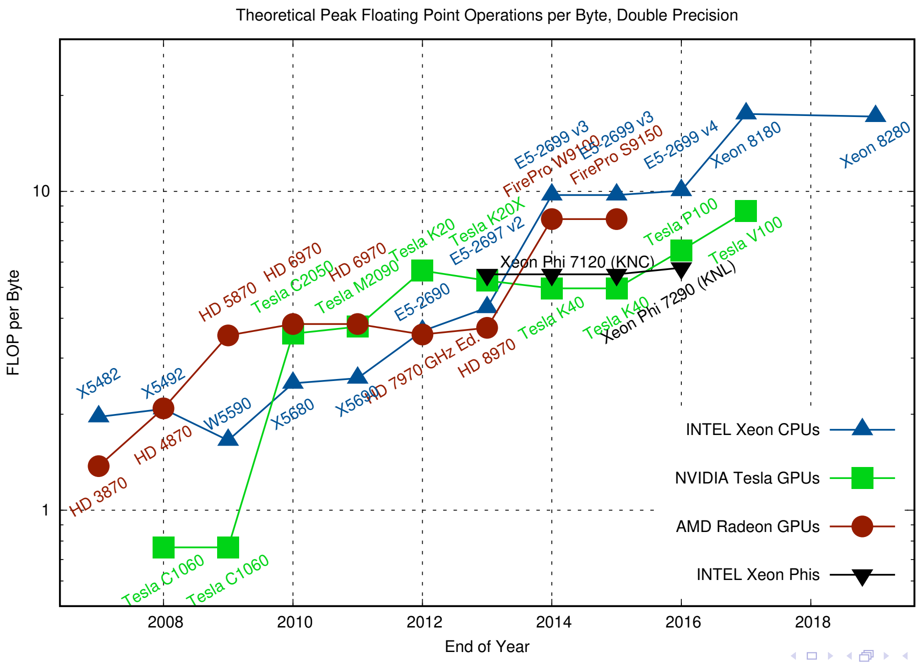
On Time Integration for Strong Scalability

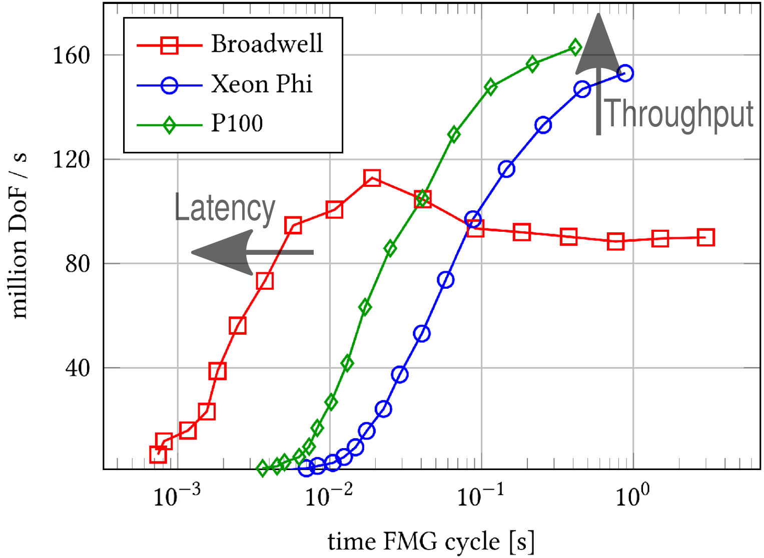
Jed Brown jed.brown@colorado.edu (CU Boulder and ANL)
Collaborators: Debojyoti Ghosh (LLNL), Matt Normile (CU),
Martin Schreiber (Exeter), Richard Mills (ANL)

PETSc User Meeting, 2019-06-06
This talk: https://jedbrown.org/files/20190606-StrongTime.pdf



Theoretical Peak Floating Point Operations per Byte, Double Precision
FLOP per Byte
End of Year
10
1
2008
2010
2012
2014
2016
2018
X5482
X5492
W5590
X5680
X5690
E5-2690
E5-2697 v2
E5-2699 v3
E5-2699 v3
E5-2699 v4
Xeon 8180
Xeon 8280
HD 3870
HD 4870
HD 5870
HD 6970
HD 6970
HD 7970 GHz Ed.
HD 8970
FirePro W9100
FirePro S9150
Tesla C1060
Tesla C1060
Tesla C2050
Tesla M2090
Tesla K20
Tesla K20X
Tesla K40
Tesla K40
Tesla P100
Tesla V100
Xeon Phi 7120 (KNC)
Xeon Phi 7290 (KNL)
INTEL Xeon CPUs
NVIDIA Tesla GPUs
AMD Radeon GPUs
INTEL Xeon Phis

Latency versus Throughput



Adapted from Kronbichler and Ljungkvist (2019)

Motivation

- Hardware trends
  - Memory bandwidth a precious commodity (8+ flops/byte)
  - Vectorization necessary for floating point performance
  - Conflicting demands of cache reuse and vectorization
  - Can deliver bandwidth, but latency is hard
- Assembled sparse linear algebra is doomed!
  - Limited by memory bandwidth (1 flop/6 bytes)
  - No vectorization without blocking, return of ELLPACK
- Spatial-domain vectorization is *intrusive*
  - Must be unassembled to avoid bandwidth bottleneck
  - Whether it is “hard” depends on discretization
  - Geometry, boundary conditions, and adaptivity

Sparse linear algebra is dead (long live sparse . . .)

- Arithmetic intensity $< 1/4$
- Idea: multiple right hand sides

$$\frac{(2k \text{ flops})(\text{bandwidth})}{\texttt{sizeof(Scalar)} + \texttt{sizeof(Int)}}, \quad k \ll \text{avg. nz/row}$$

- Problem: popular algorithms have nested data dependencies
  - Time step
    - Nonlinear solve
      - Krylov solve
        - Preconditioner/sparse matrix
- Cannot parallelize/vectorize these nested loops
- Can we create new algorithms to reorder/fuse loops?
  - Reduce latency-sensitivity for communication
  - Reduce memory bandwidth (reuse matrix while in cache)

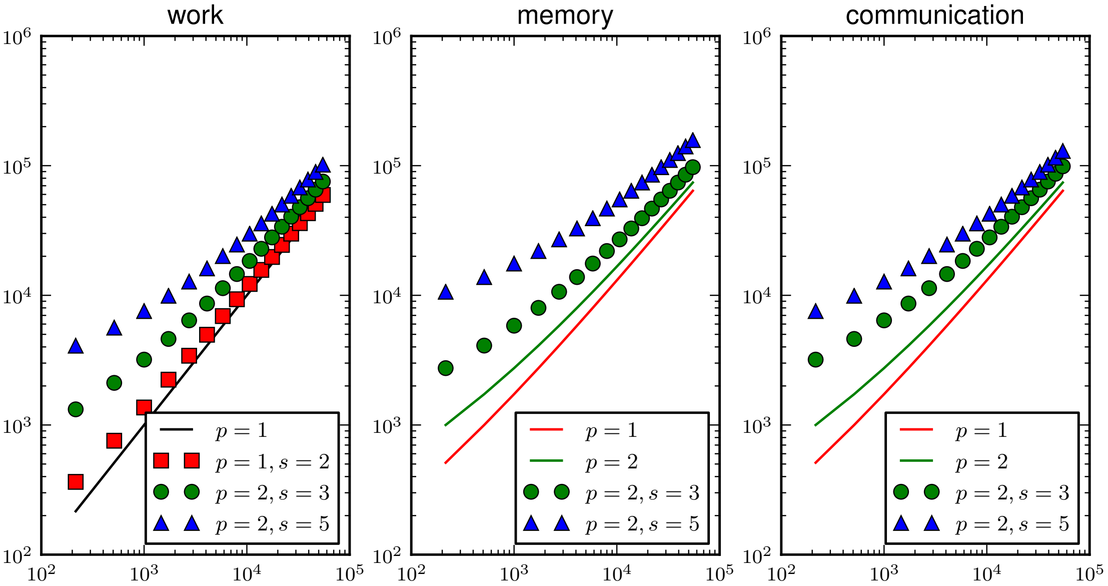
Sparse linear algebra is dead (long live sparse . . .)

- Arithmetic intensity $< 1/4$
- Idea: multiple right hand sides

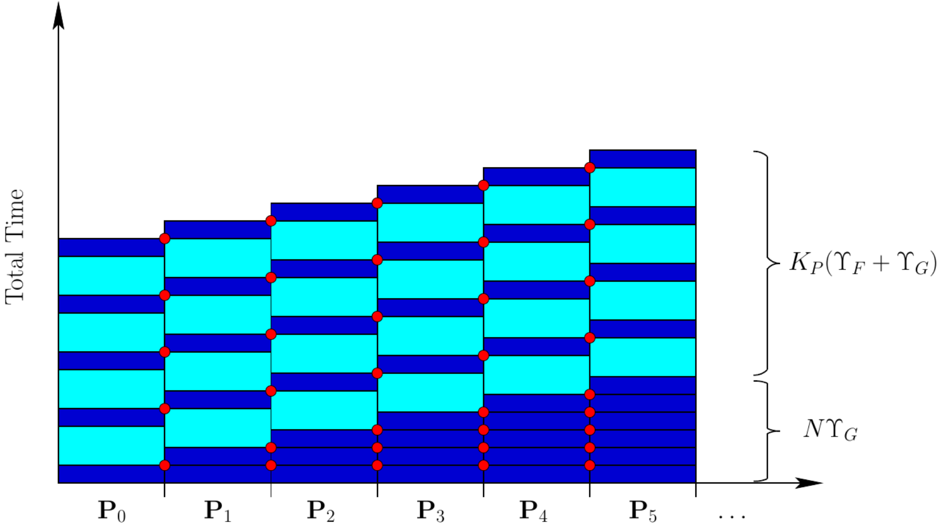
$$\frac{(2k \text{ flops})(\text{bandwidth})}{\texttt{sizeof(Scalar)} + \texttt{sizeof(Int)}}, \quad k \ll \text{avg. nz/row}$$

- Problem: popular algorithms have nested data dependencies
  - Time step
    - Nonlinear solve
      - Krylov solve
        - Preconditioner/sparse matrix
- Cannot parallelize/vectorize these nested loops
- Can we create new algorithms to reorder/fuse loops?
  - Reduce latency-sensitivity for communication
  - Reduce memory bandwidth (reuse matrix while in cache)

Attempt: *s*-step methods in 3D

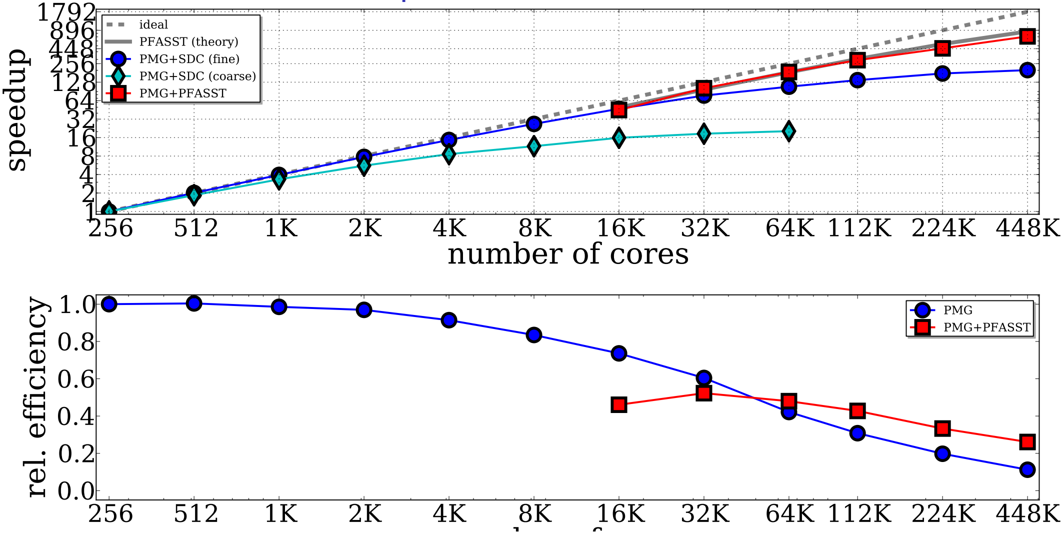


Attempt: distribute in time (multilevel SDC/Parareal)



- PFASST algorithm (Emmett and Minion, 2012)
- Zero-latency messages (cf. performance model of *s*-step)

Problems with SDC and time-parallel



c/o Matthew Emmett, parallel compared to sequential SDC

- Iteration count not uniform in s; efficiency starts low

Runge-Kutta methods

$$\dot{u} = F(u)$$

$$\underbrace{\begin{pmatrix} y_1 \\ \vdots \\ y_s \end{pmatrix}}_{Y} = u^n + h \underbrace{\begin{bmatrix} a_{11} & \cdots & a_{1s} \\ \vdots & \ddots & \vdots \\ a_{s1} & \cdots & a_{ss} \end{bmatrix}}_{A} F \begin{pmatrix} y_1 \\ \vdots \\ y_s \end{pmatrix}$$

$$u^{n+1} = u^n + hb^T F(Y)$$

- General framework for one-step methods
- Diagonally implicit: A lower triangular, stage order 1 (or 2 with explicit first stage)
- Singly diagonally implicit: all A_{ii} equal, reuse solver setup, stage order 1
- If A is a general full matrix, all stages are coupled, "implicit RK"

Implicit Runge-Kutta

$$
\begin{array}{c|ccc}
\frac{1}{2} - \frac{\sqrt{15}}{10} & \frac{5}{36} & \frac{2}{9} - \frac{\sqrt{15}}{15} & \frac{5}{36} - \frac{\sqrt{15}}{30} \\
\frac{1}{2} & \frac{5}{36} + \frac{\sqrt{15}}{24} & \frac{2}{9} & \frac{5}{36} - \frac{\sqrt{15}}{24} \\
\frac{1}{2} - \frac{\sqrt{15}}{10} & \frac{5}{36} + \frac{\sqrt{15}}{30} & \frac{2}{9} + \frac{\sqrt{15}}{15} & \frac{5}{36} \\
\hline
 & \frac{5}{18} & \frac{4}{9} & \frac{5}{18}
\end{array}
$$

- Excellent accuracy and stability properties
- Gauss methods with s stages
  - order $2s$, (s,s) Padé approximation to the exponential
  - A-stable, symplectic
- Radau (IIA) methods with s stages
  - order $2s-1$, A-stable, L-stable
- Lobatto (IIIC) methods with s stages
  - order $2s-2$, A-stable, L-stable, self-adjoint
- Stage order s or $s+1$

Method of Butcher (1976) and Bickart (1977)

- Newton linearize Runge-Kutta system at u^*

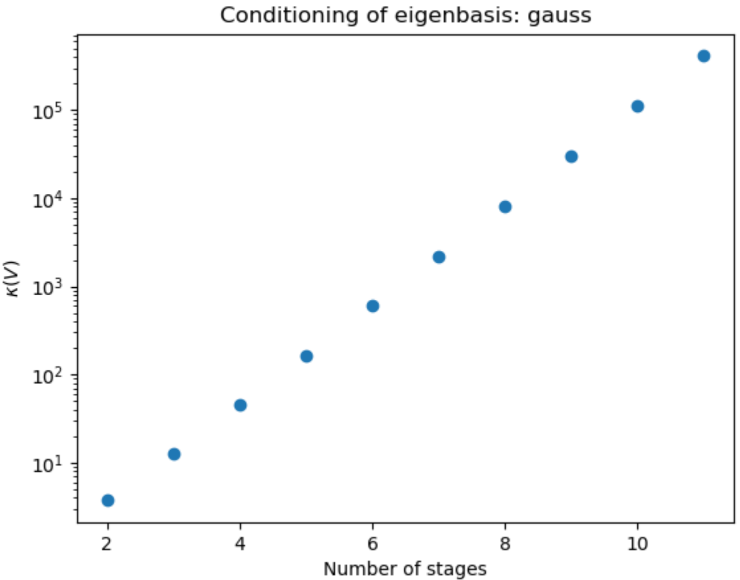
$$Y = u^n + hAF(Y) \qquad \left[I_s \otimes I_n + hA \otimes J(u^*)\right] \delta Y = RHS$$

- Solve linear system with tensor product operator

$$\hat{G} = S \otimes I_n + I_s \otimes J$$

  where $S = (hA)^{-1}$ is $s \times s$ dense, $J = -\partial F(u)/\partial u$ sparse
- SDC (2000) is Gauss-Seidel with low-order corrector
- Butcher/Bickart method: diagonalize $S = V \Lambda V^{-1}$
  - $\Lambda \otimes I_n + I_s \otimes J$
  - s decoupled solves
  - Complex eigenvalues (overhead for real problem)

Eigenbasis ill conditioning $A = V\Lambda V^{-1}$



Skip the diagonalization

$$\underbrace{\begin{bmatrix} s_{11}+J & s_{12}+J \\ s_{21}+J & s_{22}+J \end{bmatrix}}_{S\otimes I_n + I_s \otimes J} \qquad \underbrace{\begin{bmatrix} S+j_{11}I & j_{12}I & \\ j_{21}I & S+j_{22}I & j_{23}I \\ & j_{32}I & S+j_{33}I \end{bmatrix}}_{I_n\otimes S + J \otimes I_s}$$

- Accessing memory for J dominates cost
- Irregular vector access in application of J limits vectorization
- Permute Kronecker product to reuse J and make fine-grained structure regular
- Stages coupled via register transpose at spatial-point granularity
- Same convergence properties as Butcher/Bickart

PETSc MatKAIJ: “sparse” Kronecker product matrices

$$G = I_n \otimes S + J \otimes T$$

- J is parallel and sparse, S and T are small and dense
- More general than multiple RHS (multivectors)
- Compare $J \otimes I_s$ to multiple right hand sides in row-major
- Runge-Kutta systems have $T = I_s$ (permuted from Butcher method)
- Stream J through cache once, same efficiency as multiple RHS
- Unintrusive compared to spatial-domain vectorization or s-step

Convergence with point-block Jacobi preconditioning

- 3D centered-difference diffusion problem

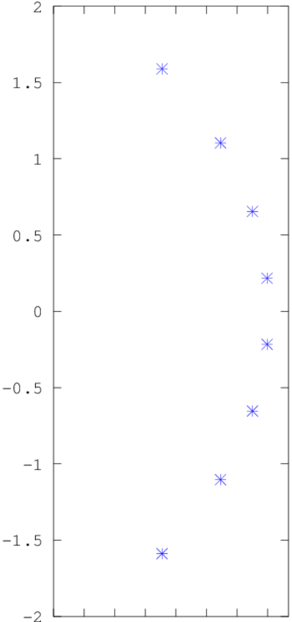
Method	order	nsteps	Krylov its.	(Average)
Gauss 1	2	16	130	(8.1)
Gauss 2	4	8	122	(15.2)
Gauss 4	8	4	100	(25)
Gauss 8	16	2	78	(39)

We really want multigrid

- Prolongation: $P \otimes I_s$
- Coarse operator: $I_n \otimes S + (RJP) \otimes I_s$
- Larger time steps
- GMRES(2)/point-block Jacobi smoothing
- FGMRES outer

Method	order	nsteps	Krylov its.	(Average)
Gauss 1	2	16	82	(5.1)
Gauss 2	4	8	64	(8)
Gauss 4	8	4	44	(11)
Gauss 8	16	2	42	(21)

Toward a better AMG for IRK/tensor-product systems



- Start with $\hat{R} = R \otimes I_s$, $\hat{P} = P \otimes I_s$

$$G_{\text{coarse}} = \hat{R}(I_n \otimes S + J \otimes I_s)\hat{P}$$

- Imaginary component slows convergence
- Can we use a Kronecker product interpolation?
- Rotation on coarse grids (connections to shifted Laplacian)

Why implicit is silly for waves

- Implicit methods require an implicit solve in each stage.
- Time step size proportional to CFL for accuracy reasons.
- Methods higher than first order are not unconditionally strong stability preserving (SSP; Spijker 1983).
  - Empirically, $c_{\text{eff}} \leq 2$, Ketcheson, Macdonald, Gottlieb (2008) and others
  - Downwind methods offer to bypass, but so far not practical
- Time step size chosen for stability
  - Increase order if more accuracy needed
  - Large errors from spatial discretization, modest accuracy
- My goal: need less memory motion *per stage*
  - Better accuracy, symplecticity nice bonus only
  - Cannot sell method without efficiency

Implicit Runge-Kutta for advection

Table: Total number of iterations (communications or accesses of J) to solve linear advection to $t = 1$ on a 1024-point grid using point-block Jacobi preconditioning of implicit Runge-Kutta matrix. The relative algebraic solver tolerance is 10^{-8}.

Method	order	nsteps	Krylov its.	(Average)
Gauss 1	2	1024	3627	(3.5)
Gauss 2	4	512	2560	(5)
Gauss 4	8	256	1735	(6.8)
Gauss 8	16	128	1442	(11.2)

- Naive centered-difference discretization
- Leapfrog requires 1024 iterations at CFL=1
- This is A-stable (can handle dissipation)

Diagonalization revisited

$$(I \otimes I - hA \otimes L)Y = (\mathbf{1} \otimes I)u_n \tag{1}$$
$$u_{n+1} = u_n + h(b^T \otimes L)Y \tag{2}$$

- eigendecomposition $A = V\Lambda V^{-1}$

$$(V \otimes I)(I \otimes I - h\Lambda \otimes L)(V^{-1} \otimes I)Y = (\mathbf{1} \otimes I)u_n.$$

- Find diagonal W such that $W^{-1}\mathbf{1} = V^{-1}\mathbf{1}$
- Commute diagonal matrices

$$(I \otimes I - h\Lambda \otimes L)\underbrace{(WV^{-1} \otimes I)Y}_{Z} = (\mathbf{1} \otimes I)u_n.$$

- Using $\tilde{b}^T = b^T VW^{-1}$, we have the completion formula

$$u_{n+1} = u_n + h(\tilde{b}^T \otimes L)Z.$$

- $\Lambda, \tilde{b}$ is new diagonal Butcher table

Exploiting realness

- Eigenvalues come in conjugate pairs

$$A = V\Lambda V^{-1}$$

- For each conjugate pair, create unitary transformation

$$T = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ i & -i \end{bmatrix}$$

- Real 2×2 block diagonal D; real $\tilde{V}$ (with appropriate phase)

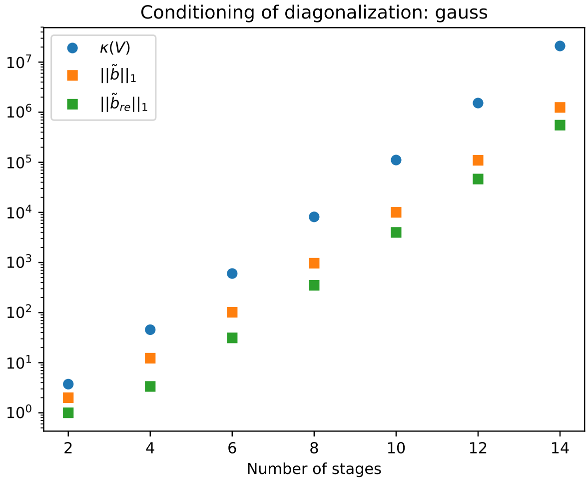
$$A = (VT^*)(T\Lambda T^*)(TV^{-1}) = \tilde{V}D\tilde{V}^{-1}$$

- Yields new block-diagonal Butcher table $D, \tilde{b}$.
- Halve number of stages using identity

$$\overline{(\alpha + J)^{-1} u} = (\overline{\alpha} + J)^{-1} u$$

Solve one complex problem per conjugate pair, then take twice the real part.

Conditioning



- Diagonalization in extended precision helps somewhat, as does real formulation
- Neither makes arbitrarily large number of stages viable

REXI: Rational approximation of exponential

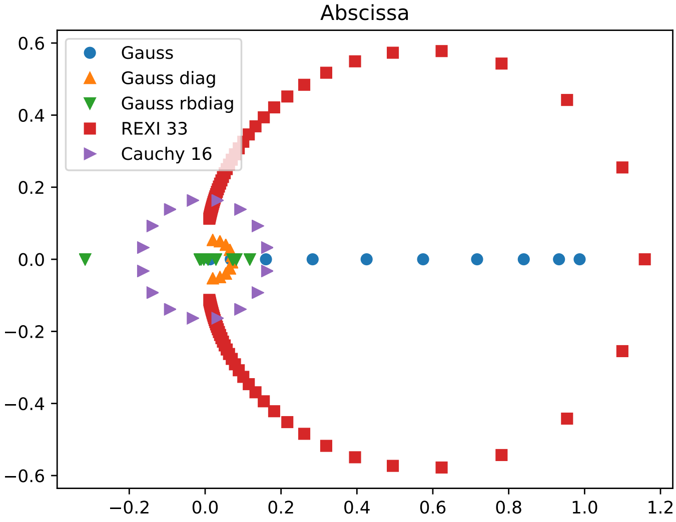
$$u(t) = e^{Lt}u(0)$$

- Haut, Babb, Martinsson, Wingate; Schreiber and Loft

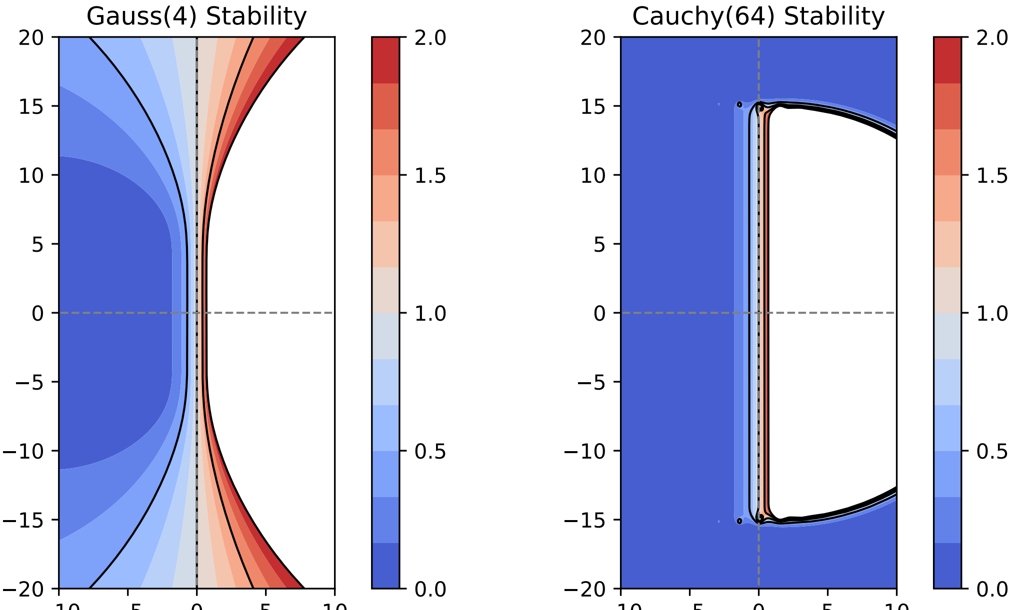
$$(\alpha \otimes I + hI \otimes L)Y = (\mathbb{1} \otimes I)u_n$$
$$u_{n+1} = (\beta^T \otimes I)Y.$$

- α is complex-valued diagonal, β is complex
- Constructs rational approximations of Gaussian basis functions, target (real part of) e^{it}
- REXI is a Runge-Kutta method: can convert via “modified Shu-Osher form”
  - Developed for SSP (strong stability preserving) methods
  - Ferracina, Spijker (2005), Higueras (2005)
  - Yields diagonal Butcher table $A = -\alpha^{-1}, b = -\alpha^{-2}\beta$

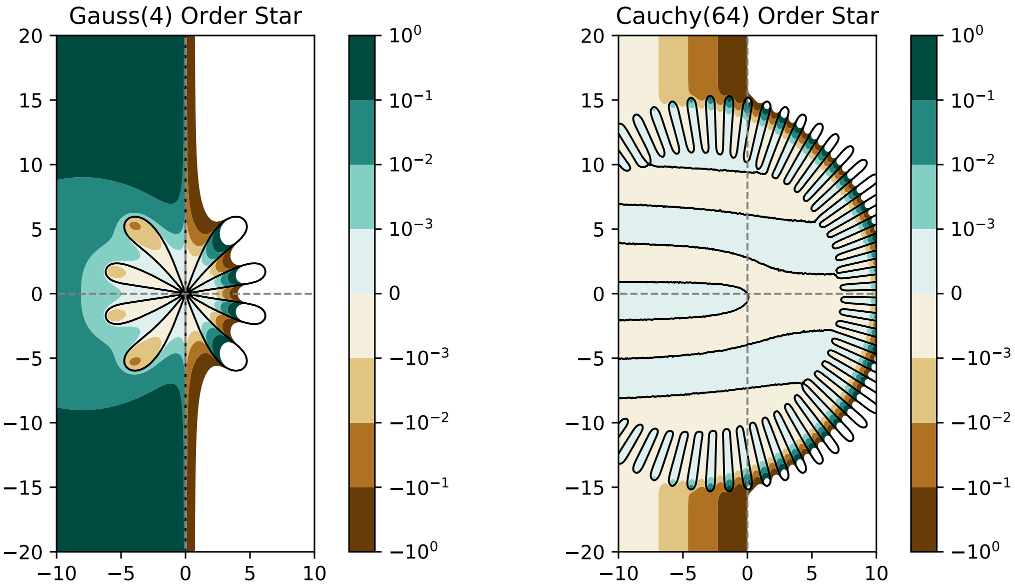
Abscissa for RK and REXI methods



Stability regions



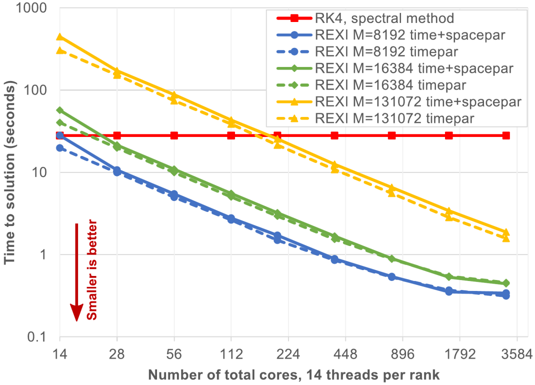
Order stars


Gauss(4) Order Star
Cauchy(64) Order Star
20
15
10
5
0
−5
−10
−15
−20
−10
−5
0
5
10
10^0
10^{-1}
10^{-2}
10^{-3}
0
-10^{-3}
-10^{-2}
-10^{-1}
-10^0

Computational Performance (SWE on the plane)

Spectral solver with **RK4 time stepping** method
vs. **REXI with spectral solver**

Resolution $= 128^2$



More than one order of magnitude **faster** with **similar accuracy**

Proof of concept that REXI works with spectral methods

Computed on Linux Cluster, LRZ / Technical University of Munich

M. Schreiber, P. Peixoto, TS. Haut, BA. Wingate, Beyond spatial scalability limitations with a

Outlook on Kronecker product solvers

$$I \otimes S + J \otimes T$$

- (Block) diagonal S is usually sufficient
- Best opportunity for “time parallel” (for linear problems)
  - Is it possible to beat explicit wave propagation *with high efficiency*?
- Same structure for stochastic Galerkin and other UQ methods
- IRK *unintrusively* offers bandwidth reuse and vectorization
- Need polynomial smoothers for IRK spectra
- Change number of stages on spatially-coarse grids (p-MG, or even increase)?
- Experiment with SOR-type smoothers
  - Prefer point-block Jacobi in smoothers for spatial parallelism
- Possible IRK correction for IMEX (non-smooth explicit function)
- PETSc implementation (works in parallel, hardening in progress)
- Thanks to DOE ASCR