Practical and Efficient Time Integration and Kronecker Product Solvers

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

SIAM Central, 2017-09-30
This talk: https://jedbrown.org/files/20170930-FastKronecker.pdf
2017 HPGMG performance spectra
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

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

- 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

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

- 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

- Limited choice of preconditioners (none optimal, surface/volume)
- Amortizing message latency is most important for strong-scaling
- s-step methods have high overhead for small subdomains
Attempt: distribute in time (multilevel SDC/Parareal)

- PFASST algorithm (Emmett and Minion, 2012)
- Zero-latency messages (cf. performance model of s-step)
- Spectral Deferred Correction: iterative, converges to IRK (Gauss, Radau, ...)
- Stiff problems use implicit basic integrator (synchronizing on spatial communicator)
Problems with SDC and time-parallel

- Iteration count not uniform in $s$; efficiency starts low
- Low arithmetic intensity; tight error tolerance (cf. Crank-Nicolson)
- Parabolic space-time also works, but comparison flawed

c/o Matthew Emmett, parallel compared to sequential SDC
Runge-Kutta methods

\[ \dot{u} = F(u) \]

\[ \begin{pmatrix} y_1 \\ \vdots \\ y_s \end{pmatrix} = u^n + h \begin{pmatrix} a_{11} & \cdots & a_{1s} \\ \vdots & \ddots & \vdots \\ a_{s1} & \cdots & a_{ss} \end{pmatrix} F \begin{pmatrix} y_1 \\ \vdots \\ y_s \end{pmatrix} \]

\[ u^{n+1} = u^n + hb^T F(Y) \]

- 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

\[
\frac{1}{2} - \frac{\sqrt{15}}{10} \quad \frac{5}{36} \quad \frac{2}{9} - \frac{\sqrt{15}}{15} \quad \frac{5}{36} - \frac{\sqrt{15}}{30} \\
\frac{1}{2} \quad \frac{5}{36} + \frac{\sqrt{15}}{24} \quad \frac{2}{9} \quad \frac{5}{36} + \frac{\sqrt{15}}{24} \\
\frac{1}{2} - \frac{\sqrt{15}}{10} \quad \frac{5}{36} \quad \frac{2}{9} + \frac{\sqrt{15}}{15} \quad \frac{5}{36} \\
\frac{5}{18} \quad \frac{4}{9} \quad \frac{5}{18}
\]

- 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 $u^*$

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

- Solve linear system with tensor product operator

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

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

  - $\Lambda \otimes I_n + l_s \otimes J$
  - $s$ decoupled solves
  - Complex eigenvalues (overhead for real problem)
Ill conditioning

\[ A = V \Lambda V^{-1} \]
Skip the diagonalization

\[
\begin{bmatrix}
  s_{11} + J & s_{12} + J \\
  s_{21} + J & s_{22} + J
\end{bmatrix}
\]

\[
\begin{bmatrix}
  S + j_{11} I & j_{12} I \\
  j_{21} I & S + j_{22} I \\
  j_{31} I & S + j_{33} I
\end{bmatrix}
\]

- 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

<table>
<thead>
<tr>
<th>Method</th>
<th>order</th>
<th>nsteps</th>
<th>Krylov its.</th>
<th>(Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss 1</td>
<td>2</td>
<td>16</td>
<td>130</td>
<td>(8.1)</td>
</tr>
<tr>
<td>Gauss 2</td>
<td>4</td>
<td>8</td>
<td>122</td>
<td>(15.2)</td>
</tr>
<tr>
<td>Gauss 4</td>
<td>8</td>
<td>4</td>
<td>100</td>
<td>(25)</td>
</tr>
<tr>
<td>Gauss 8</td>
<td>16</td>
<td>2</td>
<td>78</td>
<td>(39)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Method</th>
<th>order</th>
<th>nsteps</th>
<th>Krylov its.</th>
<th>(Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss 1</td>
<td>2</td>
<td>16</td>
<td>82</td>
<td>(5.1)</td>
</tr>
<tr>
<td>Gauss 2</td>
<td>4</td>
<td>8</td>
<td>64</td>
<td>(8)</td>
</tr>
<tr>
<td>Gauss 4</td>
<td>8</td>
<td>4</td>
<td>44</td>
<td>(11)</td>
</tr>
<tr>
<td>Gauss 8</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>(21)</td>
</tr>
</tbody>
</table>
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 $10^{-8}$.

<table>
<thead>
<tr>
<th>Method</th>
<th>order</th>
<th>nsteps</th>
<th>Krylov its.</th>
<th>(Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss 1</td>
<td>2</td>
<td>1024</td>
<td>3627</td>
<td>(3.5)</td>
</tr>
<tr>
<td>Gauss 2</td>
<td>4</td>
<td>512</td>
<td>2560</td>
<td>(5)</td>
</tr>
<tr>
<td>Gauss 4</td>
<td>8</td>
<td>256</td>
<td>1735</td>
<td>(6.8)</td>
</tr>
<tr>
<td>Gauss 8</td>
<td>16</td>
<td>128</td>
<td>1442 (11.2)</td>
<td></td>
</tr>
</tbody>
</table>

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

\[(I \otimes I - hA \otimes L) Y = (1 \otimes I) u_n\]  
\[u_{n+1} = u_n + h(b^T \otimes L) Y\]  

- eigendecomposition \( A = V \Lambda V^{-1} \)
  
  \[(V \otimes I)(I \otimes I - h\Lambda \otimes L)(V^{-1} \otimes I) Y = (1 \otimes I) u_n.\]

- Find diagonal \( W \) such that \( W^{-1} 1 = V^{-1} 1 \)

- Commute diagonal matrices
  
  \[(I \otimes I - h\Lambda \otimes L)(WV^{-1} \otimes I) Y = (1 \otimes I) u_n.\]

  \[Z\]

- 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

- Compute coefficients offline using extended precision to handle ill-conditioning of \( V \)
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 \times 2 block diagonal \( D \); real \( \tilde{V} \) (with appropriate phase)
  \[ A = (VT^*)(\Lambda T^*)(TV^{-1}) = \tilde{V}D\tilde{V}^{-1} \]

- Yields new block-diagonal Butcher table \( D, \tilde{b} \).
- Halve number of stages using identity
  \[ (\alpha + J)^{-1}u = (\bar{\alpha} + J)^{-1}u \]

Solve one complex problem per conjugate pair, then take twice the real part.
Conditioning of diagonalization: gauss

$\|b\|_1$

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

\[
(\alpha \otimes I + hl \otimes L)Y = (\beta \otimes I)u_n
\]

\[ u_{n+1} = (\beta^T \otimes I)Y. \]

- \( \alpha \) is complex-valued diagonal, \( \beta \) 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

Stability region, Gauss, #poles=4

Order star, Gauss, #poles=4

Stability region, Cauchy (16,6)

Order star, Cauchy (16,6)
Outlook on Kronecker product solvers

\[ I \otimes S + J \otimes I \]

- (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