Fast solvers for implicit Runge-Kutta systems

Jed Brown jedbrown@mcs.anl.gov
Debojyoti Ghosh ghosh@mcs.anl.gov

Mathematics and Computer Science Division
Argonne National Laboratory

JointLab, UIUC, 2013-11-26

Outline

The memory bandwidth problem

Implicit Runge-Kutta

Tensor product algebra

Hardware Arithmetic Intensity

Operation	Arithmetic Intensity (flops/B)
Sparse matrix-vector product	1/6
Dense matrix-vector product	1/4
Unassembled matrix-vector product	≈ 8
High-order residual evaluation	> 5

Processor	Bandwidth (GB/s)	Peak (GF/s)	Balanced AI (F/B
E5-2680 8-core	38	173	4.
Magny Cours 16-core	49	281	5.
Blue Gene/Q node	43	205	4.
Tesla M2090	120	665	5.
Kepler K20Xm	160	1310	8.
Xeon Phi SE10P	161	1060	6.

Optimizing Sparse Mat-Vec

- Order unknowns so vector reuses cache (Cuthill-McKee)
  - Optimal: $\frac{(2\text{ flops})(\text{bandwidth})}{\texttt{sizeof(Scalar)}+\texttt{sizeof(Int)}}$
  - Usually improves strength of ILU and SOR
- Coalesce indices for adjacent rows (Inodes)
  - Optimal: $\frac{(2\text{ flops})(\text{bandwidth})}{\texttt{sizeof(Scalar)}+\texttt{sizeof(Int)}/i}$
  - Can do block SOR (much stronger than scalar SOR)
  - Default in PETSc, turn off with `-mat_no_inode`
  - Requires ordering unknowns so that fields are interlaced, this is (much) better for memory use anyway
- Use explicit blocking, hold one index per block (BAIJ format)
  - Optimal: $\frac{(2\text{ flops})(\text{bandwidth})}{\texttt{sizeof(Scalar)}+\texttt{sizeof(Int)}/b^2}$
  - Block SOR and factorization
  - Symbolic factorization works with blocks (much cheaper)
  - Very regular memory access, unrolled dense kernels
  - Faster insertion: `MatSetValuesBlocked()`

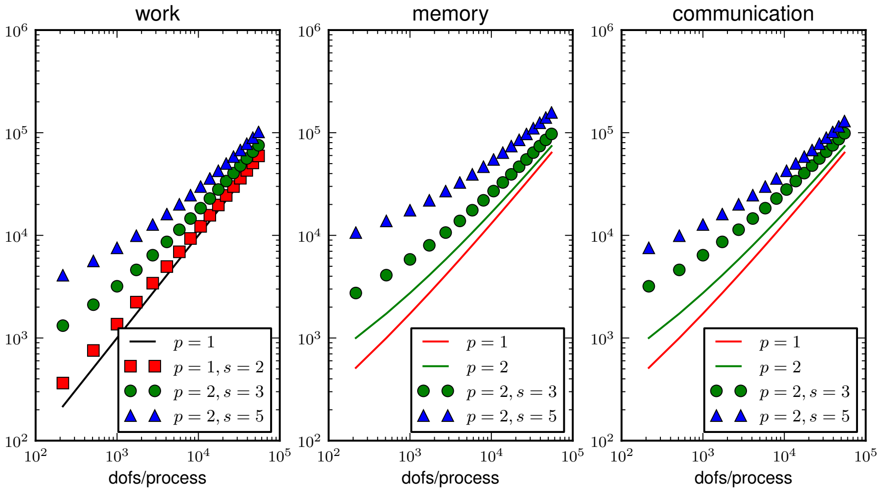
This is a dead end

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

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

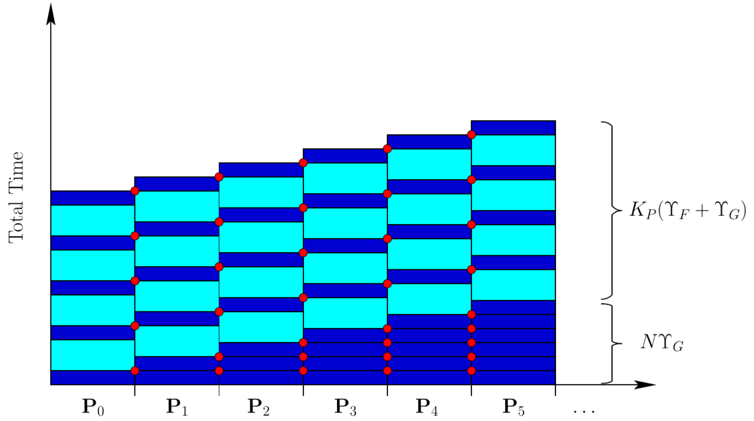
- Problem: popular algorithms have nested data dependencies
  - Time step
    - Nonlinear solve
      - Krylov solve
        - Preconditioner/sparse matrix
- Cannot parallelize/vectorize these nested loops

Attempt: *s*-step methods in 3D



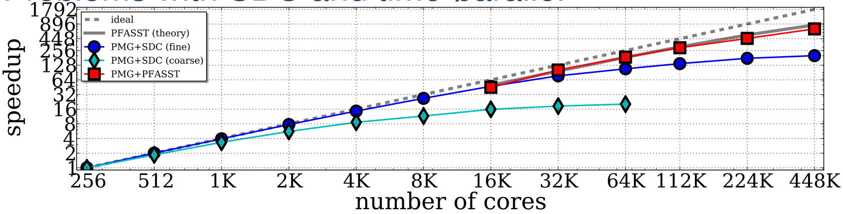
- Amortizing message latency is most important for strong-scaling
- *s*-step methods have high overhead for small subdomains
- Limited choice of preconditioners (none optimal)

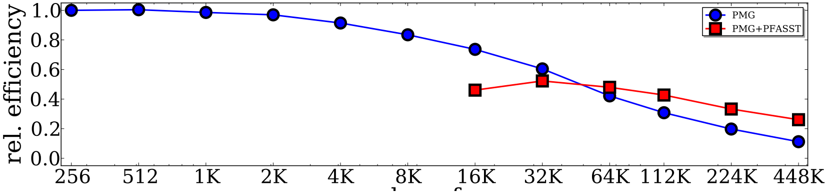
Attempt: space-time methods (multilevel SDC/Parareal)



- PFASST algorithm (Emmett and Minion, 2013)
- 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

Problems with SDC and time-parallel





c/o Matthew Emmett, parallel compared to sequential SDC

- Number of iterations is not uniform, efficiency starts low
- Arithmetic intensity unchanged
- Parabolic space-time (Greenwald and Brandt/Horton and Vandewalle)

Outline

The memory bandwidth problem

Implicit Runge-Kutta

Tensor product algebra

Runge-Kutta methods

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

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

$$u^{n+1} = b^T Y$$

- General framework for one-step methods
- Diagonally implicit: A lower triangular, stage order ≤ 2
- 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

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

- Implicit Runge-Kutta methods have 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

$$Y = u^n + hAF(Y)$$

- Solve linear system with tensor product operator

$$S \otimes I_n + I_s \otimes J$$

  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 $S = X \Lambda X^{-1}$
  - $\Lambda \otimes I_n + I_s \otimes J$
- s decoupled solves
- Problem: X is exponentially ill-conditioned wrt. s

Outline

The memory bandwidth problem

Implicit Runge-Kutta

Tensor product algebra

MatTAIJ: “sparse” tensor product matrices

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

- More general than multiple RHS (multivectors)
- Compare 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

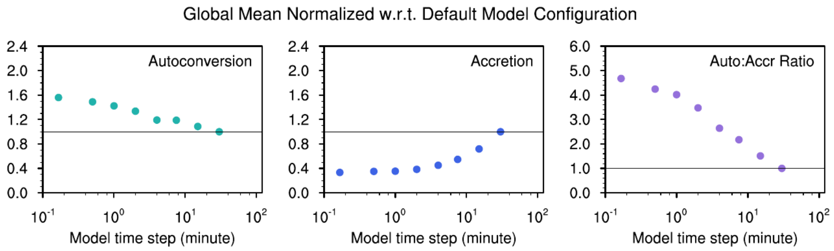
128 nodes, 16 procs/node, *small* diffusion problem

Method	order	nsteps	time
Gauss 4	8	10	3.4345e-01
Gauss 2	4	20	7.6320e-01
Gauss 1	2	40	1.1052e+00

Calibration and accuracy

- Splitting errors plague multi-physics simulation
- Verlet (leapfrog) integration is popular: symplectic and **cheap**
  - Stability problems: damping and even/odd decoupling
- Models calibrated to compensate
  - Force parametrizations in molecular dynamics
  - Atmospheric column physics

Impact of time step on autoconversion vs accretion partitioning (from Hui)



c/o Peter Caldwell (LLNL)

- Models calibrated for “efficient” time step
- Not longer solving the PDEs we write down
- Many FTE-years to recalibrate when discretization changes
- Calibration eats up a big chunk of the IPCC policy timeline

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}.

Family	Stages	Order	Iterations
Crank-Nicolson/Gauss	1	2	3627
Gauss	2	4	2560
Gauss	4	8	1735
Gauss	8	16	1442

- Naive centered-difference discretization

Trade-offs in time integration

- Properties
  - Nonlinear stability (e.g., positivity preservation)
  - Stability along imaginary axis
  - *L*-stability (damping at infinity)
  - Implicitness and reuse
- What is expensive?
  - Function evaluation
  - Operator assembly/preconditioner setup
    - How much can be reused for how long?
  - Implicit solves
    - Can we find better solver algorithm?
    - More effort in setup?
- What is “convergence”?
  - Wave propagation: implicitness useless for convergence *in a norm*
  - Non-norm functionals could be robust

Outlook

- Next up: Algebraic multigrid for tensor product operators
- Technicalities: imaginary rotation in coarse operators (cf. MG for Helmholtz)
- Stochastic Galerkin have some structure
- Is it possible to design methods with well-conditioned $S = X\Lambda X^{-1}$