Parallel-in-Time Optimization with the General-Purpose XBraid Package

Stefanie Günther, Jacob Schroder, Robert Falgout, Nicolas Gauger

7th Workshop on Parallel-in-Time methods
Wednesday, May 3rd, 2018

Lawrence Livermore National Laboratory

LLNL-PRES-750301
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: parallelizing optimization problems
- Parallel-in-time overview: Multigrid reduction in time (MGRIT)
- XBraid-adjoint code interface : Open source implementation
  - Non-intrusively uses existing user code
  - Example: user-interface for simple scalar ODE problem

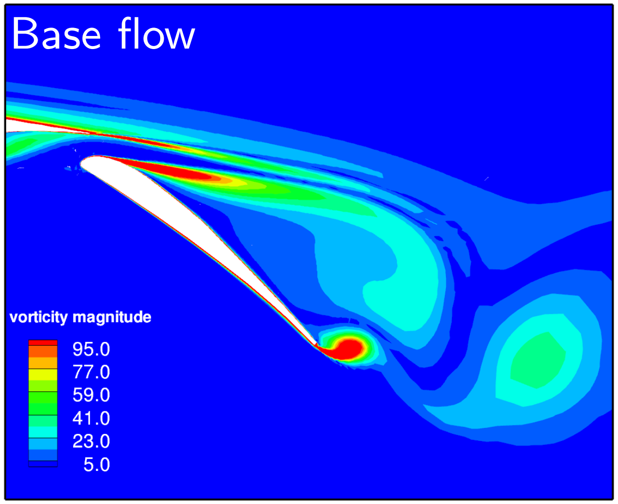
Motivation: PDE constrained optimization

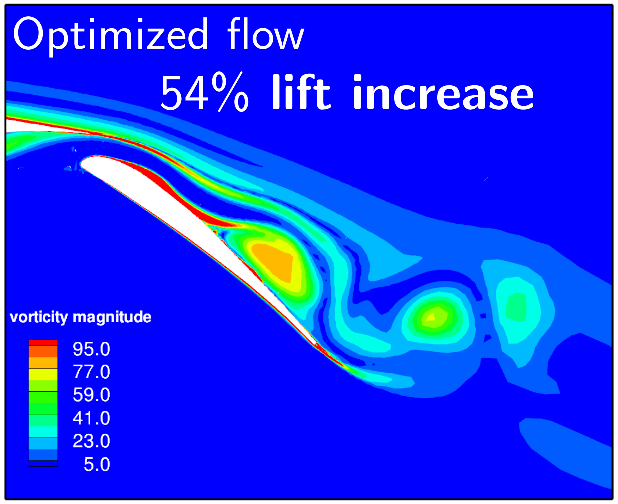
Example[1]

- 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 object 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 object 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:

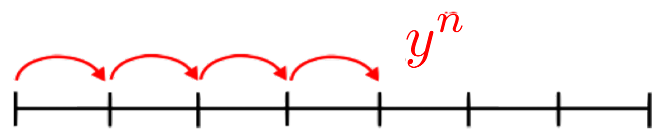
$$\sum_{n=1}^{N} \left(\nabla_u J^n + (\delta_u \Phi^{n-1})^T \bar{y}^{n-1}\right) = 0$$

Nested Optimization Approach

Initial design u_i

For $i = 1, 2, \ldots$

1. State equations **solve**:

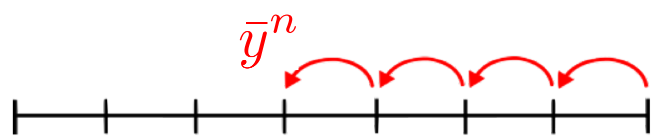


Time Parallel!

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

$$y^0 = g$$

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**:

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

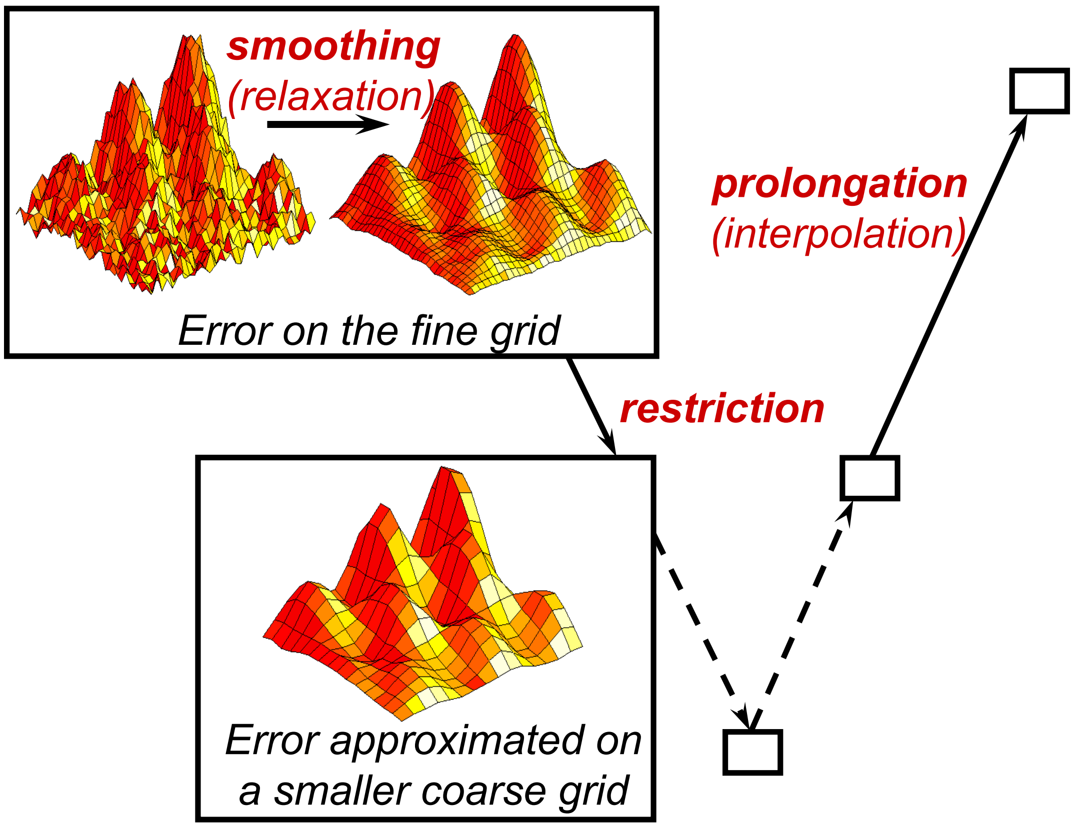
Outline

- Motivation: parallelizing optimization problems
- Parallel-in-time overview: Multigrid reduction in time (MGRIT)
- XBraid-adjoint code interface : Open source implementation
  - Non-intrusively uses existing user code
  - Example: user-interface for simple scalar ODE problem

Parallel-in-time overview: Approach leverages spatial multigrid research

- **Motivation:** Clock speeds are stagnate but speedup still possible through more concurrency
  → Parallel-in-time is needed

The Multigrid V-cycle



Multigrid reduction in time (MGRIT)

- General one-step method for a **forward** evolution process

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

- In the linear setting *(for simplicity)*, sequential marching $\equiv$ forward solve

$$A\mathbf{y} \equiv \begin{pmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{pmatrix} \begin{pmatrix} y^0 \\ y^1 \\ \vdots \\ y^N \end{pmatrix} = \begin{pmatrix} g^0 \\ g^1 \\ \vdots \\ g^N \end{pmatrix} \equiv \mathbf{g}$$

- We solve this system **iteratively** with multigrid reduction *(MGR 1979)*
  - Replace sequential $O(N)$ method with $O(N)$ parallel alternative
  - Coarsens only in *time* → non-intrusive, i.e. Φ is "arbitrary"

Multigrid reduction in time (MGRIT)

- General one-step method for a **backward** evolution process

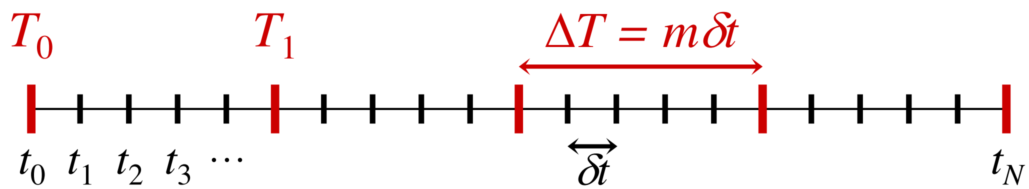
$$\bar{y}^n = \nabla_{y^n} J^n + (\delta_y \Phi^n)^T \bar{y}^n, \qquad n = N, \ldots, 1$$

- In the linear setting *(for simplicity)*, sequential marching $\equiv$ forward solve

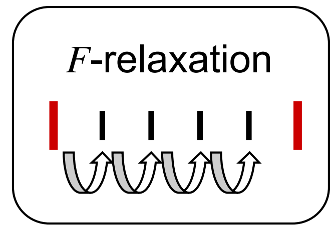
$$\bar{A}\bar{\mathbf{y}} \equiv \begin{pmatrix} I & -\Phi^T & & & \\ & I & -\Phi^T & & \\ & & \ddots & \ddots & \\ & & & I & -\Phi^T \end{pmatrix} \begin{pmatrix} \bar{\boldsymbol{y}}^0 \\ \bar{\boldsymbol{y}}^1 \\ \vdots \\ \bar{\boldsymbol{y}}^N \end{pmatrix} = \begin{pmatrix} \nabla_{y^0} J^0 \\ \nabla_{y^1} J^1 \\ \vdots \\ \nabla_{y^N} J^N \end{pmatrix}$$

- We solve this system **iteratively** with multigrid reduction *(MGR 1979)*
  - Replace sequential $O(N)$ method with $O(N)$ parallel alternative
  - Coarsens only in *time* → non-intrusive, i.e. Φ is "arbitrary"

MGRIT for forward solve



- F-point (fine grid only)
- C-point (coarse & fine grid)



- Relaxation is highly parallel
  - Alternates between F-points and C-points
  - F-relaxation = block Jacobi on each coarse time interval
- Coarse system approximates fine system
  - Approximate impractical Φ^m with Φ_Δ (e.g., time rediscretization with ΔT)

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

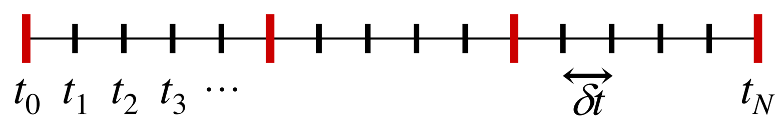
  - Apply recursively for multilevel hierarchy

A broader summary of MGRIT

- Expose **concurrency** in the evolution dimension with multigrid
- **Non-intrusive**, with unchanged fine-grid problem
- Converges to **same solution** as sequential marching

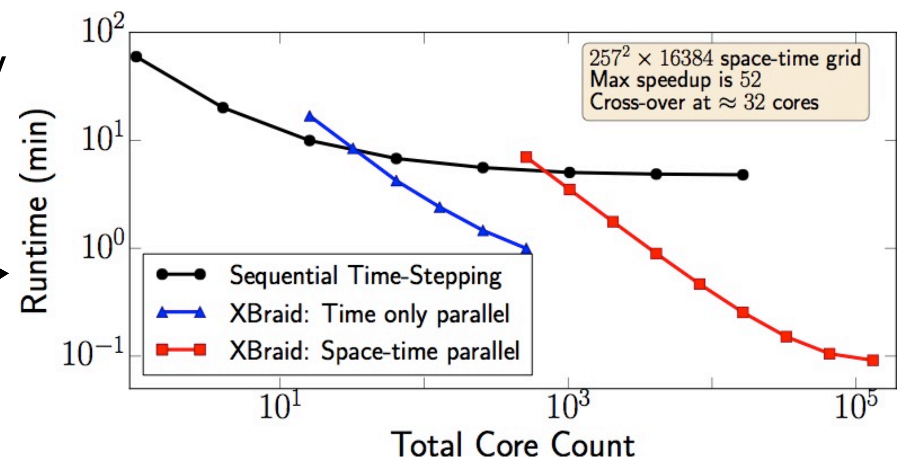
$$\begin{pmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{pmatrix}$$

- Only store ***C*-points** to minimize storage
- Optimal for variety of parabolic problems
  - Converges in ~10 iterations for any coarsening factor



- Extends to **nonlinear** problems with FAS formulation
- In specialized two-level setting, MGRIT $\equiv$ Parareal

- Large speedups available, but in a new way
  - Time stepping is already O(N)
  - Useful only beyond a crossover
  - More time steps → more speedup potential
  - Example strong scaling of the 2D heat eqn →



- Much future work to do...

Outline

- Motivation: parallelizing optimization problems

- Parallel-in-time overview: Multigrid reduction in time (MGRIT)

- XBraid-adjoint code interface : Open source implementation
  - Non-intrusively uses existing user code
  - Example: user-interface for simple scalar ODE problem

XBraid-adjoint: open source & non-intrusive

- Solve for $\mathbf{y}, \ \bar{\mathbf{y}}, \ \bar{u} := (\delta J / \delta u)^T$ *(reduced gradient of J w.r.t. design u)*
- Wrap existing user code to obtain time parallelism

Standard XBraid: forward in time solve

1. **Step:** $y^n = \Phi(y^{n-1}, u)$
2. Clone
3. Sum: $y^m = \alpha y^n + \beta y^m$
4. SpatialNorm: $\|y^n\|$
5. Buf[Un]Pack
6. Init
7. Free
8. Access

Iteration k of XBraid:

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

XBraid-adjoint: open source & non-intrusive

- Solve for $\mathbf{y}, \quad \bar{\mathbf{y}}, \quad \bar{u} := (\delta J/\delta u)^T$ *(reduced gradient of J w.r.t. design u)*
- Wrap existing user code to obtain time parallelism

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`

- Solve for the reduced gradient $\bar{u} := (\delta J / \delta u)^T$ *(later use in optimization)*
- Begin with simple problem: $J(y, \lambda) = 1/T \int_0^T \|y\| dt$

$$\text{s.t.} \qquad y_t = \lambda y$$
$$y(0) = 1$$

- First, user must define objects: `App` and `Vector`

```
typedef struct _braid_App_struct
{
   int         rank;
   double      design;
   double      gradient;

} my_App;

typedef struct _braid_Vector_struct
{
   double value;
} my_Vector;
```

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

- `Step()`: $y^n = \Phi(y^{n-1}, u)$

```
int
my_Step(braid_App        app,
        braid_Vector     ystop,
        braid_Vector     fstop,
        braid_Vector     y,
        braid_StepStatus status)
{
```

...

```

   /* Get the design variable from the app */
   double lambda = app->design;

   /* Use backward Euler to propagate solution */
   (y->value) = 1./(1. - lambda * (tstop-tstart))*(y->value);

   return 0;
```

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

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

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

```c
int
my_Step_diff(braid_App              app,
                braid_Vector        y,
                braid_Vector        y_bar,
                braid_StepStatus    status)
{
```

...

```c
   /* Get the design from the app */
   double lambda = app->design;

   /* Transposed derivative of step wrt y times y_bar */
   ddy = 1./(1. - lambda * deltat) * (y_bar->value);

   /* Transposed derivative of step wrt design times y_bar */
   ddesign = (deltat*(y->value)) / pow(1 - deltat*lambda,2) * (y_bar->value);

   /* Update y_bar and gradient */
   y_bar->value       = ddy;
   app->gradient     += ddesign;
```

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

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

```
    /* Initialize XBraid */
    braid_Init( <insert function pointers> );

   /* Initialize adjoint-based gradient computation */
    braid_InitAdjoint( <insert function pointers> );
```

...

```
    braid_SetAbsTol(core, 1e-6);               /* Tolerance on state residual norm */
    braid_SetAbsTolAdjoint(core, 1e-6);        /* Tolerance on adjoint residual norm *


    /* Run simulation and adjoint-based gradient computation */
    braid_Drive(core);

    /* Get the objective function value from XBraid */
    braid_GetObjective(core, &objective);

    /* Collect sensitivities from all processors */
    double mygradient = app->gradient;
    MPI_Allreduce(&mygradient, &(app->gradient), 1, MPI_DOUBLE, MPI_SUM, comm);
```

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

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

- Run it!

```
schroder2@kullat:$ ./ex-01-adjoint

  Braid: Begin simulation, 50 time steps

  Braid:       || r ||       || r_adj ||      Objective
  Braid:---------------------------------------------
  Braid:    0  5.399995e-01  2.521756e-02  6.270806e-02
  Braid:    1  2.446737e-02  2.604047e-02  3.604351e-02
  Braid:    2  1.098316e-03  2.278004e-03  3.600016e-02
  Braid:    3  4.838178e-05  1.498949e-04  3.600000e-02
  Braid:    4  2.065806e-06  8.581917e-06  3.600000e-02
  Braid:    5  8.359592e-08  4.482596e-07  3.600000e-02

 Objective = 3.59999999999874e-02
 Gradient  = 1.92000799963229e-02
```

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

- XBraid-adjoint solver
  - Generically solves adjoint equations, backwards in time
  - Generically computes reduced gradients $\bar{u}$
    → Designed to work with many optimization methods!

- Example: `ex-01-optimization.c`
  - Implements simple reduced space optimization loop
    (This is a template for implementing your favorite optimization method)
  - Requires:
    - Global objective evaluation function (and derivative)
    - Design update function
    - Halting metric (e.g., gradient norm)
  - Again, the goal is to wrap existing user code!

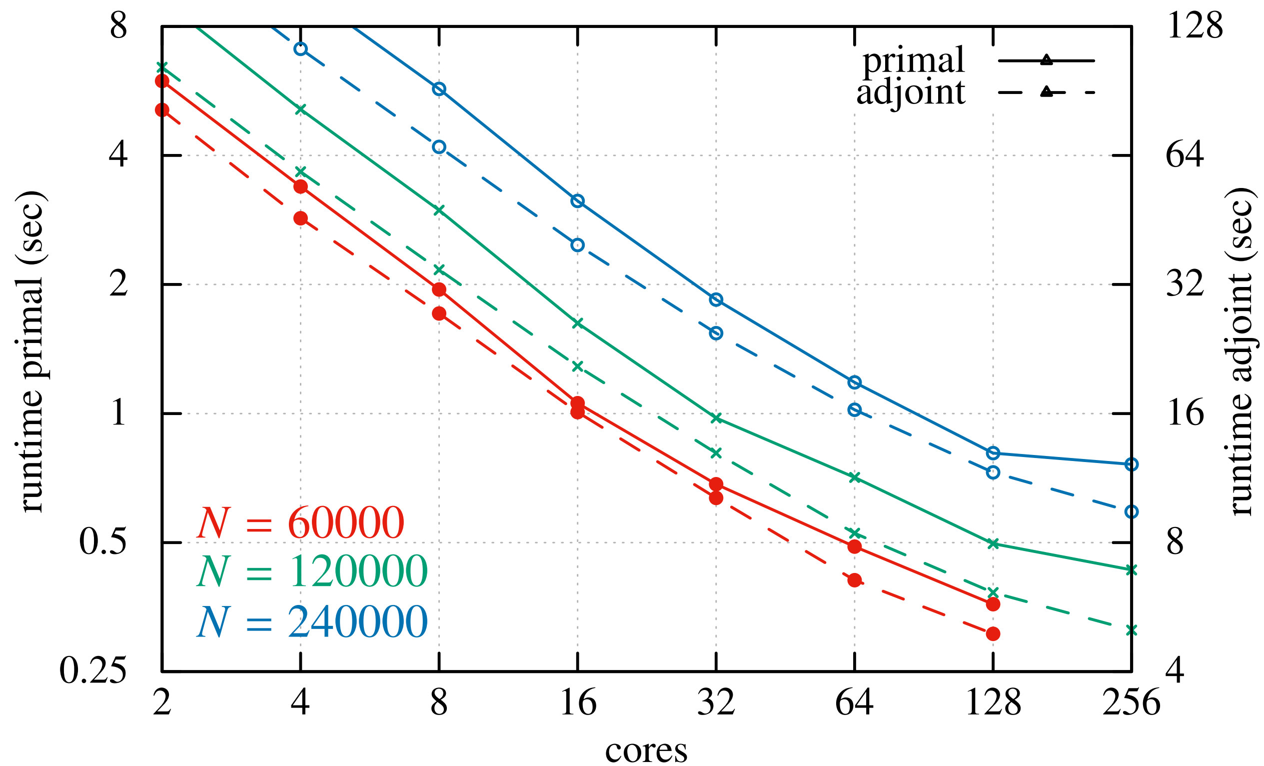
Model Problem: 1D advection-diffusion

- Advection dominated, with Van Der Pol oscillator on left boundary

$$y_t + y_x - \epsilon y_{xx} = 0, \quad \epsilon = 10^{-5}$$

- Tracking objective: Minimize difference of space-time averaged solution to preset value

- When used with one-shot strategies, the max speedup is 25x



Scaling of primal (solid lines) and adjoint (dashed lines) XBraid solvers.

Summary and Conclusions

- **Parallel time integration is needed** on future architectures, especially for adjoint-based optimization techniques

- **New XBraid-adjoint** non-intrusively couples with existing codes
  - Available in May, 2018 release
  - http://llnl.gov/casc/xbraid

- **References**

1. Günther, Gauger, and Schroder. *A Non-Intrusive Parallel-in-Time Approach for Simultaneous Optimization with Unsteady PDEs.* Optimization Methods and Software, (submitted), (2018).
2. Günther, Gauger, and Schroder. *A Non-Intrusive Parallel-in-Time Adjoint Solver with the XBraid Library.* CVS, Springer, (accepted), (2017).

Thank You! Any Questions?

A good read, *Parallel Time Integration with Multigrid*, SIAM J. Sci. Comp.

Open Source XBraid Code

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

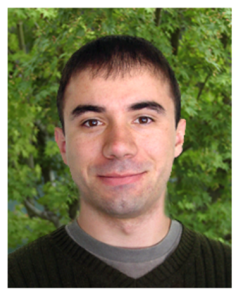
XBRAID PARALLEL MULTIGRID IN TIME

Our Team


Rob Falgout


Stefanie Günther


Matthieu Lecouvez


Tzanio Kolev


Jacob Schroder


Scott MacLachlan


Hans De Sterck


Stephanie Friedhoff

Collaborators

CU Boulder (Manteuffel, McCormick, O'Neill), Red Deer (Howse), U Stuttgart (Roehrle, Hessenthaler), Kaiserslautern (Günther, Gauger), LLNL (Woodward, Top)

Done