Collaboration
  - Phillip Top (GridDyn)
  - Carol Woodward (SUNDIALS)
  - MGRIT team (Lecouvez, Schroder, Falgout)
- GridDyn simulates real-word power grids

GRIDDYN

  - Solves differential algebraic systems (DAEs)

$$F(t, y, \partial y/\partial t) = 0$$

  - Uses SUNDIALS for sequential time integration

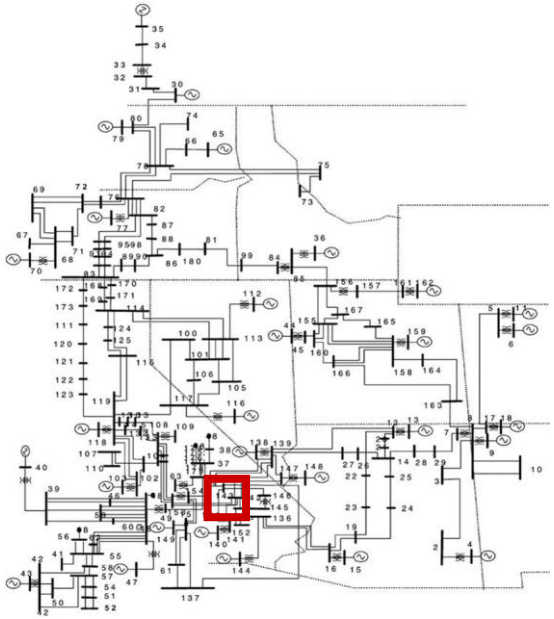


WECC System: 179 buses and 793 unknowns

- Sequential time integration bottleneck is present
  - Many time steps and a desire to achieve real-time and long-time simulations
  - Limited spatial parallelism

Parallel-in-time for power grid systems

- Target real-world scenarios with discontinuities
  - Discontinuity-handling is critical to be relevant
  - They arise due to equipment limit adjustments, controls, faults, etc.
  - Build on previous work*
- Model problem: apply square pulse to **bus 143** of WECC system every 2 seconds
  - Creates complex grid dynamics
  - Strategy:
    - Place a time point at each discontinuity
    - Use temporal adaptivity around discontinuity
    - Properly handle state at discontinuity
  - Explore scalability w.r.t. number of discontinuities
    - Longest simulation is 460s → 460 discontinuities



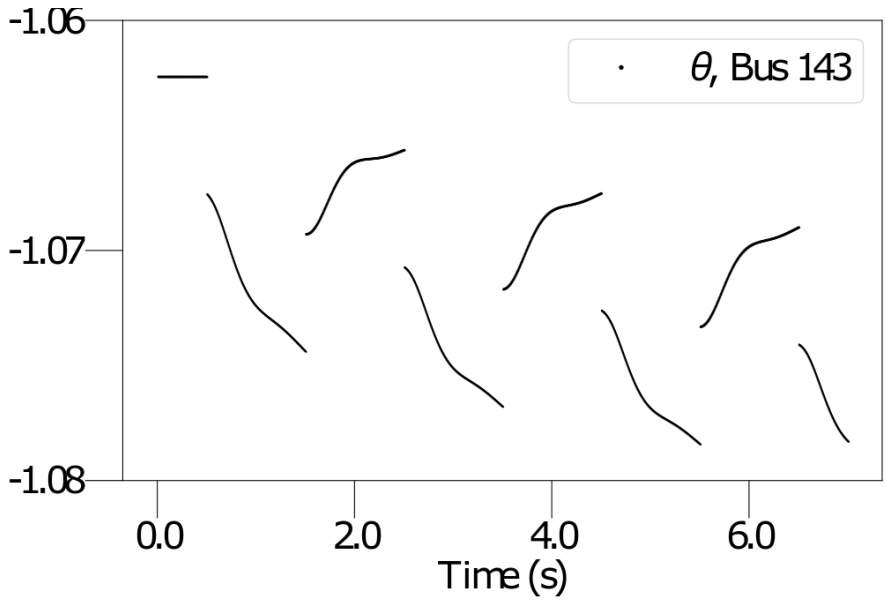
WECC System: 179 buses and 793 unknowns

50 MW load

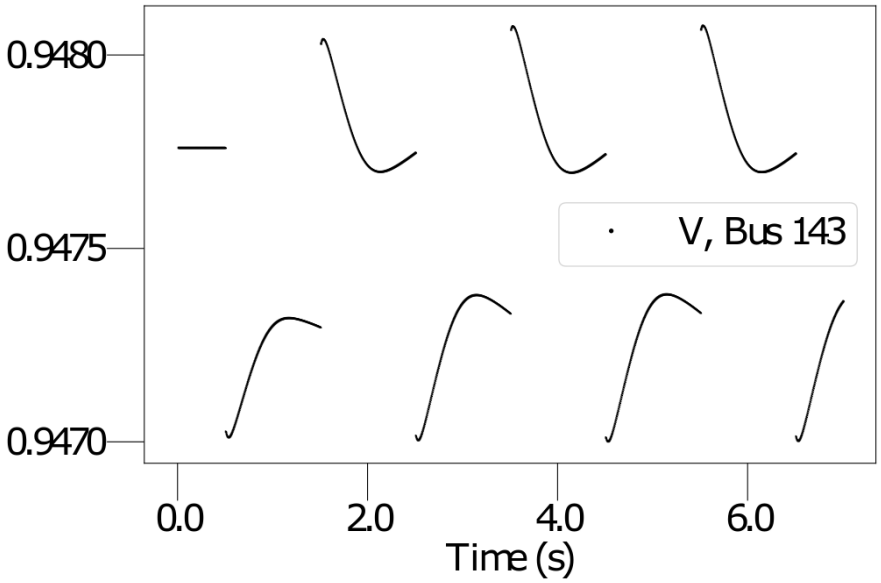
* Lecouvez, Falgout, Woodward, Top, "A Parallel Multigrid Reduction in Time Method for Power Systems," PES, 2016.

Two solution components for bus 143

Angle at bus 143



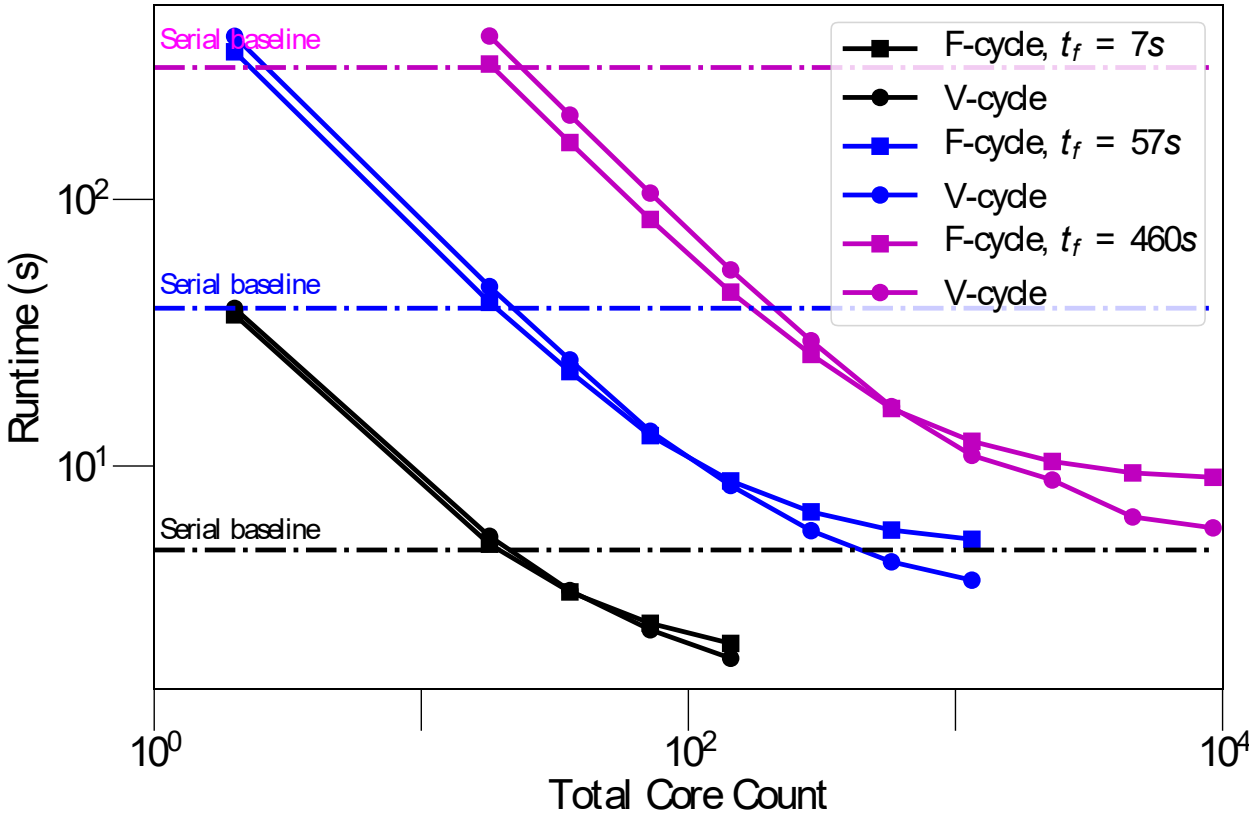
Voltage at bus 143



- Note that MGRIT coarsens well beyond a time-step size of 1 sec
  - Coarsest grid has only 4 time points

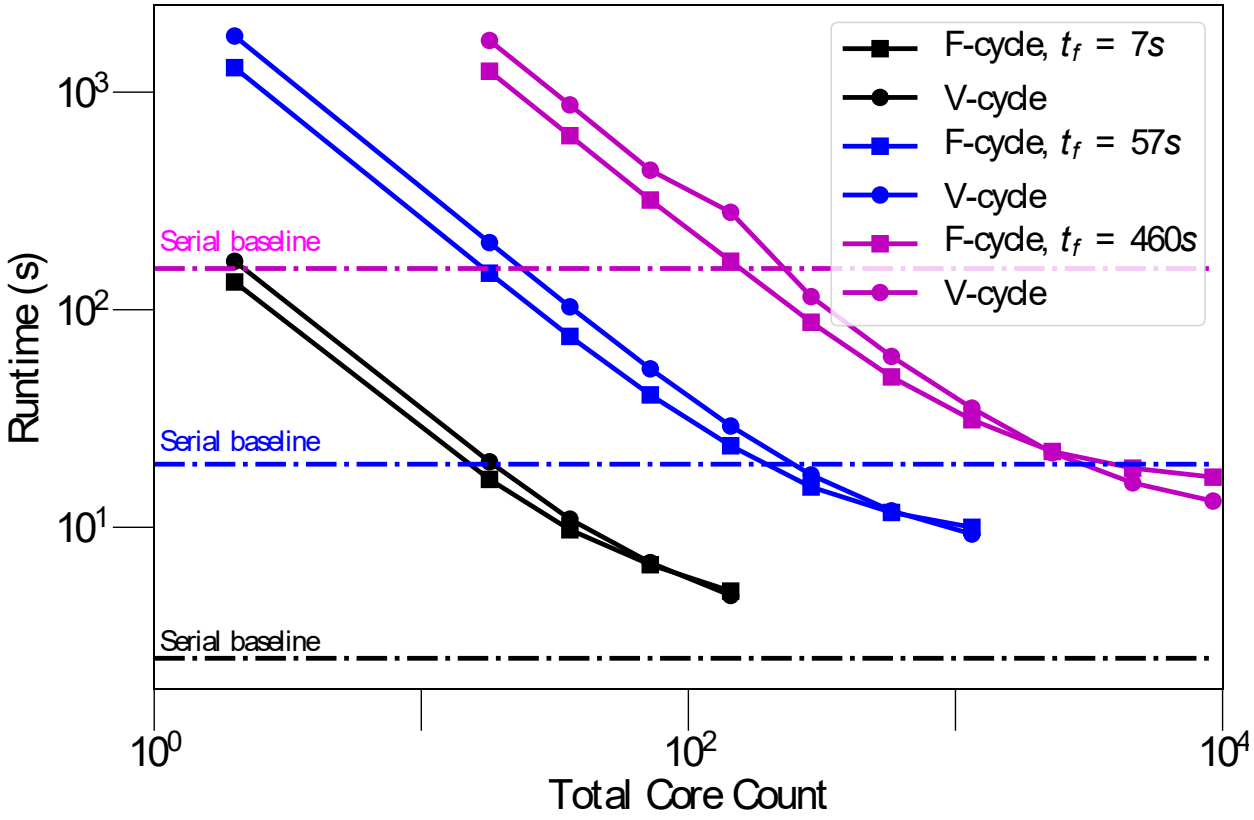
Results: SDIRK-4 method and 5ms time-step size – max speedup with 92K time steps ~53x

- Solver is robust with respect to the number of discontinuities
  - Proper placement of time-points around discontinuity is critical



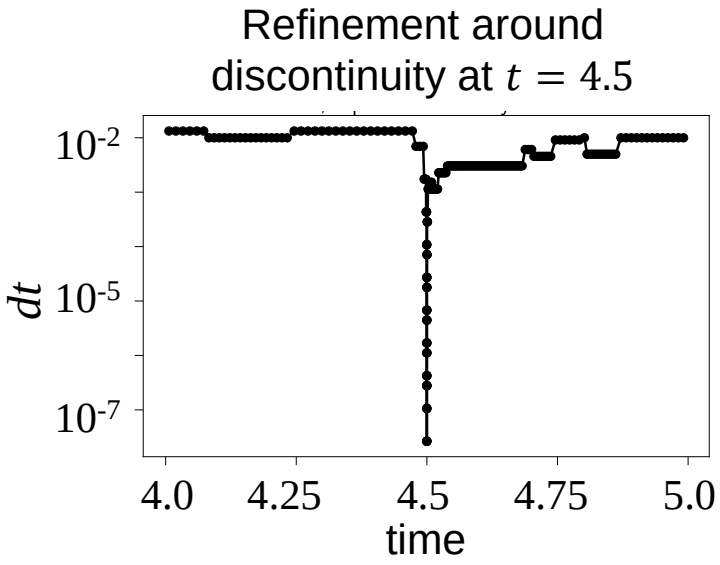
Results: BDF2 method and 5ms time-step size – max speedup with 92K time steps ~12x

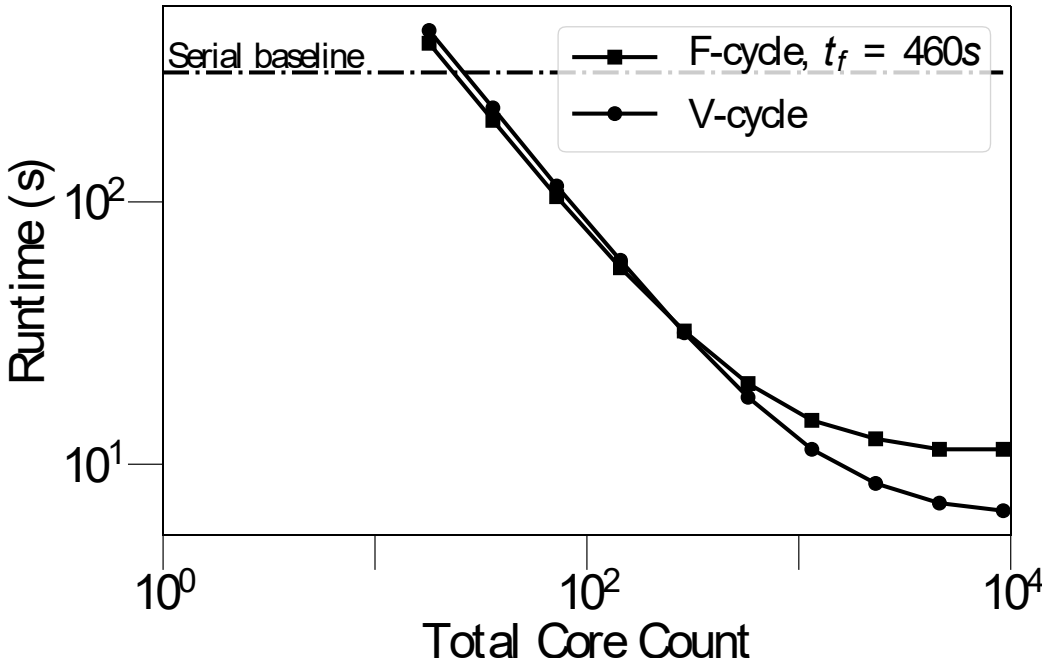
- Solver is robust with respect to the number of discontinuities
  - Proper placement of time-points around discontinuity is critical



Results: SDIRK-4 method with variable time-stepping (nested iteration) – max speedup ~47x

- Adaptively refine around discontinuities for improved accuracy
  - Approximately 114K time points





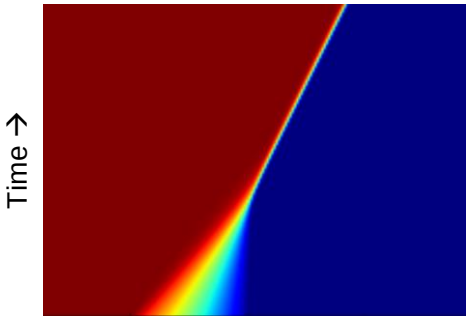
- Current research: discontinuities with unknown location

Hyperbolic Problems

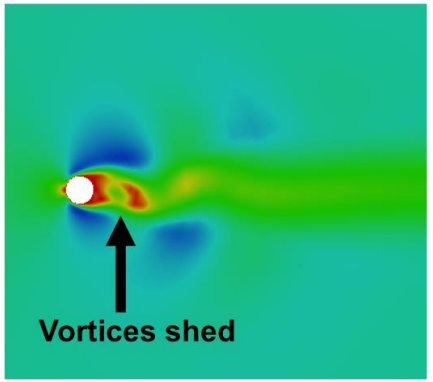
Hyperbolic problems are a major new emphasis for our MGRIT algorithm research

- We have already had some initial success…
- 1D/2D advection and Burgers' equation
  - F-cycles needed (multilevel), slow growth in iterations
  - Requires adaptive spatial coarsening
  - Dissipation improves convergence
  - Mainly SDIRK-k (implicit) schemes to date
- Combination of FCF relaxation, F-cycles, and small coarsening factors improves robustness
  - Confirmed by theory
- Navier-Stokes in 2D and 3D
  - Multiple codes: Strand2D, Cart3D, LifeV, CHeart
  - Compressible and incompressible
  - Modest Reynolds numbers (100 – 1500)

1D Inviscid Burgers

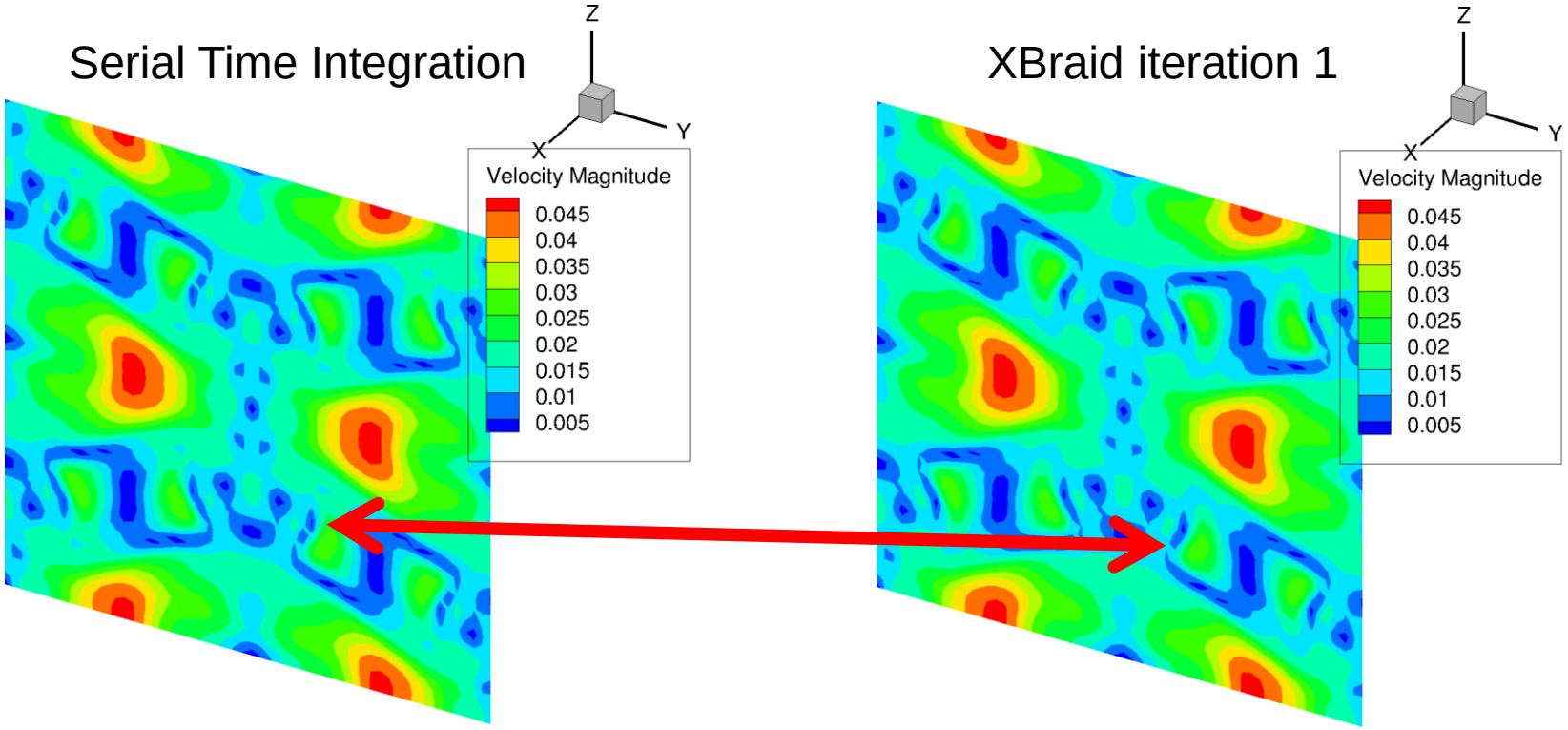


Navier-Stokes
7.5x speedup in Strand2D



Compressible Navier-Stokes with Cart3D – convergence is fast, ~5 iterations

- Taylor-Green problem: turbulent decay of vortex, Re=1600
  - Higher-order spatial discretization on 58^3 x 20,000 Cartesian grid
  - Velocity magnitude at x=0 cross-section



Adaptive space-time coarsening for linear advection and inviscid Burgers' equation

- Consider a scalar 1D conservation law

$$\partial_t u + \partial_x \big(f(u, x, t) \big) = 0$$

- Space discretization – vertex-centered finite volume approach with Lax-Friedrichs flux approximation for $f^*(t)$

$$\partial_t u_j + \frac{1}{\delta x_j} \left(f^*_{j+\frac{1}{2}}(t) - f^*_{j-\frac{1}{2}}(t) \right) = 0$$

- Time discretization – both implicit and explicit Euler

- Two equations:
  - Linear advection: $u_t + a(x,t) u_x = 0$
  - Inviscid Burgers' equation: $u_t + u u_x = 0$

Coarsening in space is detrimental when the wave speed is small (showing iteration counts)

- Advection: $u_t + au_x = 0, u_0(x) = \sin\left(\frac{\pi x}{2}\right), -2 \le x \le 2, 0 \le t \le 4$
- Factor-2 space/time coarsening; geometric interpolation/restriction in space

			Implicit			Explicit	
	$N_x \times N_t$	$2^7 \times 2^7$	$2^9 \times 2^9$	$2^{11} \times 2^{11}$	$2^7 \times 2^8$	$2^9 \times 2^{10}$	$2^{11} \times 2^{12}$
$a = 1$	Time only, 2-level	14	15	15	50	100+	100+
	Time only, F-cycle	14	17	22	100+	100+	100+
	Time + Space, 2-level	15	15	16	30	31	31
	Time + Space, F-cycle	15	20	28	34	41	54
$a = 0.1$	Time only, 2-level	8	8	8	7	7	7
	Time only, F-cycle	8	9	10	8	34	100+
	Time + Space, 2-level	64	92	92	100+	100+	100+
	Time + Space, F-cycle	64	94	95	100+	100+	100+

Adaptive spatial coarsening – coarsen in space only when wave speed is "large enough"

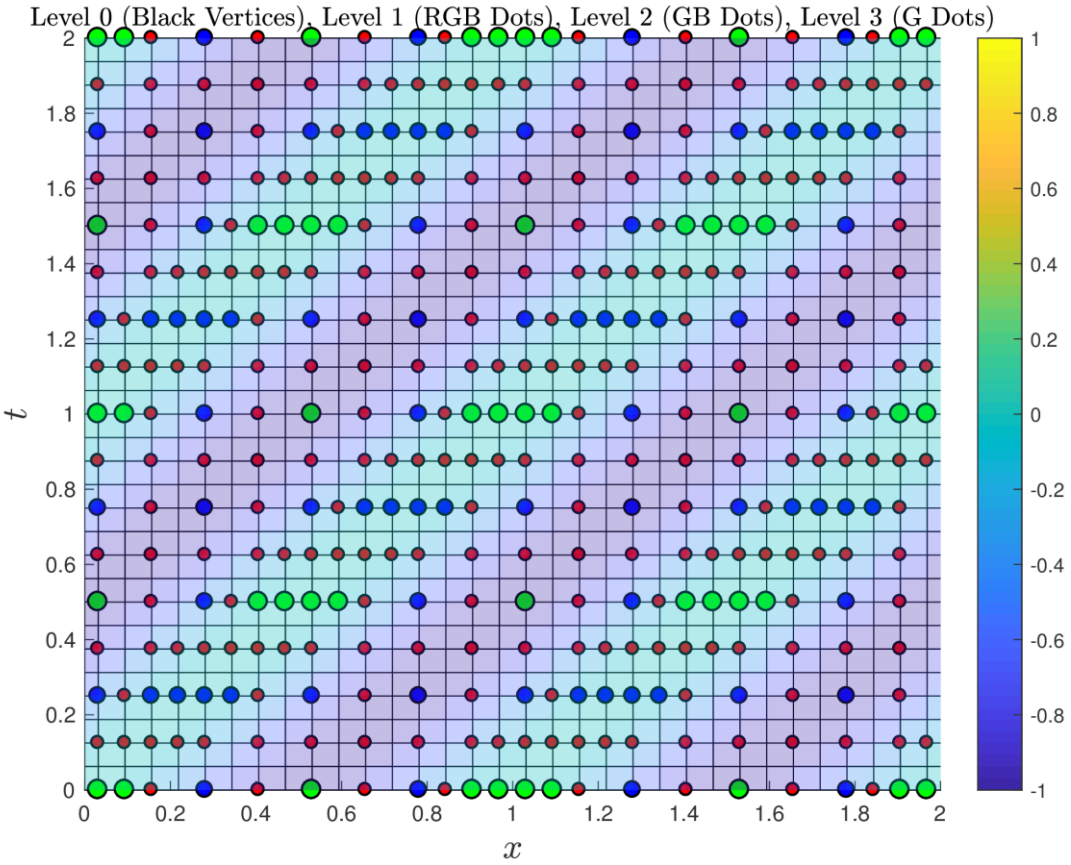
- Basic approach:
  - Coarsen in space (factor 2) to build tentative coarse grids (all levels)
  - Add points to each coarse grid if the wave speed is "small":

$$\text{if } \left|\frac{\partial f}{\partial u}\right| \frac{\delta t}{\delta x_j} < tol \text{ , add ("keep") cell } j$$

- For implicit, this is sufficient
- For explicit, need to balance convergence and stability
  - Mark cells as "keep", "delete", or "neutral" based on local Courant number
  - If several "delete" cells are adjacent, coarsen the sequence by 2
  - If a single "delete" cell lies between two "keep" cells, use a local coarse-grid Courant number to make decisions
- For Burgers, we only have approximations to wave speeds
  - Compute spatial grids for each iteration until the residual is "small"

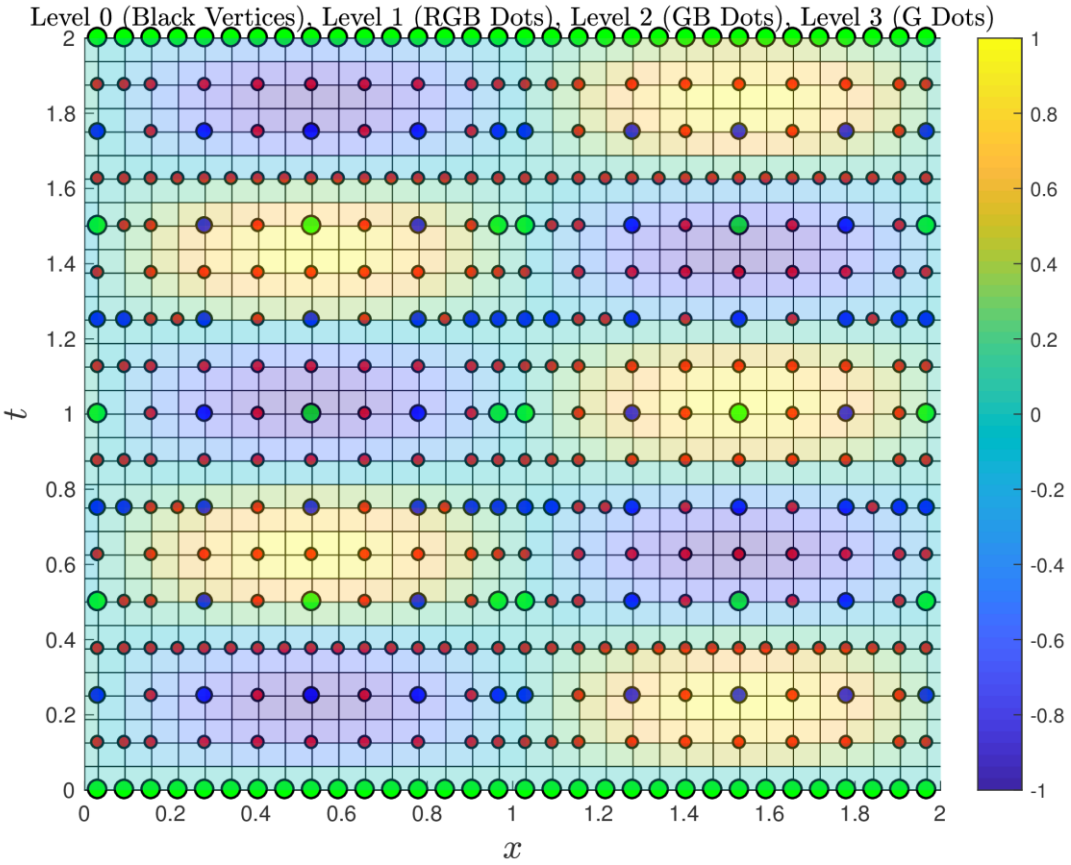
Space-time grids for linear advection (1) – performance is similar to case with $a = 1$

- $a(x,t) = -\sin^2\big(\pi(x-t)\big)$, $N_x = N_t = 64$



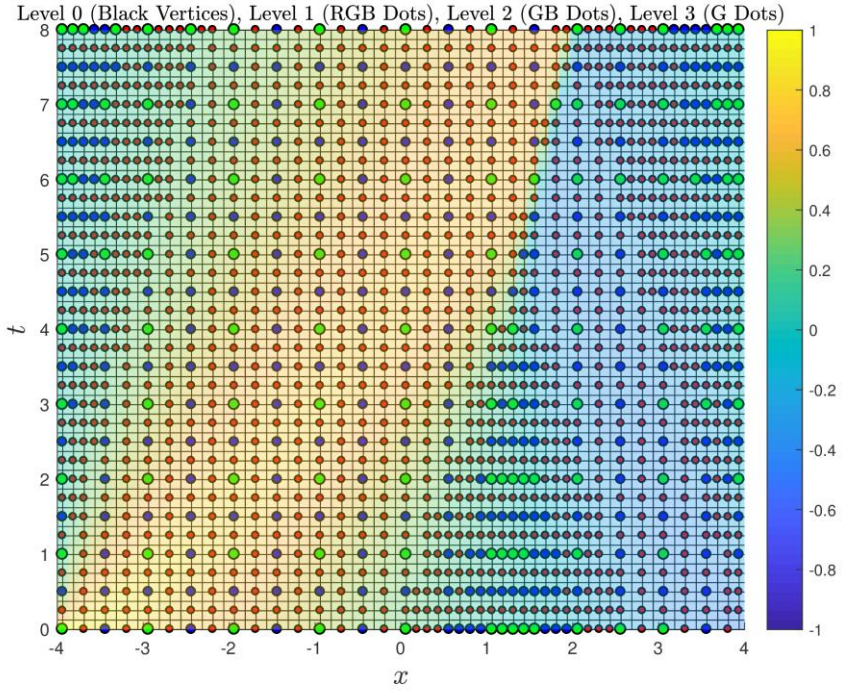
Space-time grids for linear advection (2) – performance is similar to case with $a = 1$

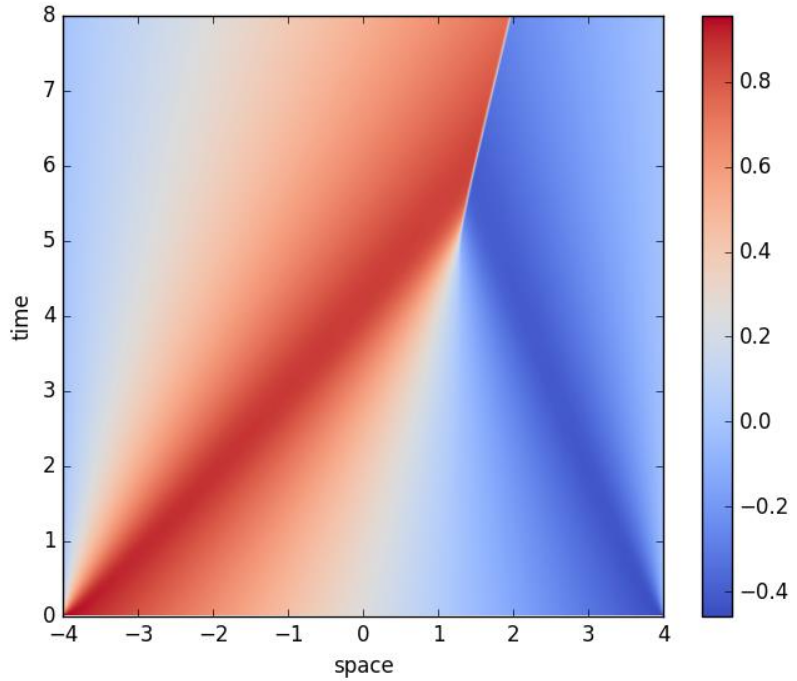
- $a(x,t) = -\sin(2.5\pi t)\sin(\pi x)\,, \quad N_x = N_t = 64$



Space-time grids for inviscid Burgers' equation – convergence is not affected by shock

- $u_0(x) = 0.25 - 0.75 \sin\left(\frac{\pi x}{16}\right),\ N_x = N_t = 64$
- Convergence ~50% slower than explicit advection studies
  - Still needs work, but initial results are promising

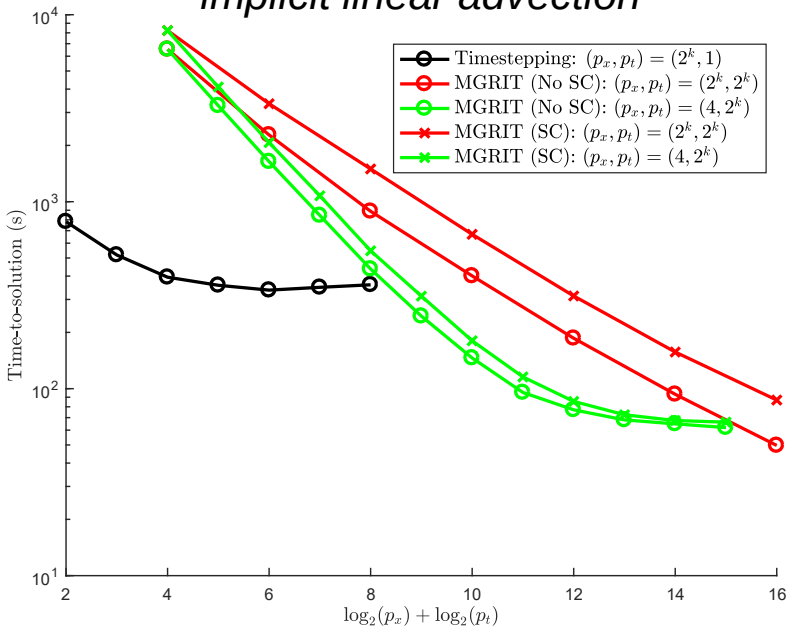




Status of adaptive space-time coarsening work

- Parallel speedups (so far) on IBM BG/Q at LLNL
  - Explicit linear advection ~ 4x on 131K cores
  - Implicit linear advection ~ 6x on 65K cores
  - Burgers' equation: still a work in progress
- Improving parallel results
  - Main issue: need faster convergence
  - Parallel space-time decomposition
  - Additional optimizations to the code
- Next steps:
  - Higher-order discs (less diffusive)
  - Higher dimensions (2D/3D)

Strong scaling - implicit linear advection



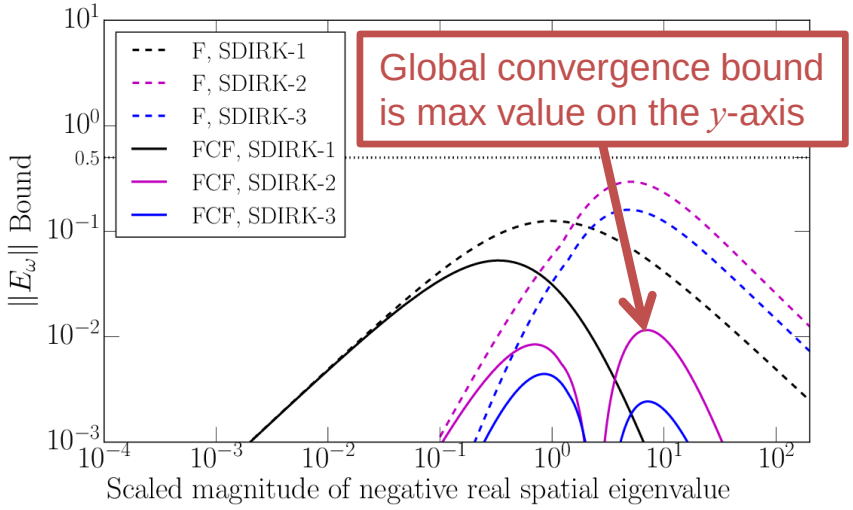
Theory

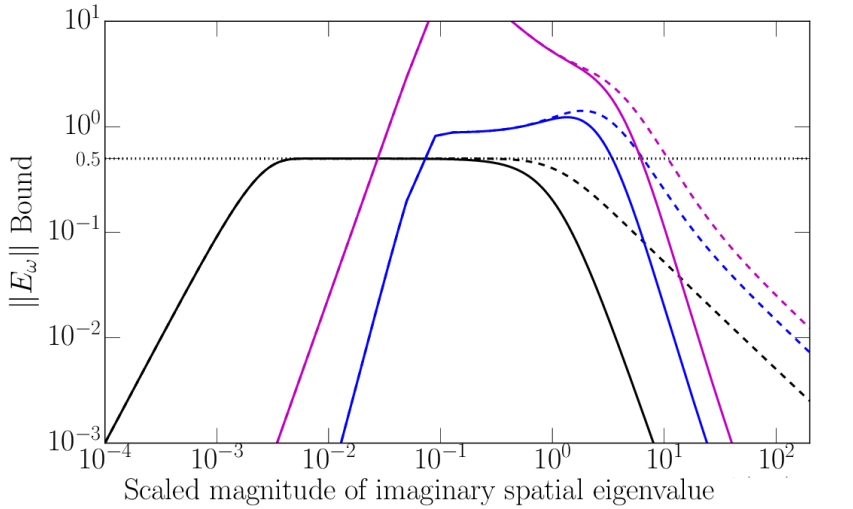
We developed a linear two-grid convergence theory to guide MGRIT algorithm development

- Assume Φ and Φ_Δ are simultaneously diagonalizable with eigenvalues $\lambda_\omega, \mu_\omega$
- Sharp bound for error propagator (FCF)

$$\|E\| \leq \max_\omega |\lambda_\omega^m - \mu_\omega| \frac{1 - |\mu_\omega|^{N_T - 1}}{1 - |\mu_\omega|} |\lambda_\omega|^m$$

- Agnostic to space-time discretization
  - But discretization affects convergence
- Eigenvalues (representative equation):
  - Real (parabolic)
  - Imaginary (hyperbolic without dissipation)
  - Complex (hyperbolic with dissipation)
- Insights:
  - FCF significantly faster
  - High order can be faster or slower
  - Small coarsening factors sometimes needed
  - Artificial dissipation helps a lot





Richardson Extrapolation

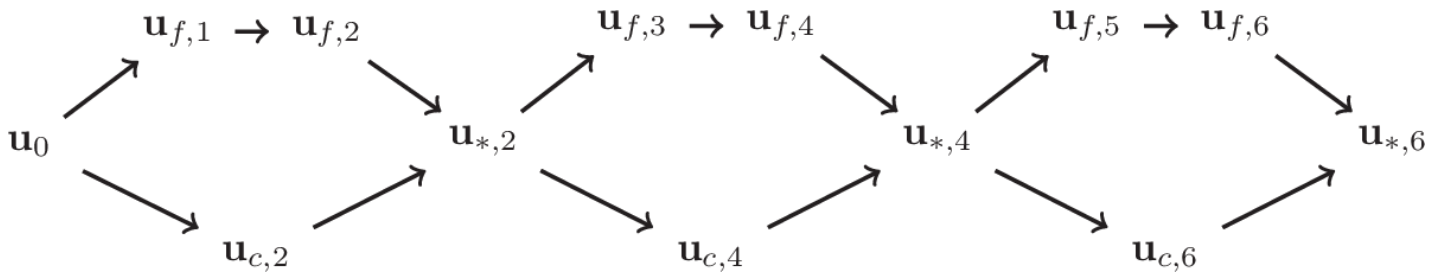
Richardson extrapolation (RE) can extend MGRIT to improve time step accuracy at almost no extra cost

- RE combines approximations at two scales to achieve higher order
  - Consider fine and coarse grids with coarsening factor m
  - Let $u_{f,i}$ and $u_{c,i}$ be k_g-order fine and coarse approximations at point i

$$u_{*,i} = a\, u_{f,i} - b\, u_{c,i}\,; \quad a = \frac{m^{k_g}}{m^{k_g} - 1}\,; \quad b = \frac{1}{m^{k_g} - 1}$$

← Richardson Extrapolation

- Note: RE does not guarantee improvement (asymptotic)
- Sequential RE:



RE is referred to as τ-extrapolation in the multigrid community → τ-MGRIT

- RE can be viewed as a forward solve of the following system (considering the linear case again for simplicity)

$$A^{\tau}\mathbf{u} = \begin{bmatrix} I & & & & & & & \\ -\Phi & I & & & & & & \\ & \ddots & \ddots & & & & & \\ b\Phi_{\Delta} & & -a\Phi & I & & & & \\ & 0 & & -\Phi & I & & & \\ & & \ddots & & \ddots & \ddots & & \\ & & & b\Phi_{\Delta} & & -a\Phi & I & \\ & & & & \ddots & & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_m \\ \mathbf{u}_{m+1} \\ \vdots \\ \mathbf{u}_{2m} \\ \vdots \end{bmatrix} = \bar{\mathbf{g}}$$

τ-MGRIT is derived similarly to MGRIT

- Ideal Petrov-Galerkin coarse-grid operator

$$A_\Delta^\tau \mathbf{u}_\Delta = \begin{bmatrix} I & & & \\ (b\Phi_\Delta - a\Phi^m) & I & & \\ & (b\Phi_\Delta - a\Phi^m) & I & \\ & & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\Delta,0} \\ \mathbf{u}_{\Delta,1} \\ \mathbf{u}_{\Delta,2} \\ \vdots \end{bmatrix}$$

- Coarse-grid discretization = practical approximation to ideal

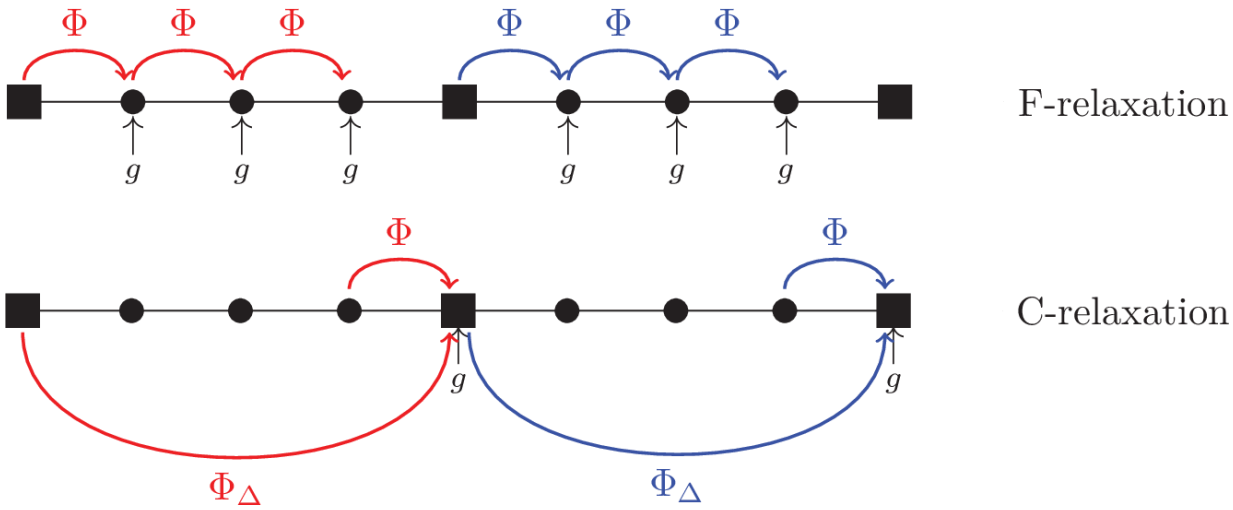
$$B_\Delta = \begin{bmatrix} I & & & \\ -\Phi_\Delta & I & & \\ & \ddots & \ddots & \\ & & -\Phi_\Delta & I \end{bmatrix}$$

τ-MGRIT involves a slight modification to the FAS coarse-grid right-hand-side

- FAS coarse-grid equations ($b = 0$ is standard MGRIT):

$$B_\Delta(u_\Delta) = R_I(g - A^\tau u) + B_\Delta(R_I u)$$
$$= R_I(g - Au) + (1 + b)B_\Delta(R_I u) - bR_I A(u)$$

- Right-hand-side is modified with "C-relaxation" below
  - These are already computed quantities in standard MGRIT



Theory for τ-MGRIT is similar to MGRIT results

- Error contraction bounds for F- and FCF-relaxation MGRIT

$$||E_{\Delta}^{F}\bar{\mathbf{e}}||_2 \leq \max_{\omega=1,2,\ldots,N_x} \left\{ |\lambda_{\omega}^{m} - \mu_{\omega}| \frac{1-|\mu_{\omega}|^{N_t/m}}{(1-|\mu_{\omega}|)} \right\} ||\bar{\mathbf{e}}||_2,$$

$$||E_{\Delta}^{FCF}\bar{\mathbf{e}}||_2 \leq \max_{\omega=1,2,\ldots,N_x} \left\{ |\lambda_{\omega}^{m} - \mu_{\omega}| \frac{1-|\mu_{\omega}|^{N_t/m}}{(1-|\mu_{\omega}|)} |\lambda_{\omega}|^{m} \right\} ||\bar{\mathbf{e}}||_2,$$

- Error contraction bounds for F- and FCF-relaxation τ-MGRIT

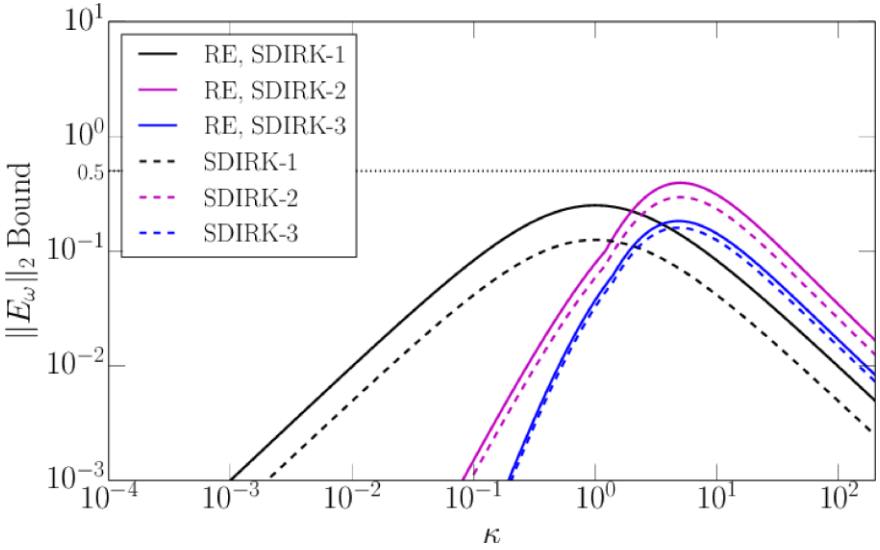
$$||\bar{E}_{\Delta}^{F}\bar{\mathbf{e}}||_2 \leq \max_{\omega=1,2,\ldots,N_x} \frac{m^{k_g}}{m^{k_g}-1} \left\{ |\lambda_{\omega}^{m} - \mu_{\omega}| \frac{1-|\mu_{\omega}|^{N_T/m}}{(1-|\mu_{\omega}|)} \right\} ||\bar{\mathbf{e}}||_2.$$

Note multiplier

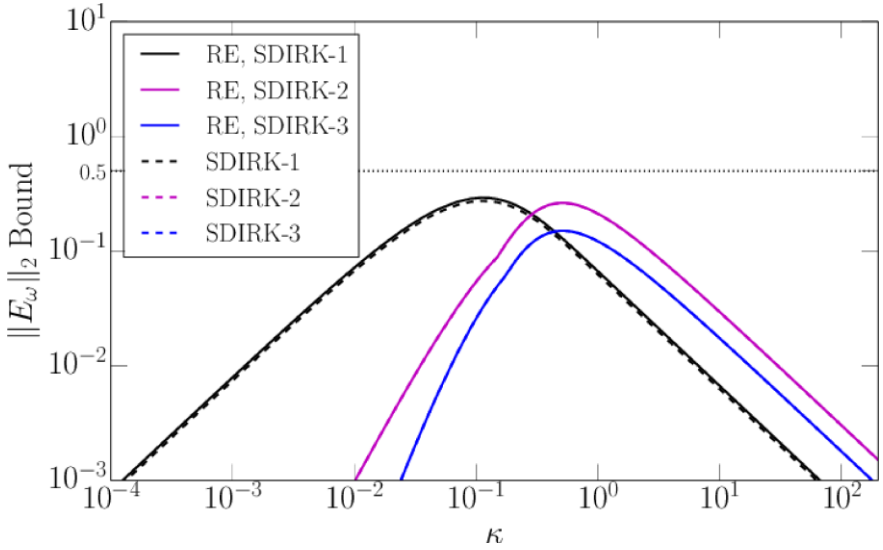
$$||\bar{E}_{\Delta}^{FCF}\bar{\mathbf{e}}||_2 \leq \max_{\omega=1,2,\ldots,N_x} \frac{m^{k_g}}{m^{k_g}-1} \left\{ |\lambda_{\omega}^{m} - \mu_{\omega}| \frac{1-|\mu_{\omega}|^{N_T/m}}{(1-|\mu|)} \left| \frac{m^{k_g}\lambda_{\omega}^{m} - \mu_{\omega}}{m^{k_g}-1} \right| \right\} ||\bar{\mathbf{e}}||_2.$$

F-relaxation – τ-MGRIT is slightly slower, but can increase convergence order

- τ-MGRIT = solid lines; MGRIT = dotted lines



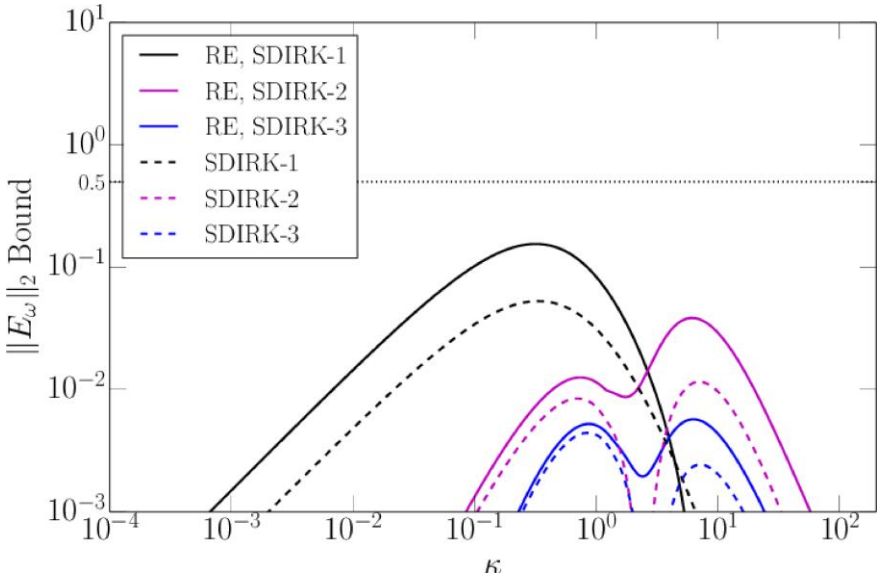
$m = 4$



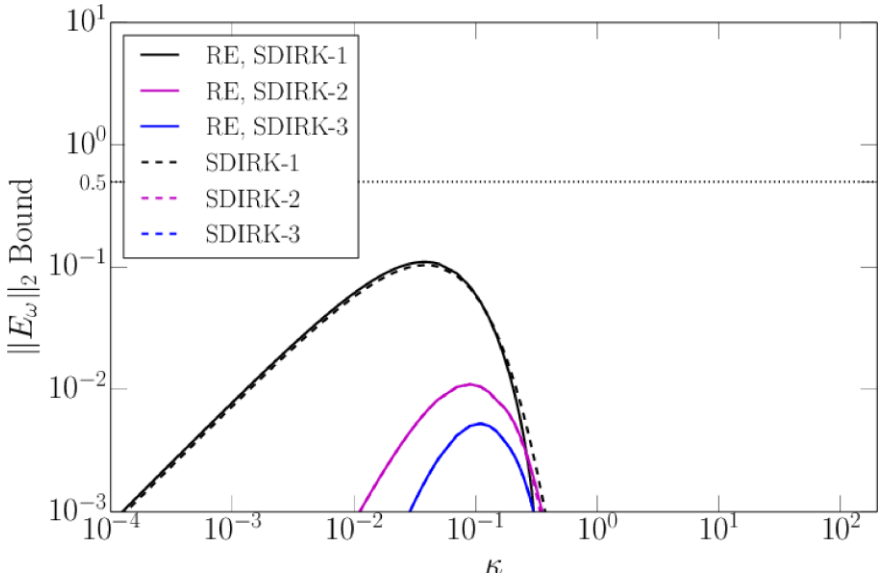
$m = 16$

FCF-relaxation – τ-MGRIT is slightly slower, but can increase convergence order

- τ-MGRIT = solid lines; MGRIT = dotted lines



$m = 4$

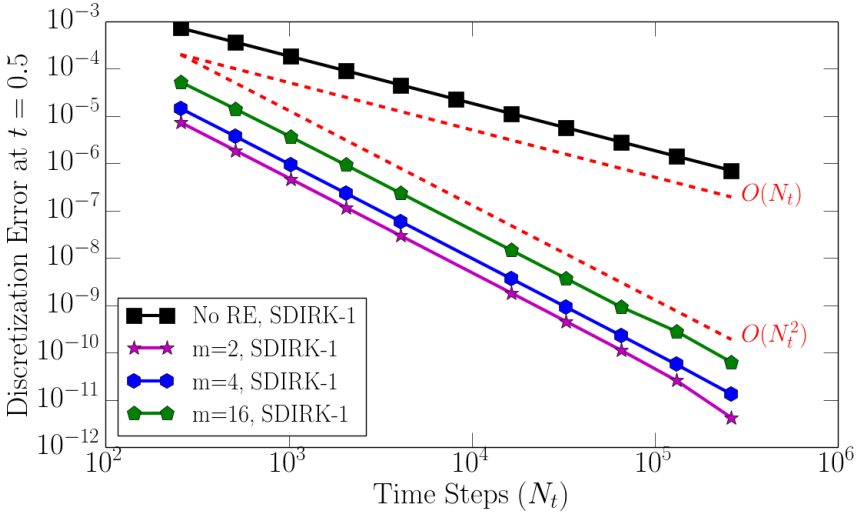


$m = 16$

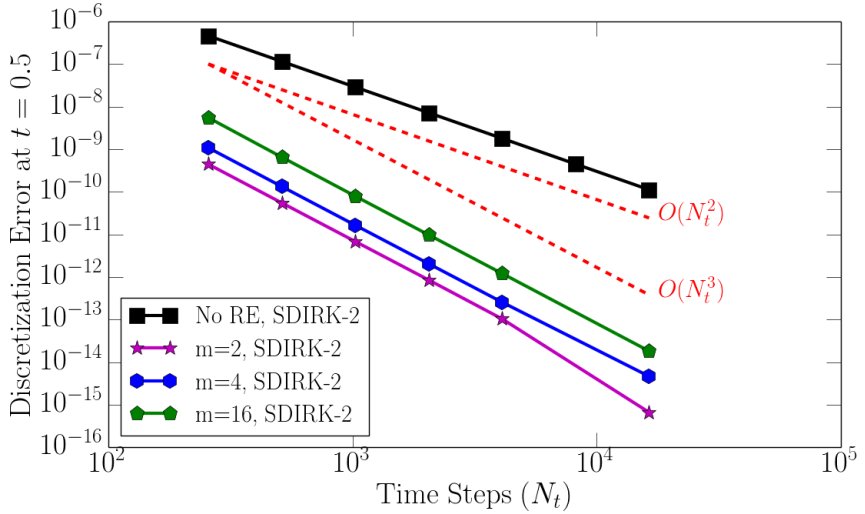
Numerical experiments – first order ODE

$$y' + 4y = 1 - t, \;\; t \in [0, 1],$$
$$y(0) = 1,$$

- Exact solution: $y(t) = (1/16)(-4t + 11e^{-4t} + 5)$



SDIRK-1

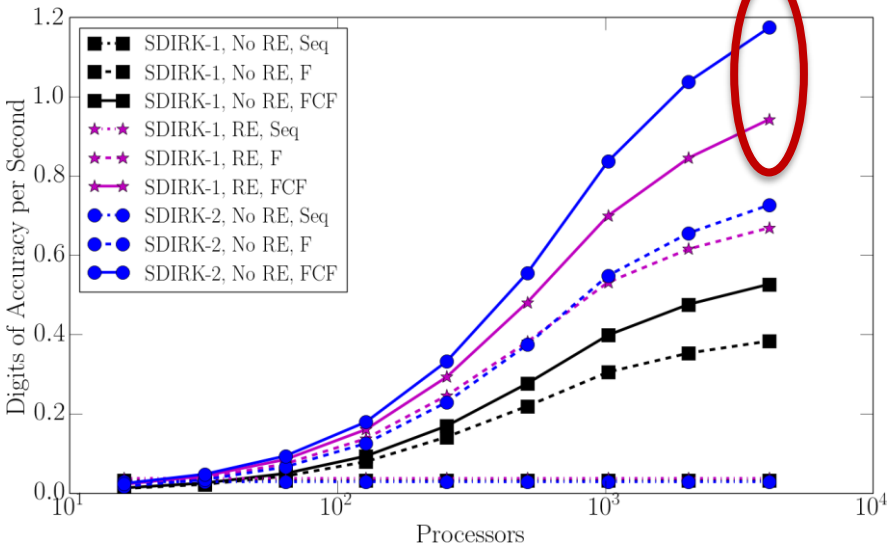


SDIRK-2

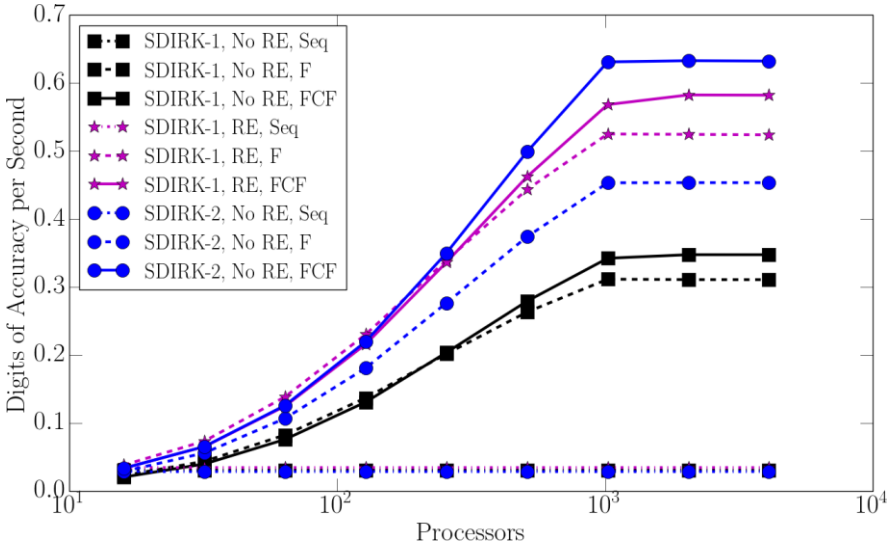
Digits accuracy per second, 1D Heat Equation

- SDIRK-1 τ-MGRIT is not better than SDIRK-2 MGRIT, but …
  - τ-MGRIT potentially improves any given method at no extra cost

Compare



$m = 4$



$m = 16$

τ-MGRIT is natural for adaptive time integration

- RE provides an error estimate (uniform grid)

$$E = \frac{u_f - u_c}{m^{k_l} - m} = \frac{u_f - u_c}{m(m^{k_g} - 1)}$$

- RE on variably-spaced grids

$$u_{*,mj} = \bar{a} u_{f,mj} - \bar{b} u_{c,mj}$$

$$\bar{a} = \frac{\Delta T_{j/m}^{k_l}}{\Delta T_{j/m}^{k_l} - \sum_{i=0}^{m-1} \delta t_{mj-i}^{k_l}}, \quad \bar{b} = \frac{\sum_{i=0}^{m-1} \delta t_{mj-i}^{k_l}}{\Delta T_{j/m}^{k_l} - \sum_{i=0}^{m-1} \delta t_{mj-i}^{k_l}}$$

- Error estimate on variably-space grid

$$E_{mj-i} = C_{mj} \delta t_{mj-i}^{k_l}$$

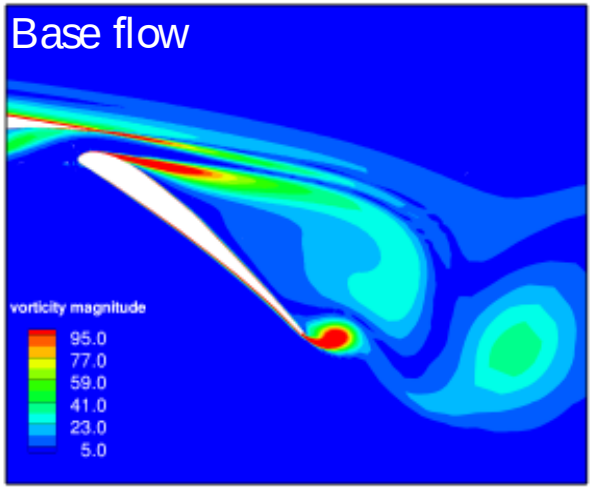
$$|C_{mj}| = \frac{1.0}{\Delta T_{j/m}^{k_l} - \sum_{i=0}^{m-1} \delta t_{mj-i}^{k_l}} ||u_{f,mj} - u_{c,mj}||$$

New Optimization Feature in Xbraid: XBraid-adjoint

Motivation: PDE constrained optimization

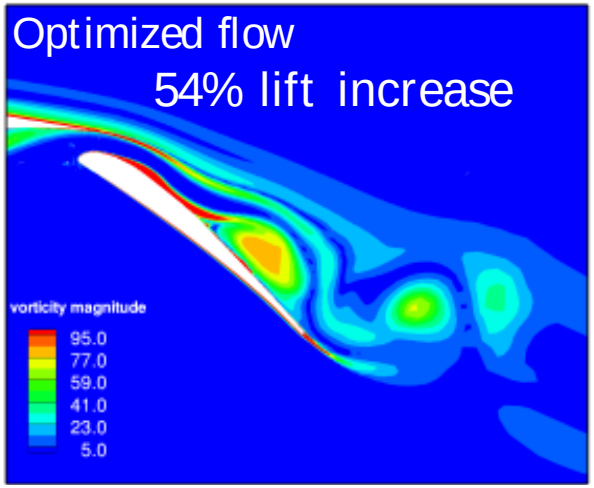
Example[1]

- Objective: Lift maximization
- Design: Amplitudes of actuation



Runtimes

- Simulation: 2.5h
- Optimization: 1,152h



1. Ötzkaya, Nemili et al., 2015

Problem description: Optimization with unsteady PDEs

- Optimize objective function J, with a design variable u *(continuous)*

$$\min \frac{1}{T} \int_0^T J(y(t), u) dt$$

- While satisfying constraint of the forward in time process, with state variable y and initial condition g *(continuous)*

$$\frac{\delta y(t)}{\delta t} + c(y(t), u) = 0, \quad \forall t \in (0, T)$$
$$y(0) = g$$

Problem description: Optimization with unsteady PDEs

- Optimize objective function *J*, with a design variable u *(discrete)*

$$\min \frac{1}{N} \sum_{i=1}^{N} J(y^i, u) \qquad J^n := J(y^n, u)$$

- While satisfying constraint of the forward in time process, with state variable y and initial condition g *(discrete)*

$$y^n = \Phi(y^{n-1}, u), \qquad n = 1, \ldots, N$$
$$y^0 = g$$

First Order Optimality Conditions

Form Lagrangian $$L = \sum_{i}^{N} (J^n + (\bar{y}^n)^T (\Phi^{n-1} - y^n))$$

1. State equations:

$$y^n = \Phi(y^{n-1}, u\ ), \qquad n = 1, \ldots, N$$

$$y^0 = g$$

2. Adjoint equations:

$$\bar{y}^n = \nabla_{y^n} J^n + (\delta_y \Phi^n)^T \bar{y}^n, \qquad n = N, \ldots, 1$$

$$\bar{y}^{N+1} = 0$$

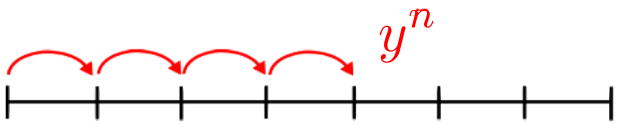
3. Design equation:

$$\sum_{n=1}^{N} \left(\nabla_u J^n + (\delta_u \Phi^{n-1})^T \bar{y}^{n-1} \right) = 0$$

Nested Optimization Approach

Initial design u_i
For $i = 1, 2, \ldots$

1. State equations **solve**:

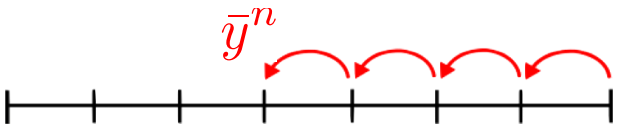


Time Parallel!

$$y^n = \Phi(y^{n-1}, u_i), \qquad n = 1, \ldots, N$$

$$y^0 = g$$

2. Adjoint equations **solve**:



Time Parallel!

$$\bar{y}^n = \nabla_{y^n} J^n + (\delta_y \Phi^n)^T \bar{y}^n, \qquad n = N, \ldots, 1$$

$$\bar{y}^{N+1} = 0$$

3. Design **update**:

$$u_{i+1} = u_i - B_i^{-1} \left(\sum_{n=1}^{N} (\nabla_u J^n + (\delta_u \Phi^{n-1})^T \bar{y}^n) \right)$$

XBraid-adjoint: three new wrapper routines of existing user code to obtain time parallelism

- Solve for $\mathbf{y},\ \bar{\mathbf{y}},\ \bar{u} := (\delta J/\delta u)^T$ *(reduced gradient of J w.r.t. design u)*

XBraid-Adjoint

1. `ObjectiveT`: $J(y^n, u)$

2. `Step_diff`: $\bar{y}^n = (\delta_y \Phi^n)^T \bar{y}^{n+1}$

   $\bar{u} \mathrel{+}= (\delta_u \Phi^n)^T \bar{y}^{n+1}$

3. `ObjectiveT_diff`: $\bar{y}^n \mathrel{+}= \nabla_{y^n} J^n$

   $\bar{u} \mathrel{+}= \nabla_u J^n$

Iteration k of XBraid-adjoint:

$$\bar{\mathbf{y}}_{\mathbf{k+1}} \leftarrow \mathrm{XBraid\_adjoint}(\mathbf{y}_k, \bar{\mathbf{y}}_{\mathbf{k}}, u_k)$$

$$\bar{u} \leftarrow \frac{\delta J(\mathbf{y}_k, u_k)}{\delta u}$$

Functions 2 and 3 allow XBraid to compute

$$\bar{y}^n = \nabla_{y^n} J^n + (\delta_y \Phi^n)^T \bar{y}^{n+1}$$

$$u_{i+1} = u_i - B_i^{-1} \left(\sum_{n=1}^{N} (\nabla_u J^n + (\delta_u \Phi^{n-1})^T \bar{y}^n) \right)$$

Reduced gradient: $\bar{u}$

XBraid example: ex-01-adjoint.c

- `Step_diff()`: $\bar{y}^n = (\delta_y \Phi^n)^T \bar{y}^{n+1}$

$$\bar{u} \mathrel{+}= (\delta_u \Phi^n)^T \bar{y}^{n+1}$$

```
270 int
271 my_Step_diff(braid_App              app,
272                   braid_Vector       y,
273                   braid_Vector       y_bar,
274                   braid_StepStatus   status)
275 {
```

...

```
285    /* Get the design from the app */
286    double lambda = app->design;
287
288    /* Transposed derivative of step wrt y times y_bar */
289    ddy = 1./(1. - lambda * deltat) * (y_bar->value);
290
291    /* Transposed derivative of step wrt design times y_bar */
292    ddesign = (deltat*(y->value)) / pow(1 - deltat*lambda,2) * (y_bar->value);
293
294    /* Update y_bar and gradient */
295    y_bar->value      = ddy;
296    app->gradient    += ddesign;
```

XBraid example: `ex-01-adjoint.c`

- `ObjectiveT()` and `ObjectiveT_diff()` are similar
- Initialize and run XBraid-adjoint:

```
343    /* Initialize XBraid */
344    braid_Init( <insert function pointers> );
345
346   /* Initialize adjoint-based gradient computation */
347    braid_InitAdjoint( <insert function pointers> );
```

...

```
359    braid_SetAbsTol(core, 1e-6);              /* Tolerance on state residual norm */
360    braid_SetAbsTolAdjoint(core, 1e-6);       /* Tolerance on adjoint residual norm *
361
362
363    /* Run simulation and adjoint-based gradient computation */
364    braid_Drive(core);
365
366    /* Get the objective function value from XBraid */
367    braid_GetObjective(core, &objective);
368
369    /* Collect sensitivities from all processors */
370    double mygradient = app->gradient;
371    MPI_Allreduce(&mygradient, &(app->gradient), 1, MPI_DOUBLE, MPI_SUM, comm);
```

$$\bar{u} \leftarrow (\delta J/\delta u)^T$$

Summary and Conclusions

- Parallel time integration is needed on future architectures
  - Major paradigm shift for computational science!
- MGRIT algorithm extends multigrid reduction "in time"
  - Non-intrusive yet flexible approach (open-source code XBraid)
- MGRIT approach is showing promise in a variety of settings
  - Adaptivity in space and time, moving meshes, BDF methods, …
  - Linear/nonlinear diffusion, advection, fluids, power grid, elasticity, …
  - Coupling to codes: MFEM, hypre, Strand2D, Cart3D, LifeV, CHeart, GridDyn
- There is much future work to be done!
  - More problem types, more complicated discretizations, performance improvements, adaptive meshing, ...

CASC

Center for Applied
Scientific Computing

Thank You!

Lawrence Livermore
National Laboratory