Some Design Considerations (and a Few Matrix Implementation Details) in PETSc, the Portable, Extensible Toolkit for Scientific Computation

Richard Tran Mills
Argonne National Laboratory

Workshop on Compiler Techniques for Sparse Tensor Algebra
Cambridge, MA

January 26, 2019
Goal is to give this audience a flavor of the design of PETSc, to facilitate discussion of how it and similar libraries can use the techniques being presented at the workshop.

1. High-level: Design principles that have guided PETSc development
2. Lower-level excursion: Matrices in PETSc, and a special case (sparse Kronecker products)
PETSc: The Portable, Extensible Toolkit for Scientific Computation

- [https://www.mcs.anl.gov/petsc/](https://www.mcs.anl.gov/petsc/)
- Open-source “solvers” library that provides linear and nonlinear solvers, distributed-memory parallel data structures, mesh management, time steppers, performance profiling, ...
- Around for over two decades, with several thousand users
- Used in many, many fields: acoustics, aerodynamics, air pollution, arterial flow, bone fractures, brain surgery, cancer surgery, cancer treatment, carbon sequestration, cardiology, cells, CFD, combustion, concrete, corrosion, data mining, dentistry, earthquakes, economics, esophageal transport, fission, fusion, glaciers, groundwater flow, linguistics, mantle convection, magnetic films, materials science, medical imaging, ocean dynamics, oil recovery, PageRank, polymer injection molding, polymeric membranes, quantum computing, seismology, semiconductors, rockets, relativity, surface water flow, ...
- Most common use is solving PDE-based problems, but also used in mathematical optimization (via TAO—now part of PETSc), eigensolvers (via SLEPc, PRIMME), and many other areas.
PFLOTRAN: PETSc-based geologic flow and reactive transport code

- Open-source (download at bitbucket.org/pflotran) code that simulates multiscale-multiphase-multicomponent reacting flows in porous media
- Finite volumes; implicit or operator-split timestepping
- Multiple interacting continua; supercritical CO\(_2\); geomechanics
- Comprehensive biogeochemistry: Ion activity models, ion exchange, aqueous speciation, aqueous-gas mass transfer, mineral precip./dissolution, sorption isotherms, surface complexation (equilibrium, kinetic, multirate), colloids, microbial reactions
- Built from ground-up with parallel scalability in mind; scales to O(100,000) cores on leadership-class supercomputers


http://www.pnl.gov/science/images/highlights/computing/uvi.jpg

Elements of the PETSc Design Philosophy

Three goals of PETSc:

1. Provide a simple and consistent way to specify algebraic systems (in general, nonlinear and
time-dependent)
   - Enable experimentation with diverse algorithms and implementations without requiring premature
     commitment to particular data structures and solvers.
2. Provide scalable, powerful parallel algebraic solvers, suitable for a variety of physical models
   - Time has shown that solutions to goal 1 greatly facilitate goal 2!
3. Support extensibility to allow the use of powerful solvers developed by other groups

Some principles guiding our development work:

- Defer algorithmic choices until execution time, and enable complex composition of multi-layered
  solvers via runtime options
- Strive to separate control logic from computational kernels
  - Allow injecting new hardware-specific computational kernels without having to rewrite the entire solver
    software library
- Hand-optimize small kernels only, and design to maximize reuse of such kernels
  - Cf. the BLIS framework, which expresses all level-3 BLAS operations in terms of one micro-kernel.
- Reuse existing, specialized libraries (e.g., MKL, cuSPARSE) when feasible
PETSc run-time options example: Composable solvers

Best choice of solver can vary significantly with problem parameters.

Driven cavity example (SNES ex19) runs well with default solvers at Grashof number 1000:

```
./ex19 -da_refine 4 -lidvelocity 100 -grashof 1e3
```

But increasing the Grashof number to 50,000 causes a solver failure.

One of the best solvers we have found for this case is a multiplicative combination of full-approximation scheme (nonlinear multigrid) combined with a Newton-Krylov-multigrid solver:

```
./ex19 -da_refine 4 -lidvelocity 100 -grashof 5e4 -snes_type composite
  -snes_composite_type multiplicative -snes_composite_sneses fas,newtonls
  -sub_0_fas_levels_snes_type ngs -sub_0_fas_levels_snes_max_it 6
  -sub_0_fas_coarse_snes_linesearch_type basic -sub_1_snes_linesearch_type basic
  -sub_l_pc_type mg
```

Very difficult to find such combinations without being able to experiment via run-time options!
PETSc Matrices I

The PETSc $\text{Mat}$ has a single user interface:

- **Matrix assembly**: `MatSetValues()`
- **Matrix-vector multiplication**: `MatMult()`
- **Matrix-matrix multiplication**: `MatMatMult()`

But multiple underlying implementations:

- AIJ, block AIJ, symmetric block AIJ
- AIJPERM, SELL (sliced ELLPACK), AIJSELL
- Dense, Elemental
- Intel MKL, cuSPARSE, ViennaCL
- Matrix-free
- Sparse Kronecker product (KAIJ)
- ...
And each matrix type may have multiple implementations of key kernels:

- AIJ MatMult() has standard and hand-coded AVX/AVX-512 implementations
- block AIJ has many hand-unrolled implementations for different block sizes.
- Currently **eight** different algorithms implemented for sparse matrix-matrix products in AIJ!
PETSc run-time options example: Different matrix back-ends

Solve solid fuel ignition (SNES ex5) example with algebraic multigrid (GAMG), using AIJ matrices:

```bash
mpirun -n $N ./ex5 -da_refine 9 -pc_type gamg -pc_mc_levels $NLEVELS
  -mg_levels_pc_type jacobi
```

Same problem and solvers, using AIJ for GAMG mesh setup, but sliced-ELLPACK for numerical setup and solve phases (Jacobi smoothing, coarse grid restriction and interpolation):

```bash
mpirun -n $N ./ex5 -da_refine 9 -pc_type gamg -pc_mc_levels $NLEVELS
  -mg_levels_pc_type jacobi -mat_seqaij_type seqaijSELL
```

Use GPU via ViennaCL’s OpenCL backend for numerical setup and solve phases:

```bash
mpirun -n $N ./ex5 -da_refine 9 -pc_type gamg -pc_mc_levels $NLEVELS
  -mg_levels_pc_type jacobi -dm_mat_type aijviennacl -dm_vec_type viennacl
  -viennacl_backend opencl
```

Use ViennaCL’s OpenMP backend for numerical setup and solve phases:

```bash
mpirun -n $N ./ex5 -da_refine 9 -pc_type gamg -pc_mc_levels $NLEVELS
  -mg_levels_pc_type jacobi -dm_mat_type aijviennacl -dm_vec_type viennacl
  -viennacl_backend openmp
```
The AIJ/CSR Storage Format

Default AIJ matrix format (compressed sparse row) in PETSc is versatile, but can be poor for SIMD.

Two main disadvantages with AIJ representation:

1. Poor utilization of SIMD units when number of nonzero (nnz) elements in a row is less than the register length, or when nnz modulo register length is small and positive.
2. Sparsity pattern can lead to poor data locality in input vector.
Sliced ELLPACK-based storage formats

- To enable better vectorization, we have added the MATSELL sparse matrix class (`-mat.type sell`), which uses a sliced ELLPACK representation.
- Supports sparse matrix-vector multiplication, Jacobi and SOR smoothers.
- Provide AVX and AVX-512 intrinsics implementations.

<table>
<thead>
<tr>
<th>Matrix</th>
<th># nonzeros</th>
<th>Column indices</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 0 0 B 0 0 0 0</td>
<td>2</td>
<td>0 3</td>
<td>A B</td>
</tr>
<tr>
<td>0 C D 0 0 0 0 0</td>
<td>2</td>
<td>1 2</td>
<td>C D</td>
</tr>
<tr>
<td>0 0 0 0 0 0 E 0</td>
<td>1</td>
<td>6 *</td>
<td>E</td>
</tr>
<tr>
<td>0 F 0 G 0 0 0 0</td>
<td>2</td>
<td>1 3</td>
<td>F G</td>
</tr>
<tr>
<td>0 H 0 I J 0 0 0</td>
<td>3</td>
<td>1 3 4</td>
<td>H I J</td>
</tr>
<tr>
<td>K 0 L 0 0 0 0 0</td>
<td>2</td>
<td>0 2 *</td>
<td>K L</td>
</tr>
<tr>
<td>M 0 0 0 0 N 0 0</td>
<td>2</td>
<td>0 5 *</td>
<td>M N</td>
</tr>
<tr>
<td>0 0 0 O 0 P 0 Q</td>
<td>3</td>
<td>3 5 7</td>
<td>O P Q</td>
</tr>
</tbody>
</table>

- Slice height
- Padding
Figure: Roofline plot generated using the Empirical Roofline Tool from LBNL on ALCF Theta. The MATSELL SpMV routine achieves performance close to the roofline when running out of the KNL MCDRAM.
Figure: SpMV performance on three generations of Xeon Processors and KNL. SELL provides the best performance on KNL but offers only marginal gains on Xeons. Note that some AVX implementations outperform AVX2 implementations!
Motivating Sparse Kronecker Product Matrices: Runge-Kutta methods

(This discussion courtesy of Jed Brown, CU Boulder)

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

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

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

- 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

<table>
<thead>
<tr>
<th></th>
<th>(\frac{5}{36})</th>
<th>(\frac{2}{9})</th>
<th>(\frac{5}{36})</th>
<th>(\frac{5}{30})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\frac{1}{2} - \sqrt{\frac{15}{10}})</td>
<td>(\frac{5}{36})</td>
<td>(+\sqrt{\frac{15}{24}})</td>
<td>(\frac{5}{30})</td>
<td>(+\sqrt{\frac{15}{24}})</td>
</tr>
<tr>
<td>(\frac{1}{2} - \sqrt{\frac{15}{10}})</td>
<td>(\frac{5}{36})</td>
<td>(+\sqrt{\frac{15}{30}})</td>
<td>(\frac{2}{9})</td>
<td>(+\sqrt{\frac{15}{15}})</td>
</tr>
<tr>
<td>(\frac{1}{2} - \sqrt{\frac{15}{10}})</td>
<td>(\frac{5}{36})</td>
<td>(+\sqrt{\frac{15}{30}})</td>
<td>(\frac{2}{9})</td>
<td>(+\sqrt{\frac{15}{15}})</td>
</tr>
<tr>
<td>(\frac{1}{2} - \sqrt{\frac{15}{10}})</td>
<td>(\frac{5}{36})</td>
<td>(+\sqrt{\frac{15}{30}})</td>
<td>(\frac{2}{9})</td>
<td>(+\sqrt{\frac{15}{15}})</td>
</tr>
<tr>
<td>(\frac{1}{2} - \sqrt{\frac{15}{10}})</td>
<td>(\frac{5}{36})</td>
<td>(+\sqrt{\frac{15}{30}})</td>
<td>(\frac{2}{9})</td>
<td>(+\sqrt{\frac{15}{15}})</td>
</tr>
</tbody>
</table>

Butcher tableau for 6th order Gauss-Legendre method

- 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 $u^*$
  \[ Y = u^n + hAF(Y) \]

  \[ [l_s \otimes I_n + hA \otimes J(u^*)] \delta Y = \text{RHS} \]

- 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 $S = V \Lambda V^{-1}$
  \[ \Lambda \otimes I_n + I_s \otimes J \]
  \[ s \text{ decoupled solves} \]
  \[ \text{Complex eigenvalues (overhead for real problem)} \]
Ill conditioning

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

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
\[ G = I_n \otimes S + J \otimes T \]

- \( J \) is parallel and sparse (AIJ storage), \( S \) 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
- Same structure for stochastic Galerkin and other UQ methods
- Implementation is working in parallel and soon to be merged to master branch; still needs performance optimization.