Argonne NATIONAL LABORATORY

Multiphysics Simulations with PETSc (and a few PFLOTRAN examples) on Novel Manycore Computer Architectures

Richard Tran Mills, Argonne National Laboratory

(with contributions from Matt Knepley, U. Buffalo; Karl Rupp, TU Wien; and Hong Zhang, ANL)

SIAM Conference on Computational Science and Engineering (CSE19)

Spokane, WA — February 27, 2019

Agenda

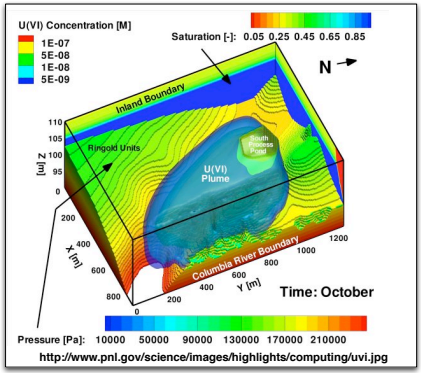
To give this audience an idea of the design philosophy behind PETSc, and how its strategy of enabling **runtime composability** facilitates experimentation with both

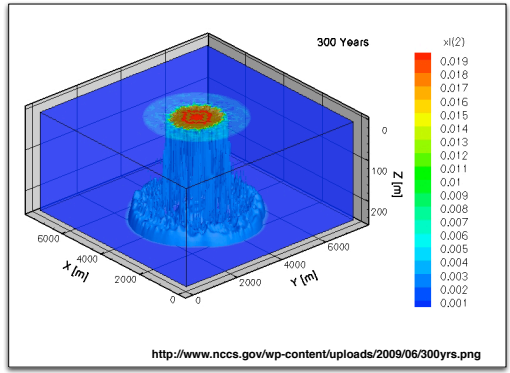
1. physics-based solver strategies and
2. low-level details of data structures, compute kernels, and mapping to hardware

to enable effective utilization of increasingly complex modern high-performance computing (HPC) architectures for solving multiphysics simulation problems.

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





PETSc: The Portable, Extensible Toolkit for Scientific Computation

- `https://www.mcs.anl.gov/petsc/`
- Open-source “solvers” library that provides linear and nonlinears 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, tarbon 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 membrances, 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.

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_1_pc_type mg
```

Being able to experiment via run-time options is crucial for finding such combinations!

PETSc FieldSplit: Composing Multi-Physics Solvers

Idea behind PETSc FieldSplit:

- For multi-physics problems, one cannot simply hand off a sparse matrix to a solver and expect good performance.
- Instead, user describes the (logical block) structure of the matrix and then specifies how to compose an outer linear solver based on inner linear solvers associated with blocks and Schur complements.

FieldSplit component in PETSc provides

- Analysis
  - Use **IS**es to define fields
  - Decouples **PC** from problem definition
- Synthesis
  - Additive, Multiplicative, Schur
  - Commutes with Multigrid

FieldSplit Customization

- Analysis
  - `-pc_fieldsplit_<split num>_fields 2,1,5`
  - `-pc_fieldsplit_detect_saddle_point`
- Synthesis
  - `-pc_fieldsplit_type <additive, multiplicative, symmetric_multiplicative, schur>`
  - `-pc_fieldsplit_diag_use_amat`
    `-pc_fieldsplit_off_diag_use_amat`
    Use diagonal blocks of operator to build PC
- Schur complements
  - `-pc_fieldsplit_schur_precondition <user,a11,full,self,selfp>`
    How to build preconditioner for S
  - `-pc_fieldsplit_schur_factorization_type <diag,lower,upper,full>`
    Which off-diagonal parts of the block factorization to use

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Block-Jacobi (Exact), Cohouet & Chabard, IJNMF, 1988.

```
-ksp_type gmres -pc_type fieldsplit -pc_fieldsplit_type additive
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} A & 0 \\ 0 & I \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Block-Jacobi (Inexact), Cohouet & Chabard, IJNMF, 1988.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type additive
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & I \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Gauss-Seidel (Inexact), Elman, DTIC, 1994.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type multiplicative
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} \hat{A} & B \\ 0 & I \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Gauss-Seidel (Inexact), Elman, DTIC, 1994.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type multiplicative
 -pc_fieldsplit_0_fields 1 -pc_fieldsplit_1_fields 0
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} I & B^T \\ 0 & \hat{A} \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Diagonal Schur Complement, Olshanskii, et.al., Numer. Math., 2006.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type diag
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type minres  -fieldsplit_pressure_pc_type none
```

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & -\hat{S} \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Lower Schur Complement, May and Moresi, PEPI, 2008.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type lower
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type minres  -fieldsplit_pressure_pc_type none
```

$$\begin{pmatrix} \hat{A} & 0 \\ B^T & \hat{S} \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Upper Schur Complement, May and Moresi, PEPI, 2008.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type upper
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type minres  -fieldsplit_pressure_pc_type none
```

$$\begin{pmatrix} \hat{A} & B \\ & \hat{S} \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Uzawa Iteration, Uzawa, 1958

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type upper
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_type richardson -fieldsplit_pressure_pc_type jacobi
 -fieldsplit_pressure_ksp_max_it 1
```

$$\begin{pmatrix} A & B \\ & \hat{S} \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Full Schur Complement, Schur, 1905.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_rtol 1e-10 -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

SIMPLE, Patankar and Spalding, IJHMT, 1972.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_rtol 1e-10 -fieldsplit_pressure_pc_type jacobi
 -fieldsplit_pressure_inner_ksp_type preonly
  -fieldsplit_pressure_inner_pc_type jacobi
 -fieldsplit_pressure_upper_ksp_type preonly
  -fieldsplit_pressure_upper_pc_type jacobi
```

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & B^T D_A^{-1} B \end{pmatrix} \begin{pmatrix} I & D_A^{-1} B \\ 0 & I \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Least-Squares Commutator, Kay, Loghin and Wathen, SISC, 2002.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
-pc_fieldsplit_schur_precondition self
 -fieldsplit_velocity_ksp_type gmres -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_rtol 1e-5 -fieldsplit_pressure_pc_type lsc
```

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & \hat{S}_{\text{LSC}} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex31: P_2/P_1 Stokes Problem with Temperature on Unstructured Mesh

Additive Schwarz + Full Schur Complement, Elman, Howle, Shadid, Shuttleworth, and Tuminaro, SISC, 2006.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type additive
-pc_fieldsplit_0_fields 0,1 -pc_fieldsplit_1_fields 2
 -fieldsplit_0_ksp_type fgmres -fieldsplit_0_pc_type fieldsplit
 -fieldsplit_0_pc_fieldsplit_type schur
 -fieldsplit_0_pc_fieldsplit_schur_factorization_type full
   -fieldsplit_0_fieldsplit_velocity_ksp_type preonly
   -fieldsplit_0_fieldsplit_velocity_pc_type lu
   -fieldsplit_0_fieldsplit_pressure_ksp_rtol 1e-10
   -fieldsplit_0_fieldsplit_pressure_pc_type jacobi
 -fieldsplit_temperature_ksp_type preonly
 -fieldsplit_temperature_pc_type lu
```

$$\begin{pmatrix} \begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} \hat{A} & 0 \\ 0 & \hat{S} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix} & 0 \\ 0 & L_T \end{pmatrix}$$

FieldSplit Solver Configuration: No New Code

SNES ex31: P_2/P_1 Stokes Problem with Temperature on Unstructured Mesh

Upper Schur Comp. + Full Schur Comp. + Least-Squares Comm.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_0_fields 0,1 -pc_fieldsplit_1_fields 2
-pc_fieldsplit_schur_factorization_type upper
 -fieldsplit_0_ksp_type fgmres -fieldsplit_0_pc_type fieldsplit
 -fieldsplit_0_pc_fieldsplit_type schur
 -fieldsplit_0_pc_fieldsplit_schur_factorization_type full
   -fieldsplit_0_fieldsplit_velocity_ksp_type preonly
   -fieldsplit_0_fieldsplit_velocity_pc_type lu
   -fieldsplit_0_fieldsplit_pressure_ksp_rtol 1e-10
   -fieldsplit_0_fieldsplit_pressure_pc_type jacobi
 -fieldsplit_temperature_ksp_type gmres
 -fieldsplit_temperature_pc_type lsc
```

$$\begin{pmatrix} \begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} \hat{A} & 0 \\ 0 & \hat{S} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix} & G \\ 0 & \hat{S}_{\text{LSC}} \end{pmatrix}$$

Constrained Pressure Residual Preconditioner (CPR) in PFLOTRAN

CPR preconditioners can be constructed with PCCOMPOSITE and PCFIELDSPLIT.

PFLOTRAN oil-water simulation example runtime options:

```
-flow_ksp_type fgmres -flow_pc_type composite
-flow_pc_composite_type multiplicative -flow_pc_composite_pcs fieldsplit,bjacobi
-flow_sub_0_ksp_type fgmres -flow_sub_0_pc_fieldsplit_type additive
-flow_sub_0_pc_fieldsplit_0_fields 0 -flow_sub_0_pc_fieldsplit_1_fields 1,2
-flow_sub_0_fieldsplit_0_ksp_type richardson
-flow_sub_0_fieldsplit_0_pc_type hypre -flow_sub_0_fieldsplit_0_pc_hypre_type boomeramg
-flow_sub_0_fieldsplit_0_pc_hypre_boomeramg_agg_nl 1
-flow_sub_0_fieldsplit_0_pc_hypre_boomeramg_strong_threshold 0.5
-flow_sub_0_fieldsplit_0_pc_hypre_boomeramg_coarsen_type PMIS
-flow_sub_0_fieldsplit_0_pc_hypre_boomeramg_interp_type ext+i
-flow_sub_0_fieldsplit_0_pc_hypre_boomeramg_relax_type_all Jacobi
-flow_sub_0_fieldsplit_1_ksp_type preonly -flow_sub_0_fieldsplit_1_pc_type none
-flow_sub_0_fieldsplit_0_ksp_max_it 10 -flow_sub_1_sub_pc_type ilu
```

(Field 0 is pressure, 1 is saturation, 2 is temperature.)

Part II, Novel HPC Architectures: What is driving current HPC trends?

Moore's Law (1965)

- Moore's Law: Transistor density doubles roughly every two years
- (Slowing down, but reports of its death have been greatly exaggerated.)
- For decades, single core performance roughly tracked Moore's law growth, because smaller transitors can switch faster.

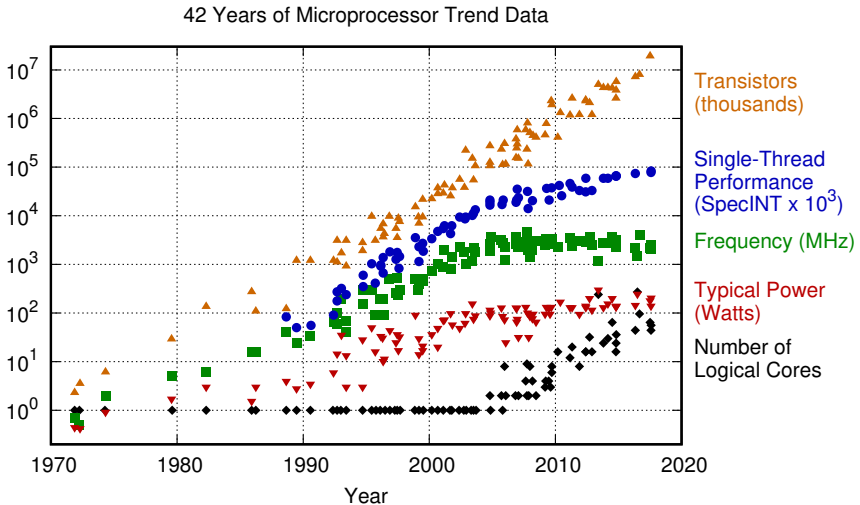
Dennard Scaling (1974)

- Dennard Scaling: Voltage and current are proportional to linear dimensions of a transistor; therefore power is proportional to the area of the transistor.
- Ignores leakage current and threshold voltage; past 65 nm feature size, Dennard scaling breaks down and power density increases, because these don't scale with feature size.

Power Considerations

- The "power wall" has limited practical processor frequencies to around 4 GHz since 2006.
- Increased parallelism (cores, hardware threads, SIMD lanes, GPU warps, etc.) is the current path forward.

Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Current trends in HPC architectures

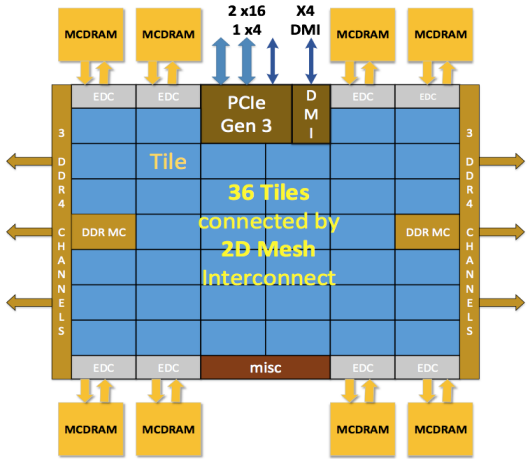
Emerging architectures are very complex...

- Lots of hardware cores, hardware threads
- Wide SIMD registers
- Increasing reliance on fused-multiply-add (FMA), with multiple execution ports, proposed quad FMA instructions
- Multiple memories to manage (multiple NUMA nodes, GPU vs. host, normal vs. high-bandwidth RAM, byte-addressable NVRAM being introduced, ...)
- Growing depth of hierarchies: in memory subsystem, interconnect topology, I/O systems

...and hard to program

- Vectorization may require fighting the compiler, or entirely re-thinking algorithm.
- Must balance vectorization with cache reuse.
- Host vs. offload adds complexity; large imbalance between memory bandwidth on device vs. between host and device
- Growth in peak FLOP rates have greatly outpaced available memory bandwidth.

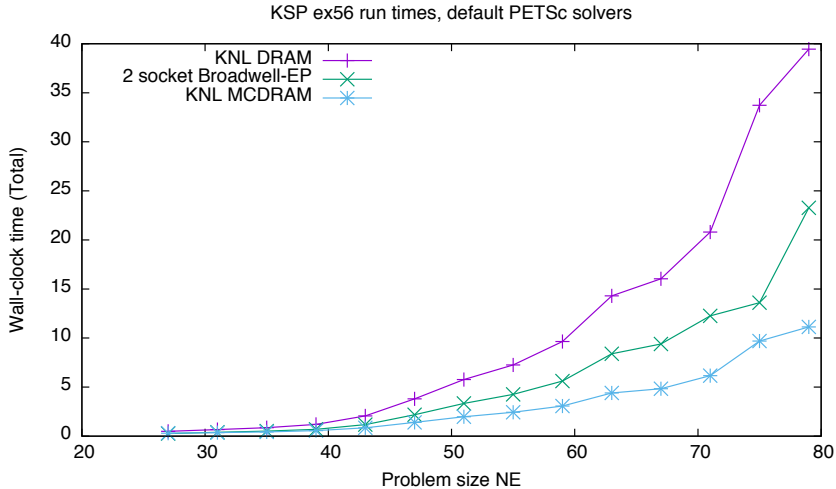
Knights Landing (KNL) Overview — CPU in NERSC Cori



- Standalone CPU—no offload bottleneck
- Up to 36 tiles interconnected via 2D mesh
- Tile: 2 cores + 2 VPU/core + 1 MB L2 cache
- Core: Silvermont-based, 4 threads per core, out-of-order execution
- Dual issue; can saturate both VPUs from a single thread
- 512 bit SIMD lanes, AVX-512 instruction set
- High bandwidth memory (MCDRAM) on package: 490+ GB/s bandwidth on STREAM triad
  - Exposed a separate NUMA node in “flat” mode
  - Treated as massive direct-mapped L3 cache in “cache” mode

KSP ex56: Linear Elasticity on KNL

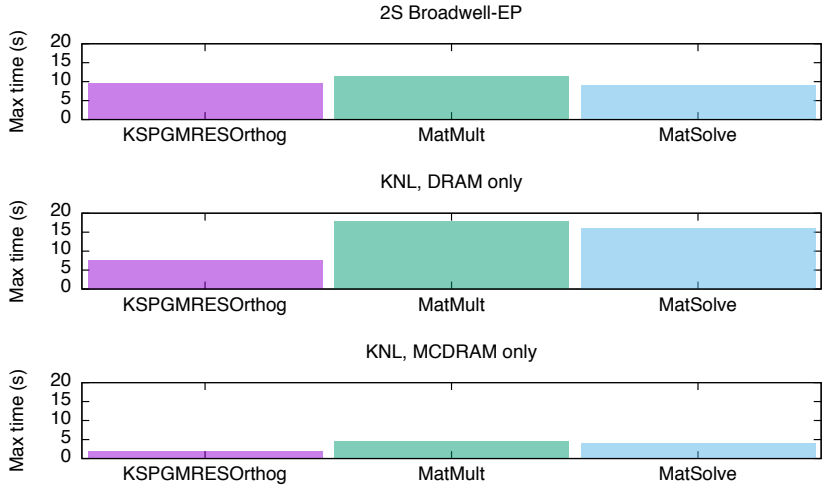
Using PETSc default solvers: GMRES(30), block-Jacobi, ILU(0) on blocks




```
mpirun -n 64 numactl --membind=1 ./ex56 -ne $ne -log_summary
```

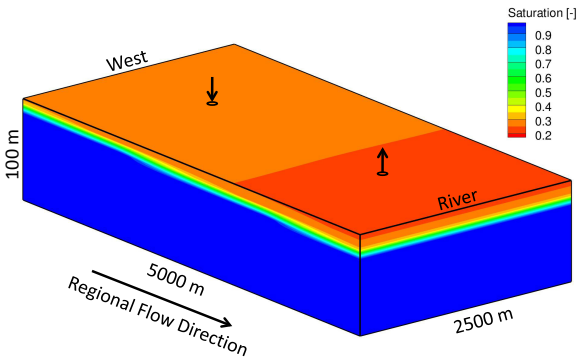
KSP ex56: Linear Elasticity on KNL

Using PETSc default solvers: GMRES(30), block-Jacobi, ILU(0) on blocks



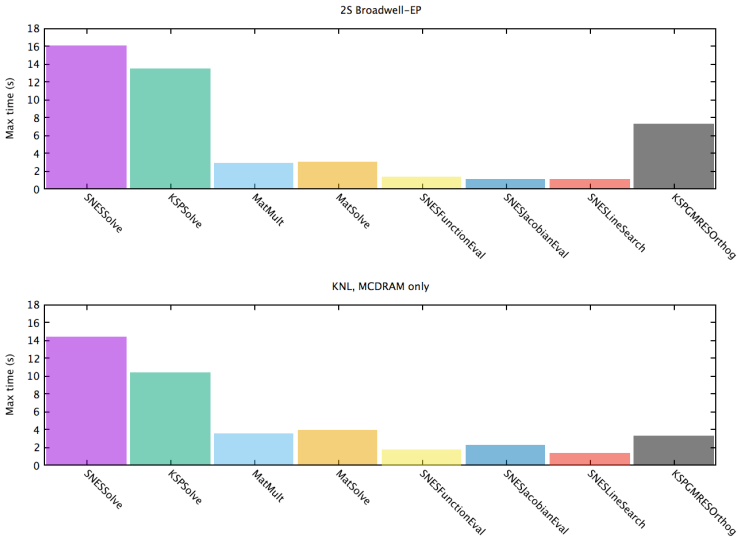

```
mpirun -n 64 numactl --membind=1 ./ex56 -ne 79 -log_summary
```

PFLOTRAN Regional Doublet Simulation: Description and Setup



- PFLOTRAN regional doublet problem analyzed in 2014 *WRR* paper (doi: 10.1002/2012WR013483)
- Variably saturated regional groundwater flow with seasonal river stage fluctuations
- Grid-aligned anisotropy: Vertical permeability order of magnitude lower than lateral
- First order FV in space, backward Euler in time
- Used inexact Newton with GMRES(30) or BCGS, block Jacobi, ILU(0) on blocks
- Used $200 \times 200 \times 100$ grid (4 million total degrees of freedom)

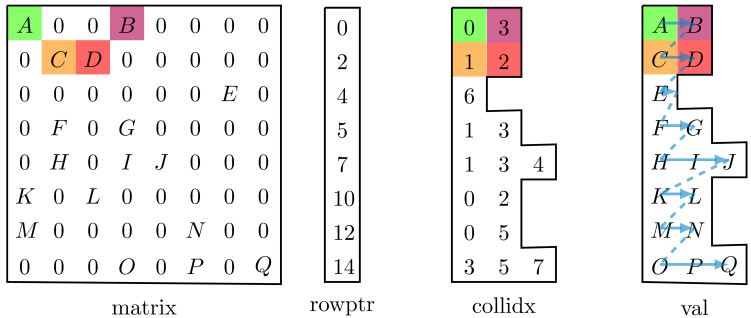
PFLOTRAN Performance on KNL: Out-of-box



- Broadwell (BDW) and KNL times comparable
- Orthogonalizations much faster on KNL
- But MatMult and MatSolve faster on BDW!
- Small work per row ($\sim$ 7 nonzeros; compare to $\sim$ 80 in KSP ex56) perhaps unable to mask latency of gathering x vector; reordering may help.
- Jacobian formation faster on BDW; vectorization work on KNL probably needed.

The AIJ/CSR Storage Format

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

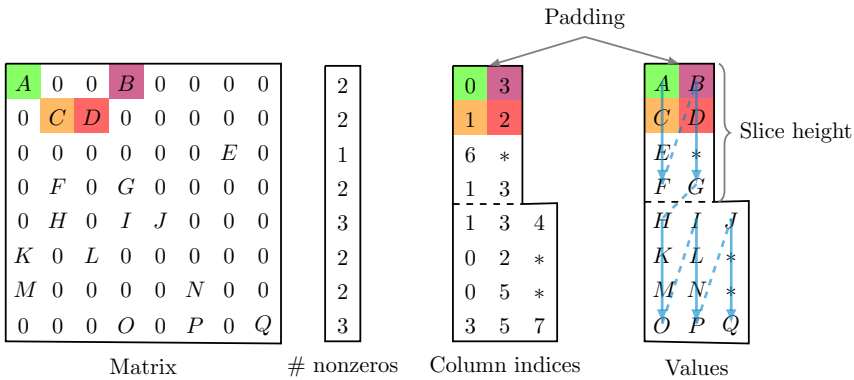


Two main disadvantages with AIJ representation:

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.



Sliced ELLPACK-based storage formats

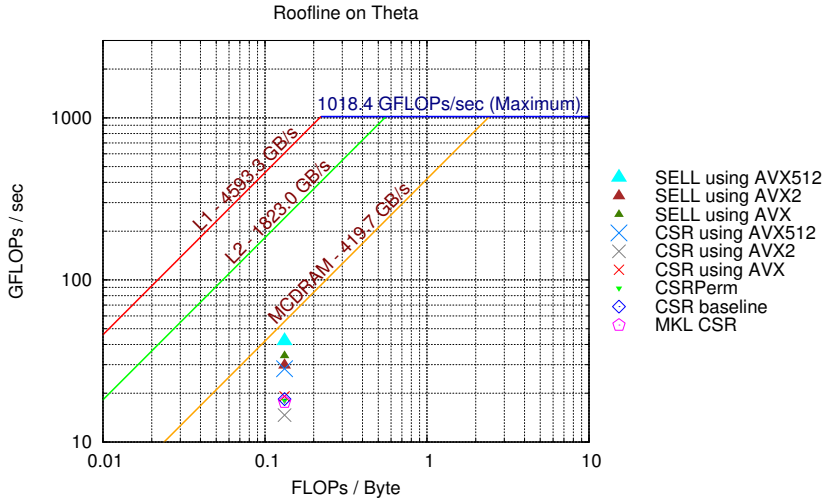


Figure: Roofline plot generated using the Empirical Roofline Tool from LBNL on ALCF Theta. The MATSELL SpMV routine achieves performance close to the roofline when running out of the KNL MCDRAM.

Sliced ELLPACK-based storage formats

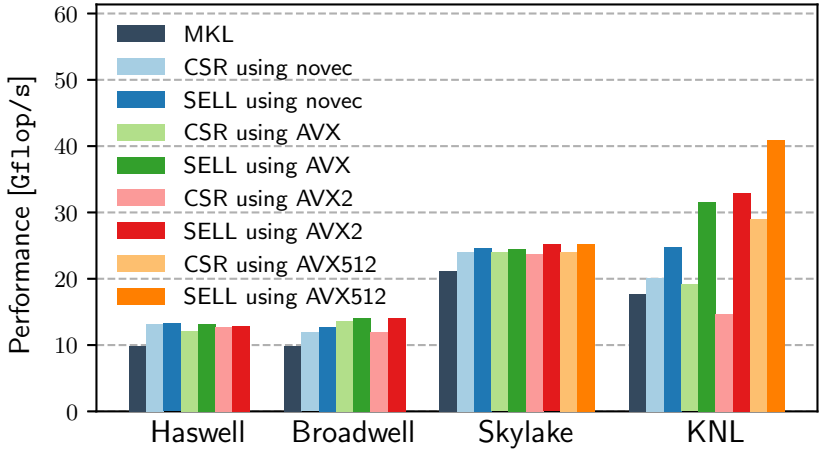


Figure: SpMV performance on three generations of Xeon Processors and KNL. SELL provides the best performance on KNL but offers only marginal gains on Xeons. Note that some AVX implementations outperform AVX2 implementations!

PETSc Matrices

The PETSc `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
- Matrix-free
- Sparse Kronecker product (KAIJ)
- GPU backends: cuSPARSE, ViennaCL
- ...

PETSc run-time options example: Different matrix back-ends

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

```
mpirun -n $N ./ex5 -da_refine 9 -pc_type gamg -pc_mg_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):

```
mpirun -n $N ./ex5 -da_refine 9 -pc_type gamg -pc_mg_levels $NLEVELS
-mg_levels_pc_type jacobi -mat_seqaij_type seqaijsell
```

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

```
mpirun -n $N ./ex5 -da_refine 9 -pc_type gamg -pc_mg_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:

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

Available GPU (or “Accelerator”) Back-Ends

CUDA

- CUDA-support through CUDA/CUSPARSE
- `./configure [...] --with-cuda=1`
- `-vec_type cuda -mat_type aijcusparse`
- Only for NVIDIA GPUs

CUDA/OpenCL/OpenMP

- CUDA/OpenCL/OpenMP-support through ViennaCL
- `./configure [...] --download-viennacl`
- `-vec_type viennacl -mat_type aijviennacl`
- OpenCL on CPUs and MIC fairly poor

How Does It Work?

Host and Device Data

```
struct _p_Vec {
  ...
  void          *data;             // host buffer
  PetscCUDAFlag valid_GPU_array;   // flag
  void          *spptr;            // device buffer
};
```

Possible Flag States

```
  typedef enum {PETSC_CUDA_UNALLOCATED,
                PETSC_CUDA_GPU,
                PETSC_CUDA_CPU,
                PETSC_CUDA_BOTH} PetscCUDAFlag;
```

How Does It Work?

Fallback-Operations on Host

- Data becomes valid on host (`PETSC_CUDA_CPU`)

```
PetscErrorCode VecSetRandom_SeqCUDA_Private(..) {
  VecGetArray(...);
  // some operation on host memory
  VecRestoreArray(...);
}
```

Accelerated Operations on Device

- Data becomes valid on device (`PETSC_CUDA_GPU`)

```
PetscErrorCode VecAYPX_SeqCUDA(..) {
  VecCUDAGetArrayReadWrite(...);
  // some operation on raw handles on device
  VecCUDARestoreArrayReadWrite(...);
}
```

Current GPU-Functionality in PETSc

	CUDA/CUSPARSE	**ViennaCL**
Programming Model	CUDA	CUDA/OpenCL/OpenMP
Operations	Vector, MatMult	Vector, MatMult
Matrix Formats	CSR, ELL, HYB	CSR
Preconditioners	ILU0	SA/Agg-AMG, Par-ILU0
MPI-related	Scatter	-

Additional Functionality

- MatMult via cuSPARSE
- OpenCL residual evaluation for PetscFE
- GPU support for SuperLU-dist
- GPU support for SuiteSparse

GPU Support in GAMG Algebraic Multigrid

New! Native PETSc GAMG algebraic multigrid support

- Specify `aijcusparse` or `aijviennacl` matrix types, then numerical setup and and solve phases (Chebyshev/Jacobi smoothing, coarse grid restriction and interpolation) will run on GPU.
  - E.g., `mpirun -n $N ./ex5 -da_refine 9 -pc_type gamg -pc_mg_levels $NLEVELS -mg_levels_pc_type jacobi -dm_mat_type aijviennacl -dm_vec_type viennacl`
- Similar approach used with AIJMKL and AIJSELL matrix types to use MKL or sliced-ELLPACK back-ends optimized for manycore/SIMD CPUs during solve phase.
  - E.g., `mpirun -n $N ./ex5 -da_refine 9 -pc_type gamg -pc_mg_levels $NLEVELS -mg_levels_pc_type jacobi -mat_seqaij_type seqaijsell`

Performance of GAMG with this usage model is largely unexplored.
We welcome feedback to help us improve its performance!

GPU Pitfalls

GPUs are not a silver bullet! Pitfalls to be aware of:

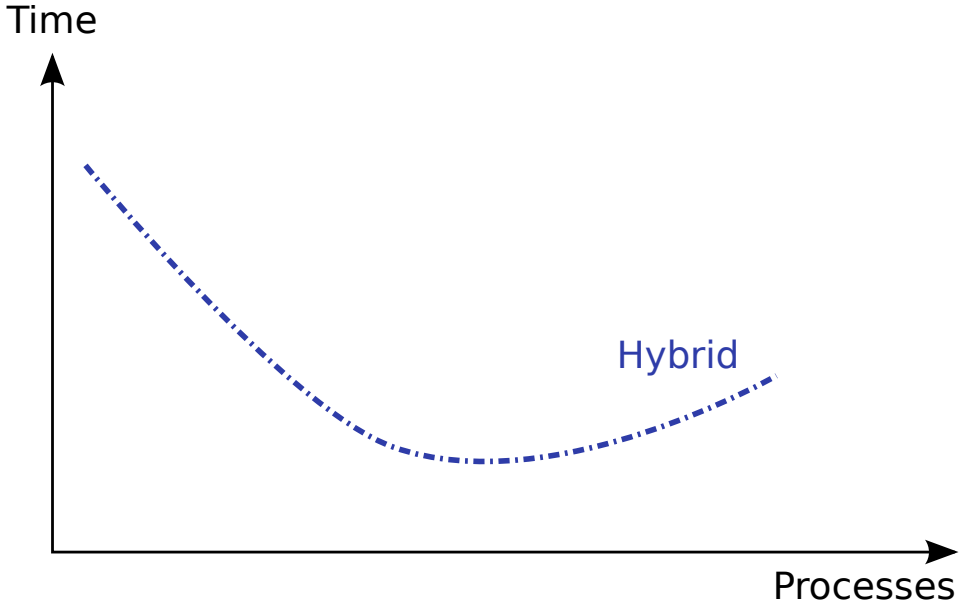
1. Imbalance in on-device memory bandwidth vs. host-GPU bandwidth.
   - E.g., 720 GB/sec from GPU-RAM, 16 GB/sec for PCI-Express. 40X imbalance!
2. Wrong data sizes:
   - Data too small: Kernel launch latencies dominate
   - Data too big: Out of memory

Many Non-Trivial PETSc Operations

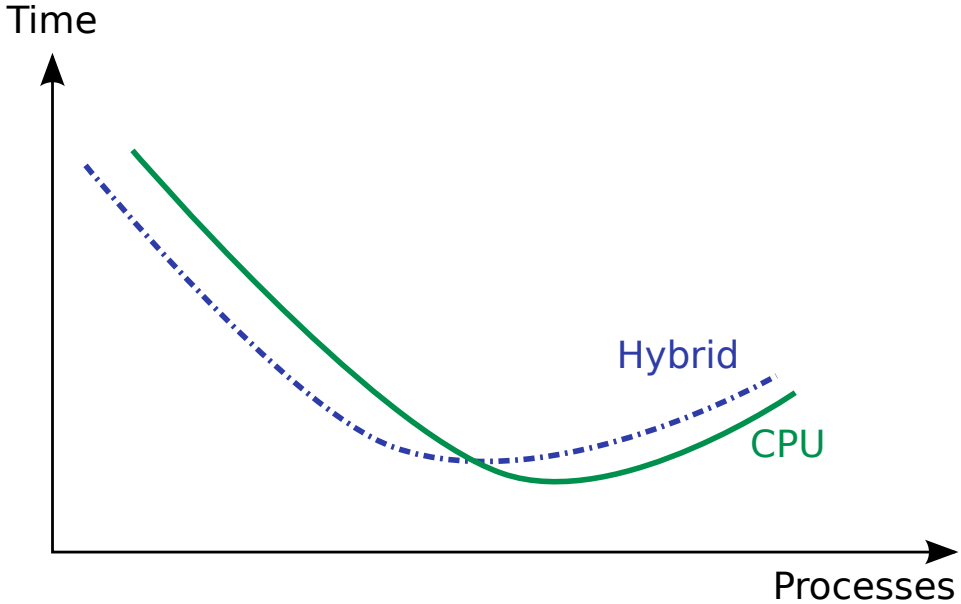
do NOT benefit from modern high-end GPUs

in a substantial way!

GPU Pitfalls: Strong Scaling Implications



GPU Pitfalls: Strong Scaling Implications



Summary

For complex multi-physics problems running on cutting-edge HPC hardware, it is not necessarily possible to know *a priori* what solver algorithms are most appropriate, what data structures and compute kernels should be employed, and how the computation should be mapped to hardware resources.

PETSc enables experimentation at runtime with composable, hierarchical solver algorithms *and* low-level implementation choices, so that appropriate choices combining good algorithmic efficiency with appropriateness to the underlying hardware can be identified.

Please experiment and give us your feedback!
{petsc-maint,petsc-users,petsc-dev}@mcs.anl.gov.

Acknowledgments: This material is based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under Contract DE-AC02-06CH11357 and the Exascale Computing Project (Contract No. 17-SC-20-SC). This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and resources of the Argonne Leadership Computing Facility, a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.