Multigrid Reduction in Time: A Flexible and Non-Intrusive Method

Jacob Schroder (LLNL)

R. Falgout, T. Kolev, V. Dobrev, U. Yang, N. A. Petersson (LLNL)
S. Friedhoff, S. MacLachlan (Tufts)

3rd Workshop on Parallel-in-Time Integration
May 27, 2014

Lawrence Livermore National Laboratory

LLNL-PRES-654654

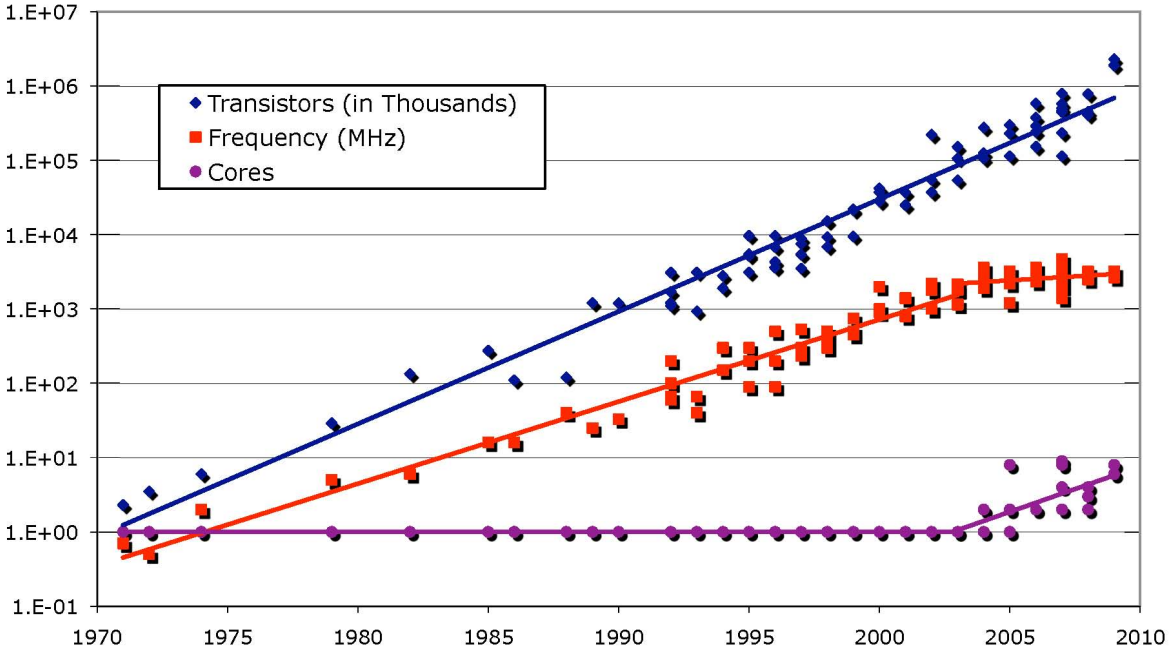
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Outline

- Motivation and overview of multigrid reduction in time (MGRIT)
- Flexible approach
  - Non-intrusiveness
  - Temporal refinement
  - Spatial coarsening
  - Multistep methods
- Application: compressible Navier-Stokes
- Conclusions

Motivation: Time integration will become a significant sequential bottleneck

- Clock speeds are no longer increasing – faster speed will be achieved through more concurrency

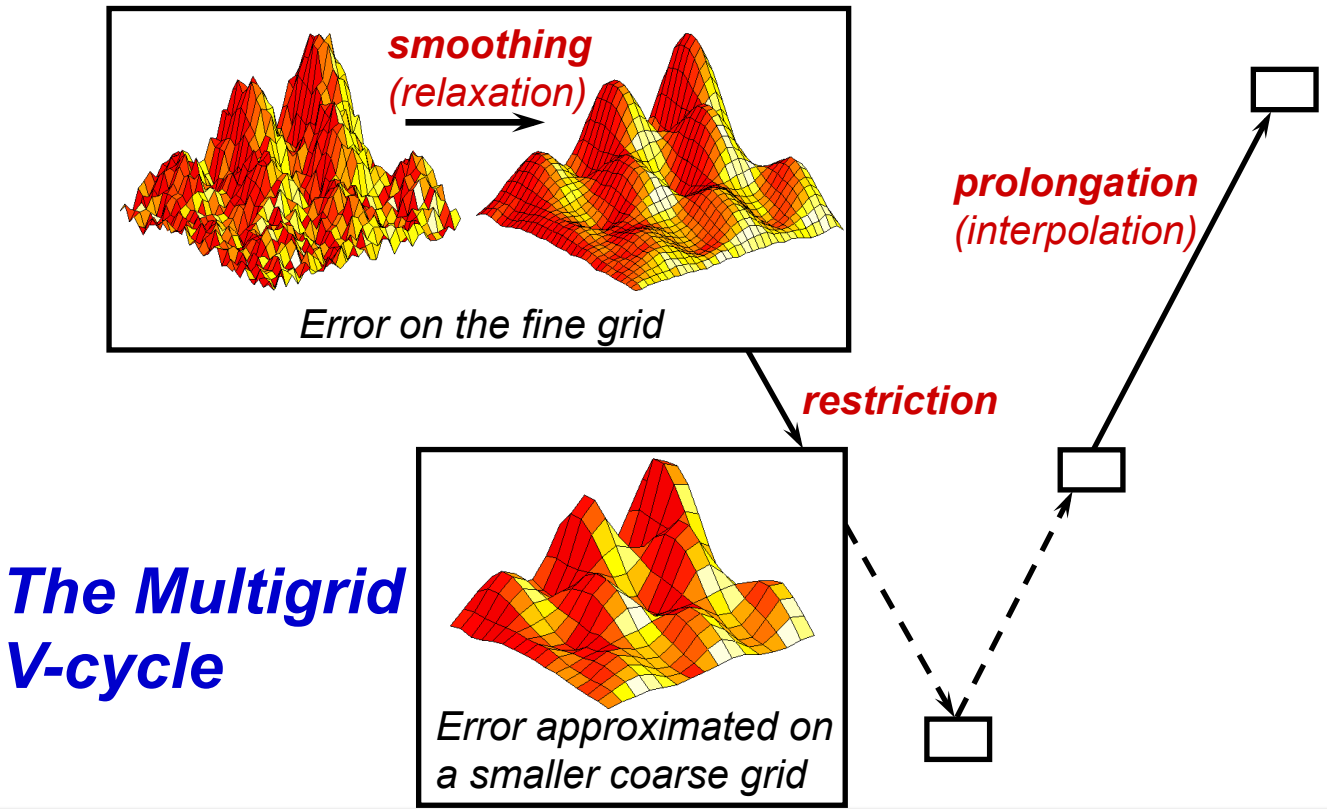


From Kathy Yelick's talk titled "Ten Ways to Waste a Parallel Computer." Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanović

- Parallel time integration methods are necessary, not just at LLNL

Our approach for parallel-in-time

- Leverage wealth of research on parallel spatial multigrid (MG)
- Our team has extensive spatial MG experience (*hypre* project)



Our approach

- Consider the **general** one-step method

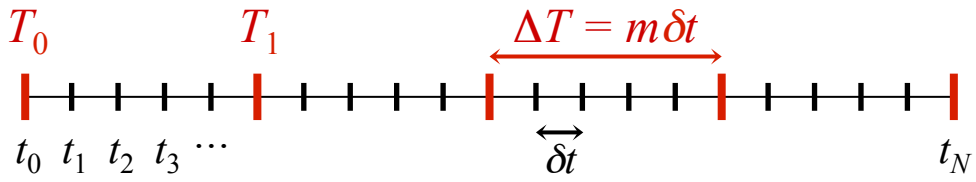
$$\boldsymbol{u}_i = \Phi_i(\boldsymbol{u}_{i-1}) + \boldsymbol{g}_i, \quad i = 1, 2, ..., N$$

- In the linear setting *(for simplicity)*, time marching $\equiv$ forward solve
  - This is an $O(N)$ direct method, **but sequential**

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

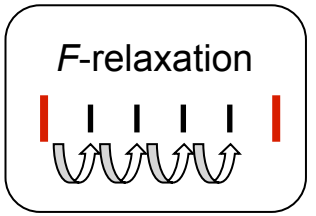
- We propose solving this system **iteratively** with a multigrid method
  - Extend multigrid reduction (MGR, 1979) to the time dimension
  - Coarsens only in time (non-intrusive)
  - $O(N)$, highly parallel

The two-grid method



– F-point (fine grid only)
— C-point (form coarse grid)

- Relaxation is highly parallel
  - Alternates between F-points and C-points
  - F-point relaxation = integration over each coarse time interval



- Coarse-grid Petrov-Galerkin system gives exact solution at C-points

Fine System

$$A\mathbf{u} = \mathbf{g}$$

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

Coarse Petrov-Galerkin System

$$A_\Delta u_\Delta = g_\Delta$$

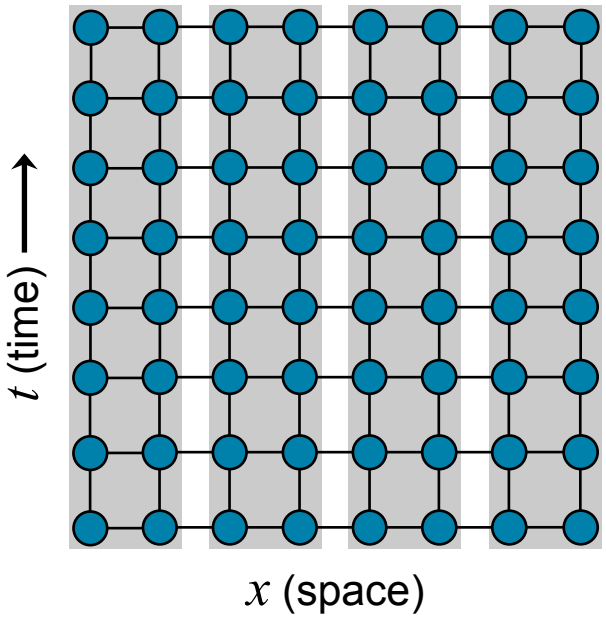
$$A_\Delta = \begin{pmatrix} I & & & \\ -\Phi^m & I & & \\ & \ddots & \ddots & \\ & & -\Phi^m & I \end{pmatrix}$$

- Replace impractical Φ^m with Φ_Δ, a rediscretization with ΔT

Parallel decomposition

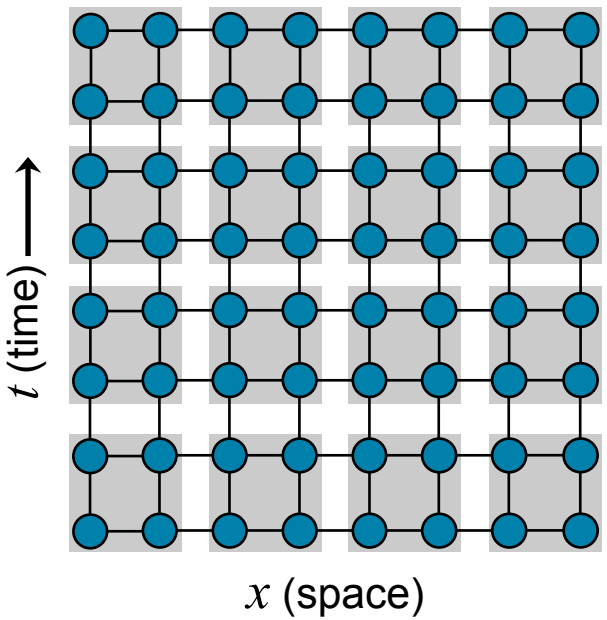
- Our code (***Warp)*** is agnostic to spatial decomposition and only parallelizes in time

Serial time stepping



- Parallelize in space only
- Store only one time step

Multigrid-in-time



- Parallelize in space and time
- Store several time steps

Properties of Multigrid Reduction in Time (MGRIT)

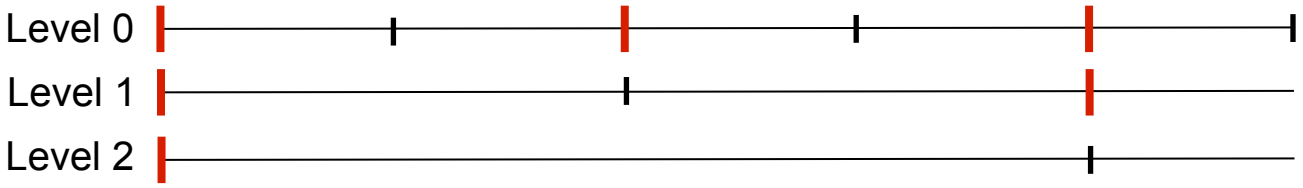
- Full approximation scheme (FAS) formulation for nonlinear problems
- Non-intrusive approach with unchanged time discretization
  - User provides time integrator Φ
- MGRIT is optimal for simple parabolic problems (implicit and explicit)
- In practice, store and solve one space-time slab at a time
- MGRIT with F-relaxation and two levels is equivalent to parareal
  - Two levels still requires a significant sequential solve
  - Multigrid perspective proved useful for achieving the additional parallelism of a full multilevel method without sacrificing optimality
- Parallel time integration is only useful beyond some scale
  - There is a crossover point, but we have already observed speedups around 10x

Flexible framework: non-intrusive

- User defines two objects:
  - *App* and *Vector*
- User also writes several wrapper routines:
  - *Phi*, *Init*, *Clone*, *Free*, *Sum*, *Dot*, *Write*, *BufPack*, *BufUnpack*
  - *Coarsen*, *Restrict* (optional, for spatial coarsening)
- *Phi(app, tstart, tstop, accuracy, u, &rfactor)*
  - Advances vector *u* from time *tstart* to *tstop*
  - Return value *rfactor* specifies a requested temporal refinement factor
- Code stores only *C*-points to minimize storage
- Consider relaxation over a processor's portion of the time interval
  - Each proc starts with right-most interval to overlap comm/comp

Flexible framework: Adaptivity in time

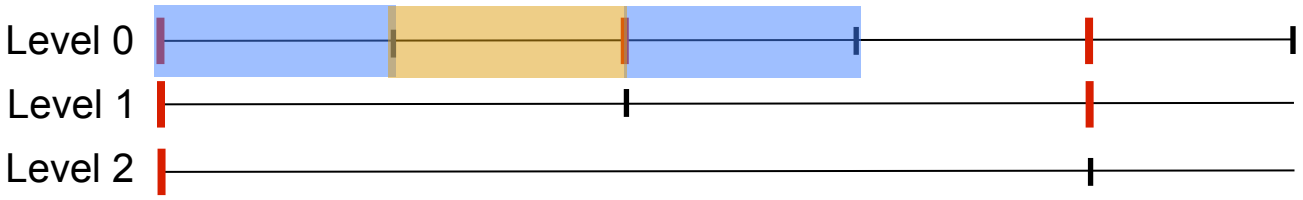
- In Phi, user returns rfactor, indicating whether to subdivide the interval
- Example time domain
  - F-point (fine grid only)
  - C-point (coarse grid)



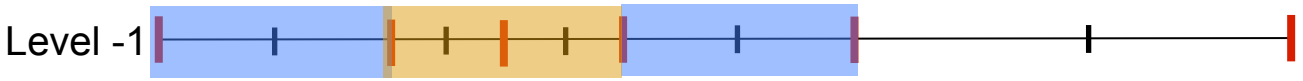
This aspect is a work in progress and is only partially implemented.

Flexible framework: Adaptivity in time

- In Phi, user returns rfactor, indicating whether to subdivide the interval
- Example time domain
  - F-point (fine grid only)
  - C-point (form coarse grid)

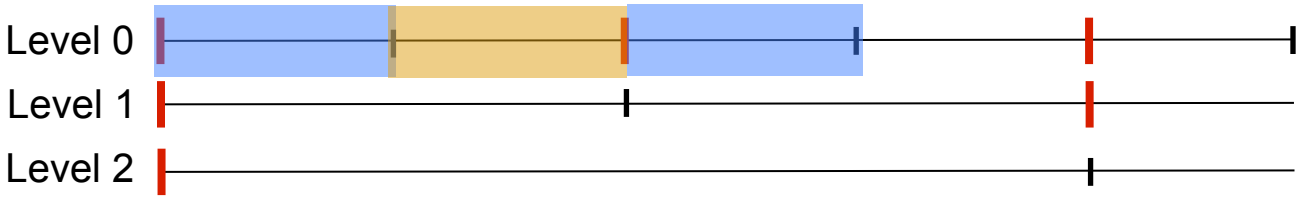


- Refine by 2 in intervals 1 and 3
- Refine by 4 in interval 2

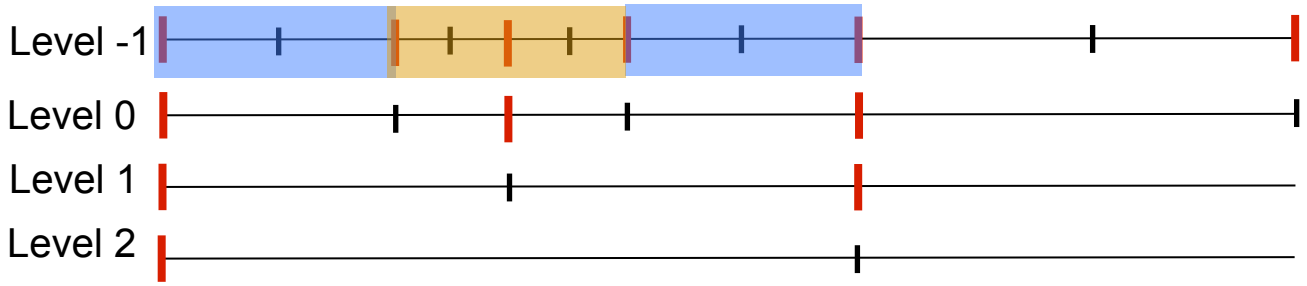


Flexible framework: Adaptivity in time

- In Phi, user returns rfactor, indicating whether to subdivide the interval
- Example time domain
  - F-point (fine grid only)
  - C-point (form coarse grid)



- Refine by 2 in intervals 1 and 3
- Refine by 4 in interval 2

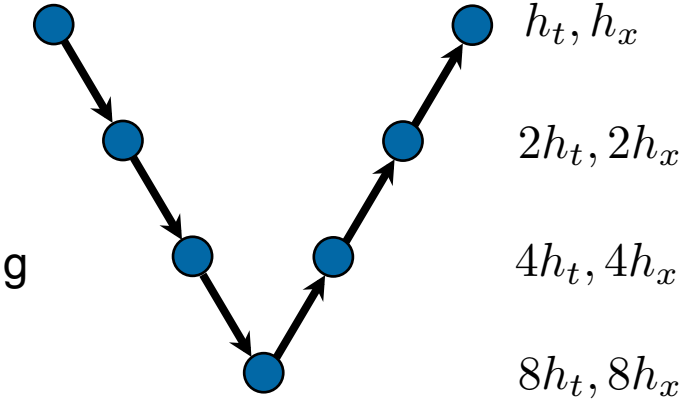


New hierarchy built from Level -1

Flexible framework

Vary spatial resolution

- User can define spatial coarsening and refinement routines
- Can allow for stable explicit time stepping on coarse temporal meshes
- This is also a work in progress
  - Success so far primarily for finite differencing

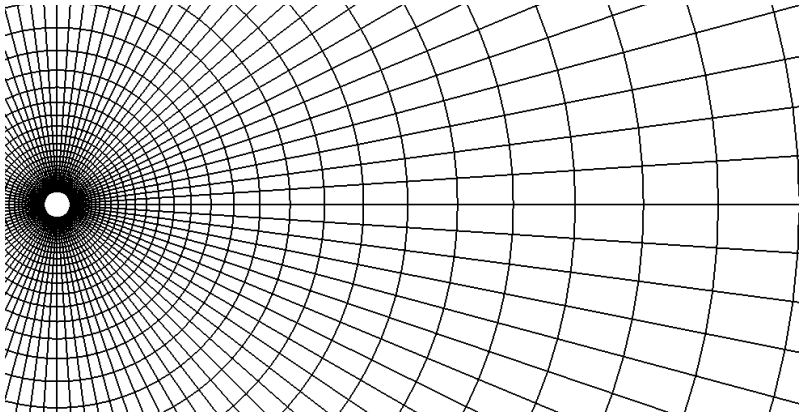


Multistep methods

- Results in more subdiagonal(s) for the space-time system
- Multiple options
  - Group time points together to form a “node”
  - Switch to lower-order methods on coarse levels

Application: Compressible Navier-Stokes

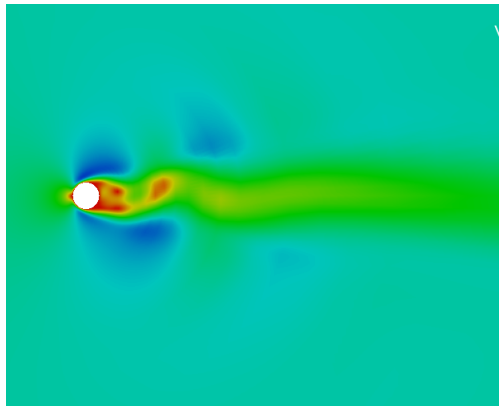
- Use the serial Strand2D code***
  - Solves compressible Navier-Stokes (nonlinear)
  - Problem is unsteady vortex shedding over a cylinder
  - Implicit time stepping and a spatial FAS multigrid cycle for implicit solves
    - Our tests use backward Euler
  - 3rd-order finite-differencing on “Strand Grids” for efficiency and accuracy



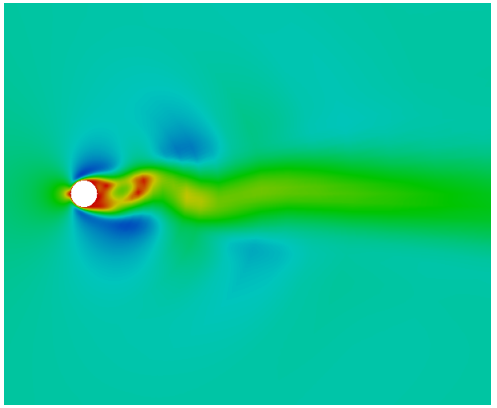
*** A. J. Katz and D. Work, “High-order flux correction/finite difference schemes for strand grids,” 52nd Aerospace Sciences Meeting, American Institute of Aeronautics and Astronautics, Jan. 2014.

Application: Compressible Navier-Stokes

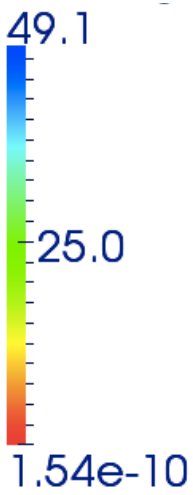
- Plot velocity magnitude
- Solution snapshots exhibiting unsteady vortex shedding

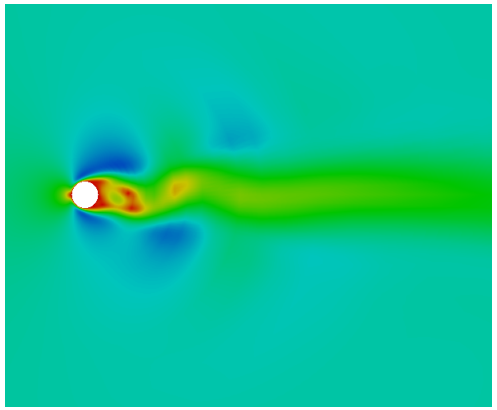


$t = 0.6875$

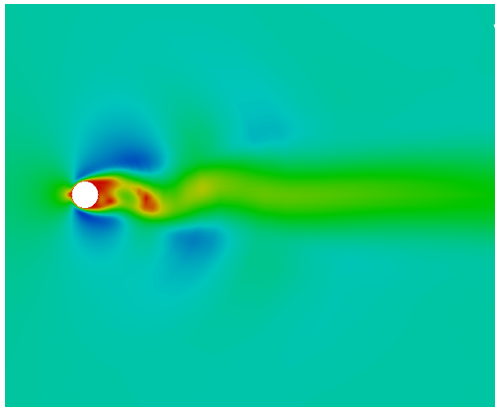


$t = 0.725$





$t = 0.7625$



$t = 0.8$

Application: Compressible Navier-Stokes

- Strand2D Code: approximately 13,500 lines
  - Not counting library code like LAPACK
- We added 129 lines to Strand2D
  - 20 lines to facilitate file output in parallel
  - 109 lines to enable restarting the code at a new time for a new state vector
  - With little outside help, this took about 3-4 weeks
  - If restarting had already been enabled, this would have been **much shorter**
- **Warp**: our MGRIT code
- Warp wrapper code is about 475 lines
  - Includes main(), command line parsing, etc...
  - Important code is much shorter

*** A. J. Katz and D. Work, “High-order flux correction/finite difference schemes for strand grids,” 52nd Aerospace Sciences Meeting, American Institute of Aeronautics and Astronautics, Jan. 2014.

Warp vector definition

- Warp vector is simply a 3D array
  - Other routines like dot, sum, buffer pack are trivial

```
20 // Wrapper for WARP's Vector object
21 class WarpVector
22 {
23 public:
24    // Empty Constructor
25    WarpVector() {  }
26
27    // State vector
28    Array3D<double> vec;
29
30 };
```

Warp app definition

- Warp app holds a few key pieces of information

```
61 class WarpApp
62 {
63 public:
64     MPI_Comm comm_t;
```

Warp app definition

- Warp app holds a few key pieces of information

```
61 class WarpApp
62 {
63 public:
64    MPI_Comm comm_t;
65    string   in_file;
```

Warp app definition

- Warp app holds a few key pieces of information

```
61 class WarpApp
62 {
63 public:
64     MPI_Comm comm_t;
65     string   in_file;
66     double   dt;
67     double   tstart;
68     double   tstop;
69     int      nsteps;
```

Warp app definition

- Warp app holds a few key pieces of information

```
61 class WarpApp
62 {
63 public:
64    MPI_Comm comm_t;
65    string   in_file;
66    double   dt;
67    double   tstart;
68    double   tstop;
69    int      nsteps;
70    int      noutput;
71    int      buff_size;
72    int      state_vec_size;
73    int      nSurfNode, nStrandNode, nq;
```

Warp app definition

- Warp app holds a few key pieces of information

```
61 class WarpApp
62 {
63 public:
64    MPI_Comm comm_t;
65    string   in_file;
66    double   dt;
67    double   tstart;
68    double   tstop;
69    int      nsteps;
70    int      noutput;
71    int      buff_size;
72    int      state_vec_size;
73    int      nSurfNode, nStrandNode, nq;
74    WarpVector *q_init;
```

Warp app definition

- Warp app holds a few key pieces of information

```
61 class WarpApp
62 {
63 public:
64    MPI_Comm comm_t;
65    string   in_file;
66    double   dt;
67    double   tstart;
68    double   tstop;
69    int      nsteps;
70    int      noutput;
71    int      buff_size;
72    int      state_vec_size;
73    int      nSurfNode, nStrandNode, nq;
74    WarpVector *q_init;
75    Strand2dFCManager *manager;
```

Warp phi definition

```
167    static int Phi(warp_App     _app,
168                   double       tstart,
169                   double       tstop,
170                   double       accuracy,
171                   warp_Vector  _u,
172                   int         *rfactor_ptr)
```

Warp phi definition

```
167    static int Phi(warp_App     _app,
168                   double       tstart,
169                   double       tstop,
170                   double       accuracy,
171                   warp_Vector  _u,
172                   int         *rfactor_ptr)
173    {
174       // Initialize
175       WarpApp *app = (WarpApp*) _app;
176       WarpVector *u = (WarpVector*) _u;
177       double dt = tstop-tstart;
178       int step = round(tstop / dt);
179       *rfactor_ptr = 1;
180
```

Warp phi definition

```
167    static int Phi(warp_App     _app,
168                   double       tstart,
169                   double       tstop,
170                   double       accuracy,
171                   warp_Vector  _u,
172                   int         *rfactor_ptr)
173    {
174       // Initialize
175       WarpApp *app = (WarpApp*) _app;
176       WarpVector *u = (WarpVector*) _u;
177       double dt = tstop-tstart;
178       int step = round(tstop / dt);
179       *rfactor_ptr = 1;
180
181       // Reset Strand with new time step size and state vector
182       app->manager->w_resetQ(u->vec);
183       app->manager->w_resetDtUnsteady(dt);
184
```

Warp phi definition

```
167    static int Phi(warp_App     _app,
168                   double       tstart,
169                   double       tstop,
170                   double       accuracy,
171                   warp_Vector  _u,
172                   int         *rfactor_ptr)
173    {
174       // Initialize
175       WarpApp *app = (WarpApp*) _app;
176       WarpVector *u = (WarpVector*) _u;
177       double dt = tstop-tstart;
178       int step = round(tstop / dt);
179       *rfactor_ptr = 1;
180
181       // Reset Strand with new time step size and state vector
182       app->manager->w_resetQ(u->vec);
183       app->manager->w_resetDtUnsteady(dt);
184
185       // Carry out one time step (via many pseudo steps)
186       int  nPseudoSteps  = app->manager->getNPseudoSteps();
187       bool converged     = false;
188       for (int pseudoStep=0; pseudoStep<nPseudoSteps; pseudoStep++)
189       {
190          app->manager->takePseudoStep(step, pseudoStep, converged);
191          if (converged)
192             break;
193       }
```

Warp phi definition

```
167    static int Phi(warp_App     _app,
168                   double       tstart,
169                   double       tstop,
170                   double       accuracy,
171                   warp_Vector  _u,
172                   int         *rfactor_ptr)
173    {
174       // Initialize
175       WarpApp *app = (WarpApp*) _app;
176       WarpVector *u = (WarpVector*) _u;
177       double dt = tstop-tstart;
178       int step = round(tstop / dt);
179       *rfactor_ptr = 1;
180
181       // Reset Strand with new time step size and state vector
182       app->manager->w_resetQ(u->vec);
183       app->manager->w_resetDtUnsteady(dt);
184
185       // Carry out one time step (via many pseudo steps)
186       int  nPseudoSteps  = app->manager->getNPseudoSteps();
187       bool converged     = false;
188       for (int pseudoStep=0; pseudoStep<nPseudoSteps; pseudoStep++)
189       {
190          app->manager->takePseudoStep(step, pseudoStep, converged);
191          if (converged)
192             break;
193       }
194
195       // Save state vector from Strand
196       app->manager->w_getQ(u->vec);
197
198       return 0;
199    }
```

Test Warp similarly to a standard MG scaling study

- Fix time domain with $t_{final} = 2.56s$, then refine in time
- Linux cluster: Intel Sandybridge, InfiniBand QDR interconnect
- Warp: F-cycles, FCF relaxation, coarsen by 5, relative tol of 1e-5
- Strand: 24,960 d.o.f. spatial mesh
  - 1280 time step case barely resolves unsteady behavior
- Strouhal number indicates shedding frequency at a point in wake
  - Shows that the smaller time step sizes are worth running

Num steps	nprocs	Warp iters	Strouhal #	Run time	Speedup
1280	256	21	0.126	73 min	0.64
5120	1024	15	0.140	81 min	2.63
20480	4096	14	0.112	116 min	7.53

Test Warp similarly to a standard MG scaling study

- Fix time domain with $t_{final} = 2.56s$, then refine in time
- If dt is held constant and this experiment is run out to 5120 time steps
  - The iteration count increases to 58.
  - t_{final} increases from $2.56s$ to $10.24s$
  - For this larger time domain, some increase in iterations is expected
  - 1280 time step case barely resolves unsteady behavior
- Strouhal number indicates shedding frequency at a point in wake
  - Shows that the smaller time step sizes are worth running

Num steps	nprocs	Warp iters	Strouhal #	Run time	Speedup
1280	256	21	0.126	73 min	0.64
5120	1024	15	0.140	81 min	2.63
20480	4096	14	0.112	116 min	7.53

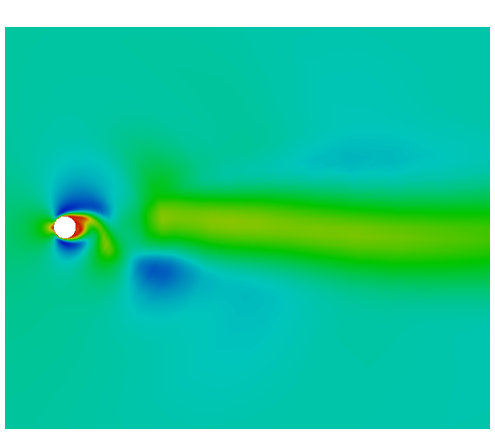
Test Warp similarly to a standard MG scaling study

- Fix time domain with $t_{final} = 2.56s$, then refine in time
- Linux cluster: Intel Sandybridge, InfiniBand QDR interconnect
- If dt is held constant and this experiment is run out to 20480 time steps
  - The iteration count increases to 45.
  - t_{final} increases from $2.56s$ to $10.24s$
  - For this larger time domain, some increase in iterations is expected
- Strouhal number indicates shedding frequency at a point in wake
  - Shows that the smaller time step sizes are worth running

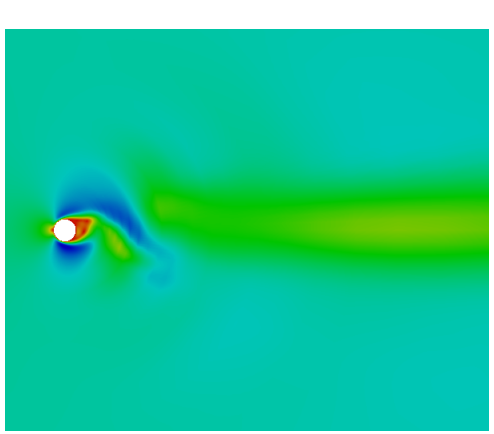
Num steps	nprocs	Warp iters	Strouhal #	Run time	Speedup
1280	256	21	0.126	73 min	0.64
5120	1024	15	0.140	81 min	2.63
20480	4096	14	0.112	116 min	7.53

Warp convergence at final time

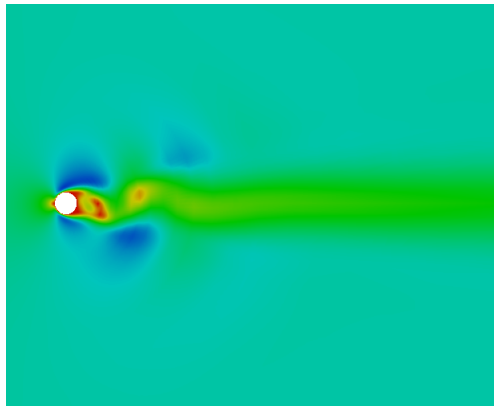
- Plot velocity magnitude
- 5120th time step ($t=2.56s$)
- After 13 Warp iterations, accuracy is good



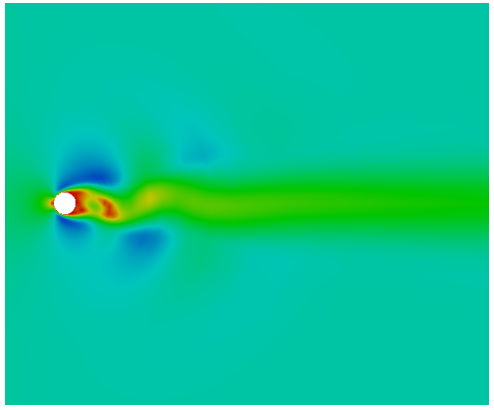
Iteration 1



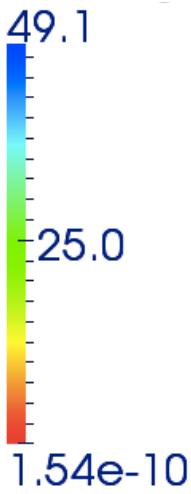
Iteration 5



Iteration 9



Iteration 13



Conclusions and future work

Conclusions

- MGRIT is a flexible and non-intrusive parallel-in-time approach
  - Temporal refinement, spatial coarsening and multistep methods
  - Already shown some success wrapping a compressible Navier-Stokes code

Future work

- Leverage the periodic part of the solution
  - Hopefully, slow the iteration growth as t_{final} increases
- Explore optimizations and improve speedup
  - Use cheaper coarse-level time stepping
- Explore BDF2

Thank You! Any Questions?

References

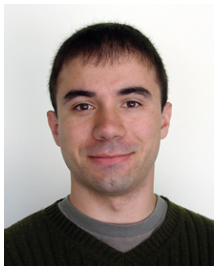
[1] R. Falgout, S. Friedhoff, Tz. Kolev, S. MacLachlan, and J. Schroder, *Parallel Time Integration with Multigrid*, SIAM J. Sci. Comput., (submitted).

http://computation.llnl.gov/casc/linear_solvers/pubs.html

Our Team

Veselin Dobrev

Rob Falgout



Tzanio Kolev

Anders Petersson

Jacob Schroder

Ulrike Yang

LLNL

Stephanie Friedhoff

Scott MacLachlan

Tufts University

Lawrence Livermore
National Laboratory