Multigrid perspectives on parallel-in-time integration

Scott MacLachlan
smaclachlan@mun.ca

Memorial University of Newfoundland

February 26, 2021

My Collaborators

My work in this area has been in collaboration with

- **Stephanie Friedhoff**, University of Wuppertal
- Rob Falgout, Tzanio Kolev, Lawrence Livermore National Laboratory
- Jacob Schroder, University of New Mexico
- Stefan Vandewalle, KU-Leuven
- Hans De Sterck, University of Waterloo
- Alex Howse, Verafin
- Oliver Krzysik, Monash University
- Federico Danieli, Oxford University

Acknowledgments

This research was supported by

- the National Science Foundation under grant DMS-1015370
- Lawrence Livermore National Laboratory under the auspices of the U.S. Department of Energy, under contract DE-AC52-07NA27344
- OPTEC (OPTimization in Engineering Center of excellence KU Leuven), which is funded by the KU Leuven Research Council under grant no. PFV/10/002.
- an ARC Discovery project
- the Natural Sciences and Engineering Research Council of Canada

My Philosophy on Parallel-in-Time

I'm not here to tell you how to discretize your problem.
I just want to solve it faster!

My Philosophy on Parallel-in-Time

I'm not here to tell you how to discretize your problem.
I just want to solve it faster!

Two reasons:

- Interesting to understand how a fixed parallel-in-time methodology performs with different discretizations of same problem
- Many real-world applications have carefully chosen spatial and temporal discretization schemes, and users are not interested in changing these (even to gain performance)

What is MGRIT?

Multigrid Reduction in Time

- Consider monolithic system arising from space-time discretization
- Write this as a block system, ordered by time levels
- Apply Multigrid Reduction techniques to its solution

Start simple:

- Consider ODE $x'(t) = a(t)x(t) + b(t)$, $x(0) = x_0$.
- Consider one-step method for finding $x_i \approx x(t_i)$

Bidiagonal Linear Systems

Need to solve the linear system

$$\begin{bmatrix} a_{11} & 0 & \cdots & & 0 \\ a_{21} & a_{22} & 0 & \cdots & \\ 0 & a_{32} & a_{33} & 0 & \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

Bidiagonal Linear Systems

Need to solve the linear system

$$\begin{bmatrix} a_{11} & 0 & \cdots & & 0 \\ a_{21} & a_{22} & 0 & \cdots & \\ 0 & a_{32} & a_{33} & 0 & \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

Standard forward solve algorithm: $x_k = (b_k - a_{k,k-1}x_{k-1})/a_{k,k}$

- 3 floating point operations per DoF
- Optimal, $\mathcal{O}(n)$ algorithm

Cyclic Reduction

Another approach is to chain two steps together:

$$
\begin{aligned}
x_k &= (b_k - a_{k,k-1}x_{k-1})/a_{k,k} \\
&= \left(b_k - a_{k,k-1}(b_{k-1} - a_{k-1,k-2}x_{k-2})/a_{k-1,k-1}\right)/a_{k,k} \\
&= \Big((b_k - a_{k,k-1}b_{k-1}/a_{k-1,k-1}) \\
&\qquad\qquad +(a_{k,k-1}a_{k-1,k-2}/a_{k-1,k-1})x_{k-2}\Big)/a_{k,k} \\
&= \left(\tilde{b}_k - \tilde{a}_{k,k-2}x_{k-2}\right)/a_{k,k}
\end{aligned}
$$

- Given x_1, use $\tilde{A}$, $\tilde{b}$ to solve for odd indices, then use original system to solve for even indices

Cost of Cyclic Reduction

Consider costs:

- Form $\tilde{b}_k = b_k - a_{k,k-1} b_{k-1} / a_{k-1,k-1}$, $n/2$ times
- Form $\tilde{a}_{k,k-2} = -a_{k,k-1} a_{k-1,k-2} / a_{k-1,k-1}$, $n/2$ times
- Solve system of size $n/2$ for odd indices
- Solve system of size $n/2$ for even indices

Total cost: $(3 + 2 + 3 + 3) \times n/2 = \mathbf{5.5n}$

Cost of Cyclic Reduction

Consider costs:

- Form $\tilde{b}_k = b_k - a_{k,k-1} b_{k-1} / a_{k-1,k-1}$, $n/2$ times
- Form $\tilde{a}_{k,k-2} = -a_{k,k-1} a_{k-1,k-2} / a_{k-1,k-1}$, $n/2$ times
- Solve system of size $n/2$ for odd indices
- Solve system of size $n/2$ for even indices

Total cost: $(3 + 2 + 3 + 3) \times n/2 = \mathbf{5.5n}$

Why use cyclic reduction when it costs more than forward solve?

Concurrency

The forward solve algorithm is fully sequential

- Must know x_{k-1} to solve for x_k

Cyclic Reduction allows parallelism:

- Can compute all entries $\tilde{b}_k$ and $\tilde{a}_{k,k-2}$ in parallel
- Sequential solve of size $n/2$ for odd indices
- Can compute solution at even indices in parallel

Concurrency

The forward solve algorithm is fully sequential

- Must know x_{k-1} to solve for x_k

Cyclic Reduction allows parallelism:

- Can compute all entries $\tilde{b}_k$ and $\tilde{a}_{k,k-2}$ in parallel
- Sequential solve of size $n/2$ for odd indices
- Can compute solution at even indices in parallel

Recursive cyclic reduction reduces sequential bottleneck

- Trade extra flops for better parallelism

Modern Computing Architectures

In the past decade, dominant computing architectures have switched from single-core to many-core

- Standard laptops have 2-8 cores
- Standard desktops have 8-32 cores
- GPUs provide thousands of cores
- Modern HPC architectures have 100,000's of cores

Modern Computing Architectures

In the past decade, dominant computing architectures have switched from single-core to many-core

- Standard laptops have 2-8 cores
- Standard desktops have 8-32 cores
- GPUs provide thousands of cores
- Modern HPC architectures have 100,000's of cores

Necessary paradigm shift:

- Total number of floating-point operations does not determine "cost"
- Must take possible concurrency into account

Accept algorithms with higher cost if they allow greater parallelism

From ODEs to PDEs

Now consider a first-order-in-time PDE

$$u_t + F(u) = 0 \text{ with } u(0, \mathrm{x}) = u_0(\mathrm{x})$$

Take a time-stepping perspective on space-time discretization and write single-step discretization as $\mathrm{u}_i = \Phi(\mathrm{u}_{i-1})$

If $F(u)$ is linear, then get block-structured linear system:

$$A\mathrm{u} \equiv \begin{bmatrix} I & & & \\ -\Phi_{\delta t} & I & & \\ & \ddots & \ddots & \\ & & -\Phi_{\delta t} & I \end{bmatrix} \begin{bmatrix} \mathrm{u}_0 \\ \mathrm{u}_1 \\ \vdots \\ \mathrm{u}_{N_t} \end{bmatrix} = \begin{bmatrix} \overline{\mathrm{u}}_0 \\ \mathrm{g}_1 \\ \vdots \\ \mathrm{g}_{N_t} \end{bmatrix} \equiv \mathrm{g}$$

Block Bidiagonal System

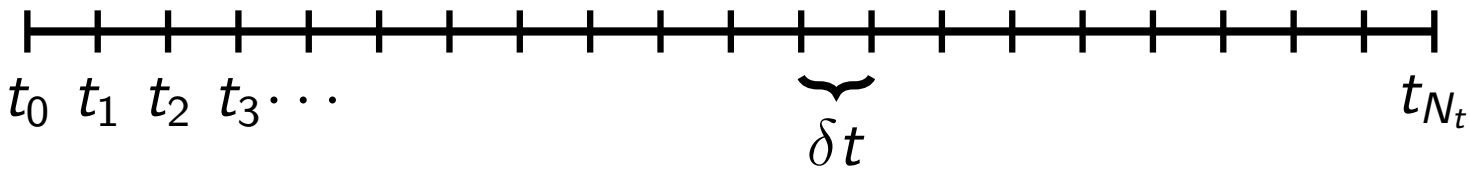
The space-time linear system, $A\mathrm{u} = \mathrm{g}$, is block lower bidiagonal

- Traditional time-stepping is a block forward sweep through the equations
- Could also solve the system using *Block Cyclic Reduction* (or *Block Factorization*)
  - Idea: split time-points into "C-points" and "F-points", then permute and factor

$$A = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix} = \begin{bmatrix} A_{ff} & 0 \\ A_{cf} & A_C \end{bmatrix} \begin{bmatrix} I & A_{ff}^{-1} A_{fc} \\ 0 & I \end{bmatrix}$$

for $A_C = A_{cc} - A_{cf} A_{ff}^{-1} A_{fc}$.

Schur Complement

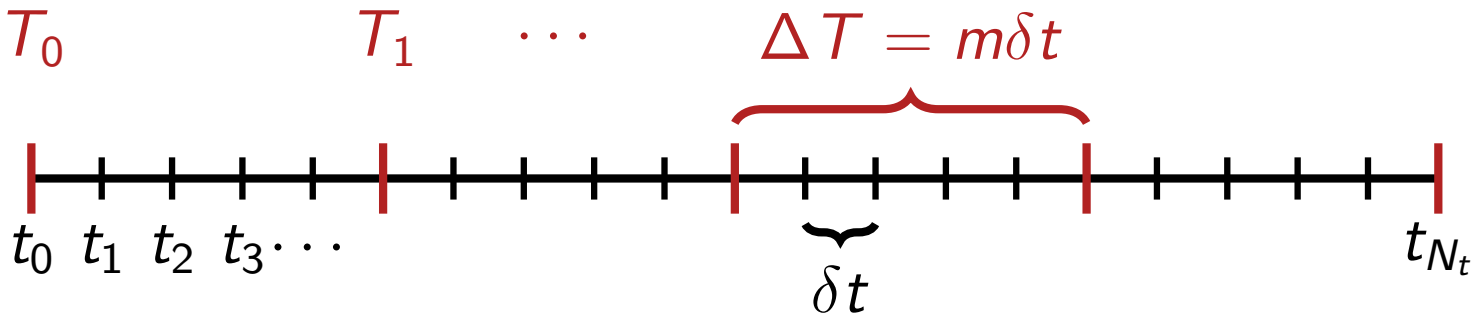


fine system

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

$$A = \begin{bmatrix} I & & & \\ -\Phi_{\delta t} & I & & \\ & \ddots & \ddots & \\ & & -\Phi_{\delta t} & I \end{bmatrix}$$

Schur Complement



fine system

$$Au = g$$

Schur complement system

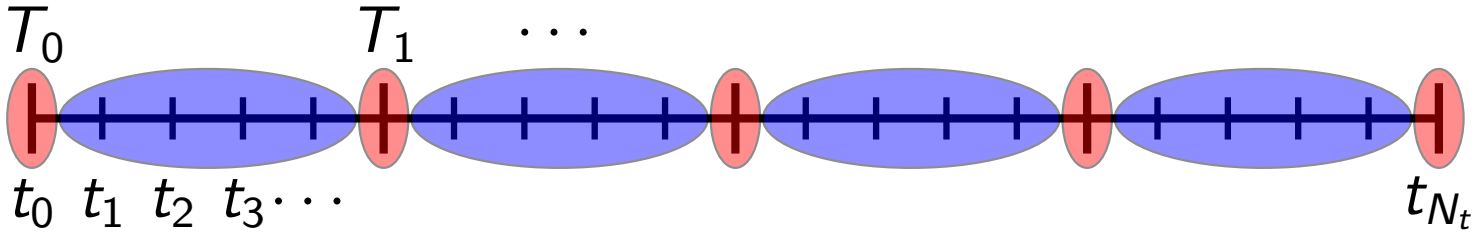
$$A_C u_\Delta = g_\Delta \equiv R_\Phi g$$

$$A = \begin{bmatrix} I & & & \\ -\Phi_{\delta t} & I & & \\ & \ddots & \ddots & \\ & & -\Phi_{\delta t} & I \end{bmatrix}$$

$$A_C = \begin{bmatrix} I & & & \\ -\Phi_{\delta t}^m & I & & \\ & \ddots & \ddots & \\ & & -\Phi_{\delta t}^m & I \end{bmatrix}$$

Exact Two-Level Method

- Partition the time grid into C-points and F-points



- reorder the fine-grid operator,

$$A = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix}$$

Exact Two-Level Method

- reordered fine-grid operator, $A = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix}$
- define operators

$$R_\Phi = \begin{bmatrix} -A_{cf}A_{ff}^{-1} & I_c \end{bmatrix}, \quad P_\Phi = \begin{bmatrix} -A_{ff}^{-1}A_{fc} \\ I_c \end{bmatrix}, \quad S = \begin{bmatrix} I_f \\ 0 \end{bmatrix}$$

"ideal" restriction — "ideal" interpolation

- Petrov-Galerkin coarse-grid operator: $A_C = R_\Phi A P_\Phi$ is the Schur complement of A_{ff} in A, $A_C = A_{cc} - A_{cf}A_{ff}^{-1}A_{fc}$
- cyclic reduction can be interpreted as an exact two-level method

$$(I - S(S^T A S)^{-1} S^T A)\,(I - P_\Phi A_C^{-1} R_\Phi A)$$

F-relaxation — coarse-grid correction

Exact Two-Level Method

- reordered fine-grid operator, $A = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix}$
- define operators

$$R_\Phi = \begin{bmatrix} -A_{cf}A_{ff}^{-1} & I_c \end{bmatrix}, \quad P_\Phi = \begin{bmatrix} -A_{ff}^{-1}A_{fc} \\ I_c \end{bmatrix}, \quad S = \begin{bmatrix} I_f \\ 0 \end{bmatrix}$$

"ideal" restriction — "ideal" interpolation

- Petrov-Galerkin coarse-grid operator: $A_C = R_\Phi A P_\Phi$ is the Schur complement of A_{ff} in A, $A_C = A_{cc} - A_{cf}A_{ff}^{-1}A_{fc}$
- cyclic reduction can be interpreted as an exact two-level method

$$(I - P_\Phi A_C^{-1} R_\Phi A)\,(I - S(S^T A S)^{-1} S^T A)$$

coarse-grid correction — F-relaxation

Cost of Cyclic Reduction

$$(I - P_\Phi A_C^{-1} R_\Phi A)\,(I - S(S^T A S)^{-1} S^T A)$$

F-relaxation

- Cheap, each F-point takes 1 time-stepping operation
- Parallelizable, blocks between C-points are independent

Coarse-grid correction

- Expensive, application of $\Phi_{\delta t}^m$ requires m time-stepping solves
- Sequential, with no natural parallelism
- Could recurse. but get larger powers of $\Phi_{\delta t}$ on coarser grids

Multigrid Reduction (MGR)

MGR modifies cyclic reduction to incorporate multigrid ideas

- replace A_C with the coarse-scale time discretization (analog of A on the coarse grid) A_Δ,

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

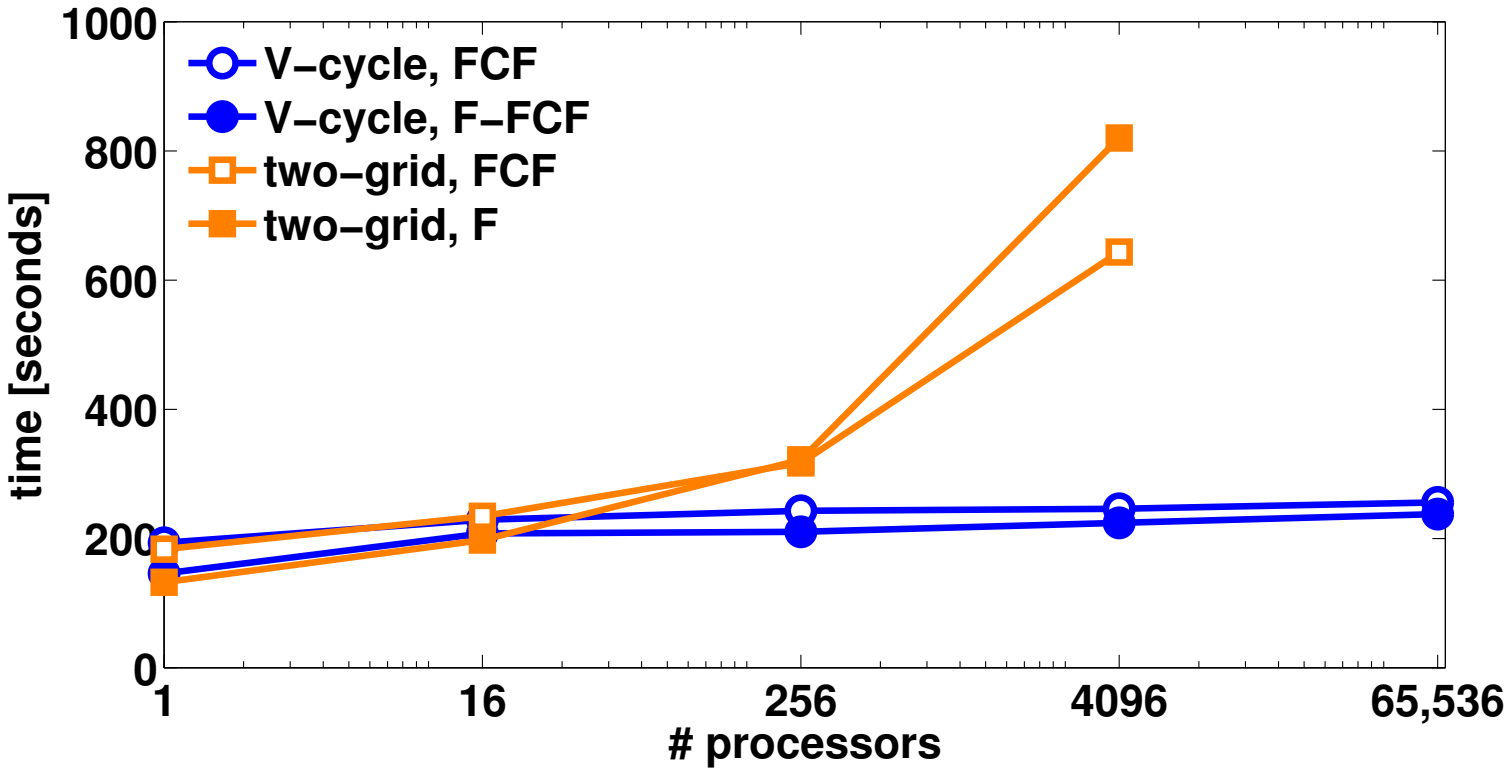
- use *FCF*-relaxation instead of *F*-relaxation

Coarse-grid correction is now cheap, and recursion is natural

- No longer an exact method, iteration needed

M. Ries, U. Trottenberg, and G. Winter, Linear Algebra Appl., **49** (1983)

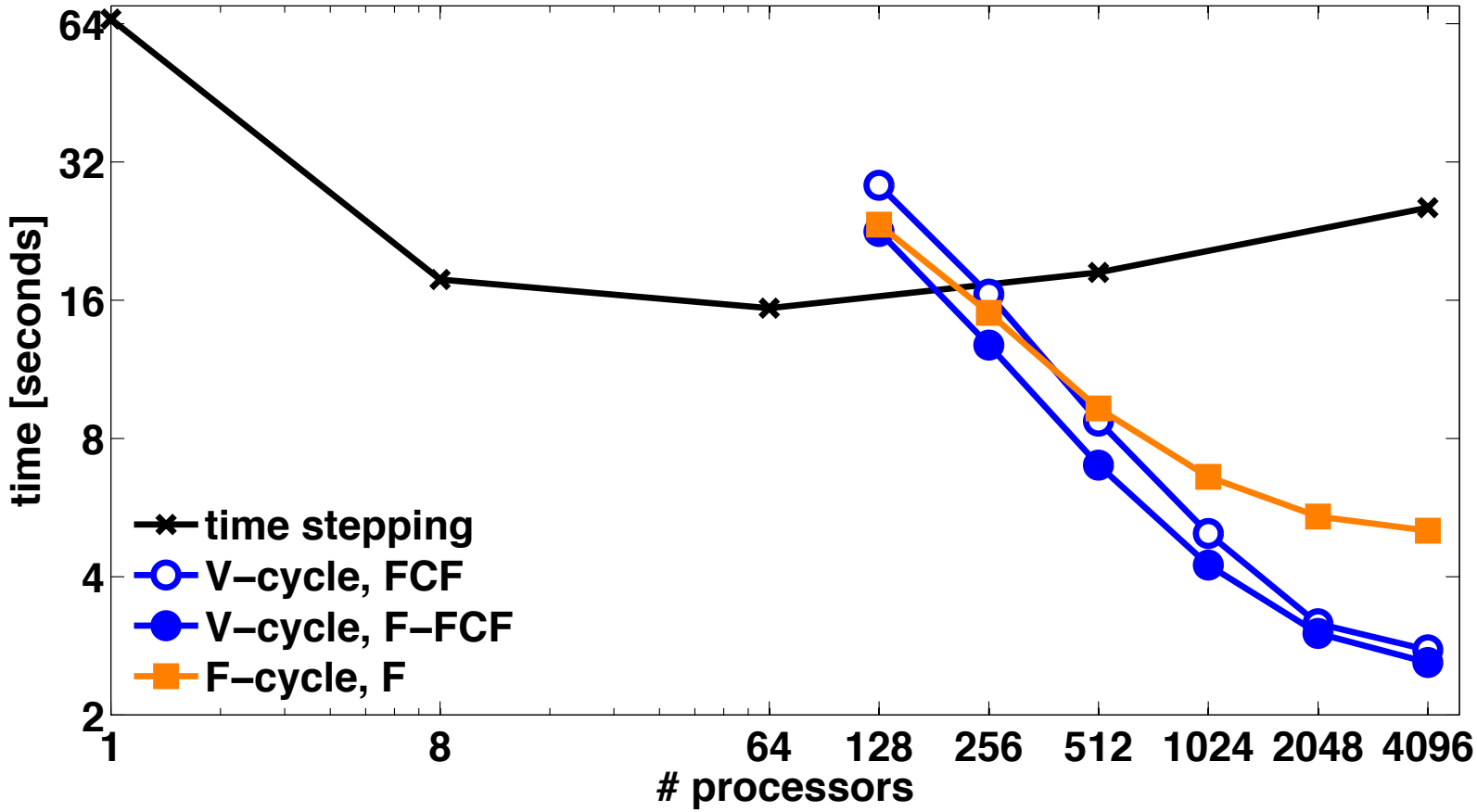
Weak Parallel Scaling - BG/Q



Heat equation in 2D space, finite differences in space, implicit Euler in time, about $(2^7)^2 \times 2^8 = 4 \times 2^{20}$ DoFs per core

Falgout, Friedhoff, Kolev, MacLachlan, Schroder, SISC 2014

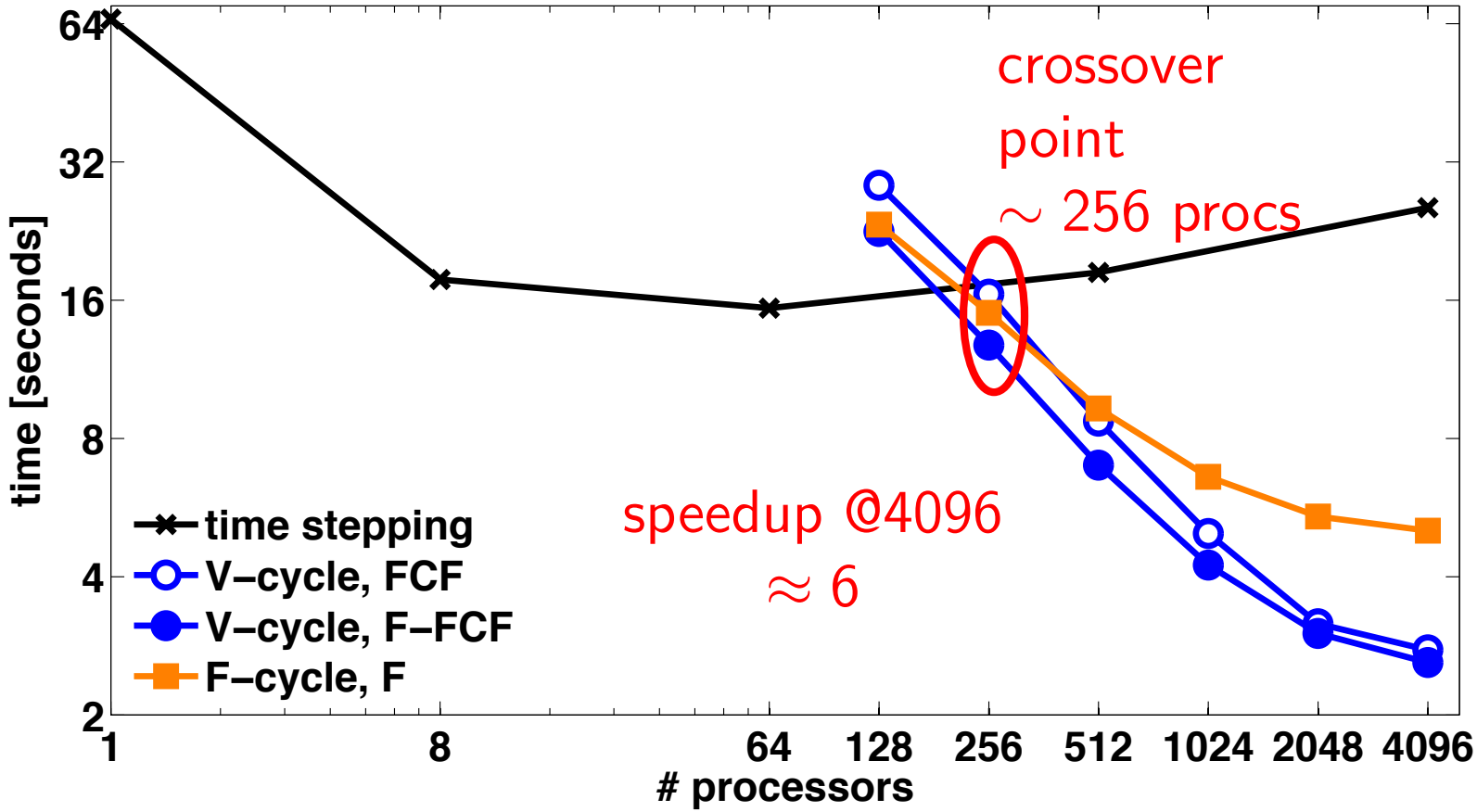
3D Strong Parallel Scaling - Cluster



Heat equation in 3D space, finite differences in space, implicit Euler in time, on a $33^3 \times 4097$ grid ($\approx 128 \times 2^{20}$ DOFs)

Falgout, Friedhoff, Kolev, MacLachlan, Schroder, SISC 2014

3D Strong Parallel Scaling - Cluster



Heat equation in 3D space, finite differences in space, implicit Euler in time, on a $33^3 \times 4097$ grid ($\approx 128 \times 2^{20}$ DOFs)

Falgout, Friedhoff, Kolev, MacLachlan, Schroder, SISC 2014

Slowdown and Speedup

MGRIT is an inefficient algorithm

- It offers optimal scaling: $\mathcal{O}(N_x N_t)$
- The constant is huge
  - MGRIT requires about 40$\times$ the computation as time-stepping for 3D heat equation

MGRIT has excellent parallel efficiency

- Near-perfect weak scaling
- Excellent strong scaling over wide range of processor counts

Overall speedup depends on number of cores available

- Need at least 40$\times$ more cores than can effectively use with spatial parallelism to see pay-off

Implicit 1D Advection

Use least-squares process to solve for coarse-grid operator

- Two-level, with coarsening by factor of m
- Use explicit-style CGO for implicit fine-grid problem

Scheme	$n_x \times n_t$	m					
		2	4	8	16	32	64
SDIRK1+U1	$2^{10} \times 2^{10}$	10	7	8	10	8	7
	$2^{12} \times 2^{12}$	10	7	8	11	9	9
SDIRK2+U2	$2^{10} \times 2^{10}$	10	8	7	8	8	7
	$2^{12} \times 2^{12}$	11	8	7	8	8	8
SDIRK3+U3	$2^{10} \times 2^{10}$	5	5	5	4	4	4
	$2^{12} \times 2^{12}$	5	5	5	4	4	4
SDIRK4+U4	$2^{10} \times 2^{10}$	6	6	5	5	5	5
	$2^{12} \times 2^{12}$	6	6	5	5	5	5

De Sterck, Falgout, Friedhoff, Krzysik, MacLachlan, NLAA, to appear

Explicit 1D Advection

Use least-squares process to solve for coarse-grid operator

- Inflow/Outflow BCs (not periodic)
- Coarsening by factor of 4 on each level

Scheme	$n_x \times n_t$	Number of levels, ℓ					
		2	3	4	5	6	7
ERK1+U1	$2^8 \times 2^{10}$	6	6	6	6	–	–
	$2^{10} \times 2^{12}$	6	6	6	6	6	–
	$2^{12} \times 2^{14}$	6	7	7	7	7	7
ERK3+U3	$2^8 \times 2^9$	6	6	6	–	–	–
	$2^{10} \times 2^{11}$	6	7	7	7	–	–
	$2^{12} \times 2^{13}$	6	7	7	7	7	–
ERK5+U5	$2^8 \times 2^9$	5	5	5	–	–	–
	$2^{10} \times 2^{11}$	5	5	5	5	–	–
	$2^{12} \times 2^{13}$	5	5	5	5	5	–

De Sterck, Falgout, Friedhoff, Krzysik, MacLachlan, NLAA, to appear

Software

Non-intrusive C-implementation as XBraid

- Uses existing time-stepping algorithm in all MGRIT computations

Python implementation in PyMGRIT

- Write your own time-stepper or use Firedrake
- Coupled with PETSc via petsc4py

`llnl.gov/casc/xbraid`
`pymgrit.github.io/pymgrit/`

Ongoing directions

Multitude of open questions

- Hyperbolic PDEs (with De Sterck, Friedhoff, others)
  - Extend advection results to harder problems
  - Effects of fine and coarse discretizations
- Theoretical analysis
  - Mode analysis explaining hyperbolic results
  - Methodology still under development
- Real applications
  - Falgout et al.: compressible flow
  - Schroder et al.: optimization, power systems
  - Friedhoff, Schöps: induction machines